

Job-tree

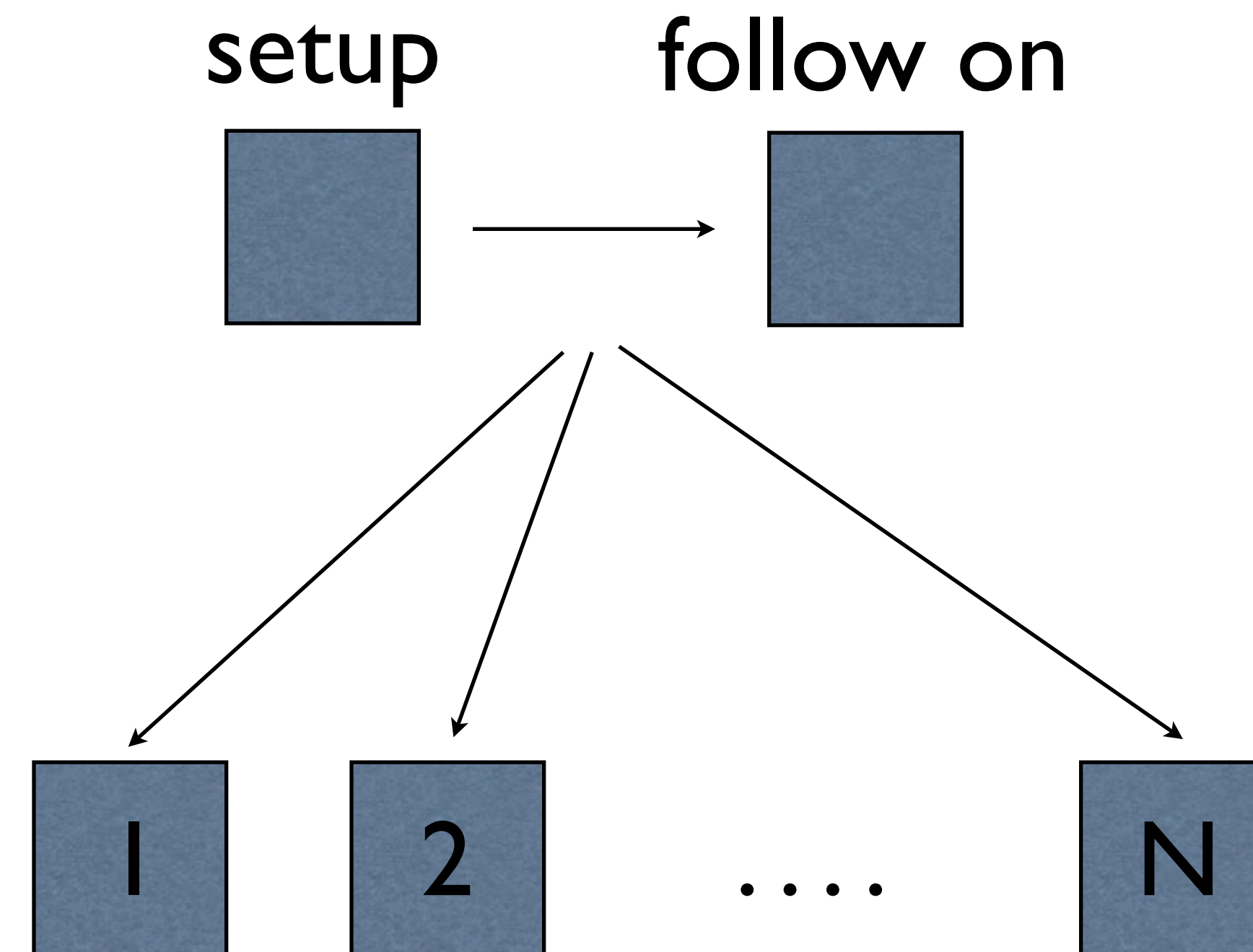
benedict@soe.ucsc.edu

First off...

- Get it at github...
- <https://benedictpaten@github.com/benedictpaten/jobTree.git>
- <git@github.com:benedictpaten/jobTree.git>

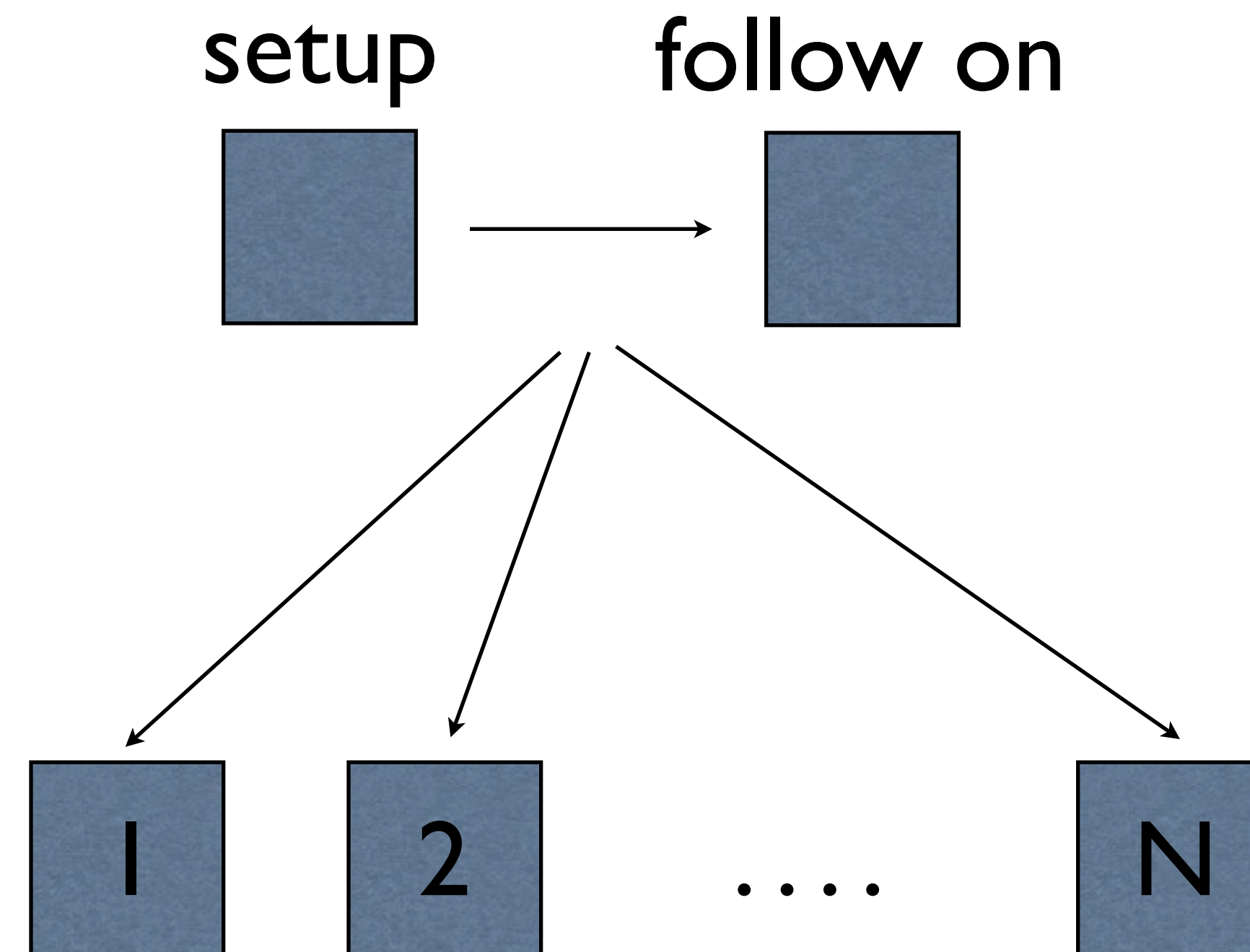
Traditional Batch System

- Composed of a series of 'jobs'
- Single 'setup' job
- Parallel 'child' job
- Single 'follow on' job



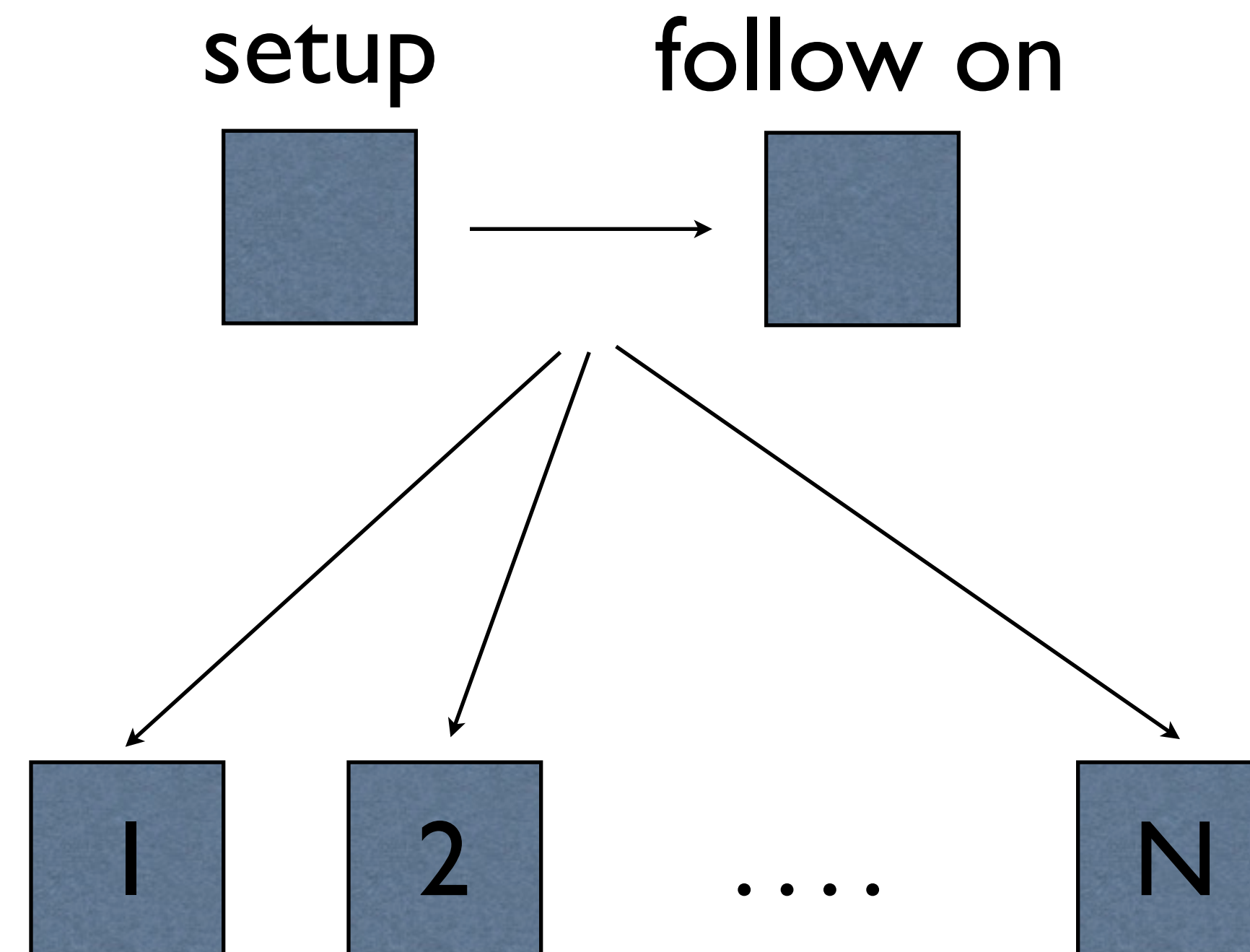
Traditional Batch System

- Setup creates inputs for children
- Children create output
- Follow on aggregates output



Traditional Batch System

- Jobs can get lost at any point during their runtime.
- To maintain 'atomicity' jobs must always be restartable, therefore:
 - No job must alter its own input.
 - The follow on 'cleans up' the setup job



Traditional Batch System Summary

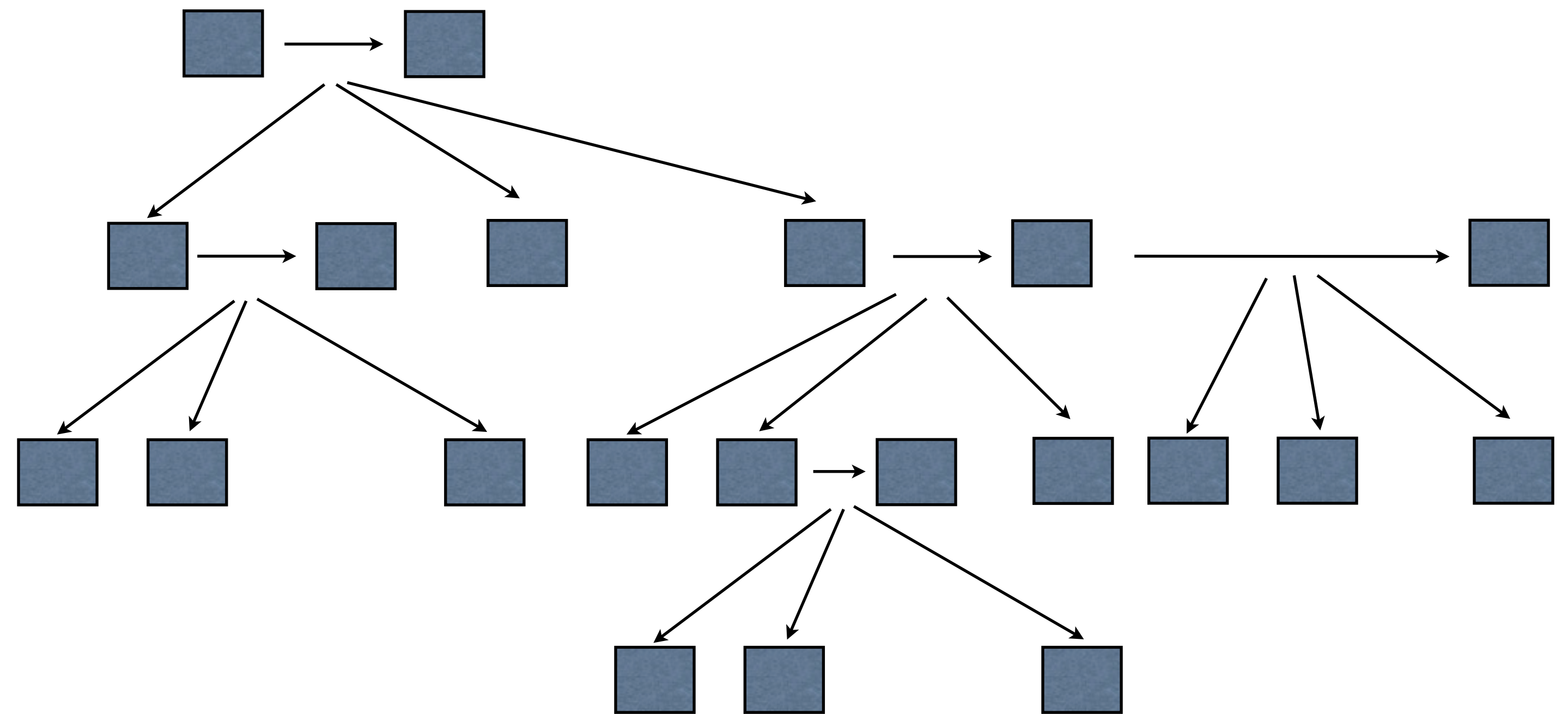
- Traditional batch system can be described, from the users point of view, as a parallel ‘for’ loop.
- Robust pipelines must maintain atomicity

Job-tree

- For loops are useful, but what if..
 - I want to further parallelise a ‘child’ job?
 - I want to do this selectively?
 - I want to do this dynamically and recursively?

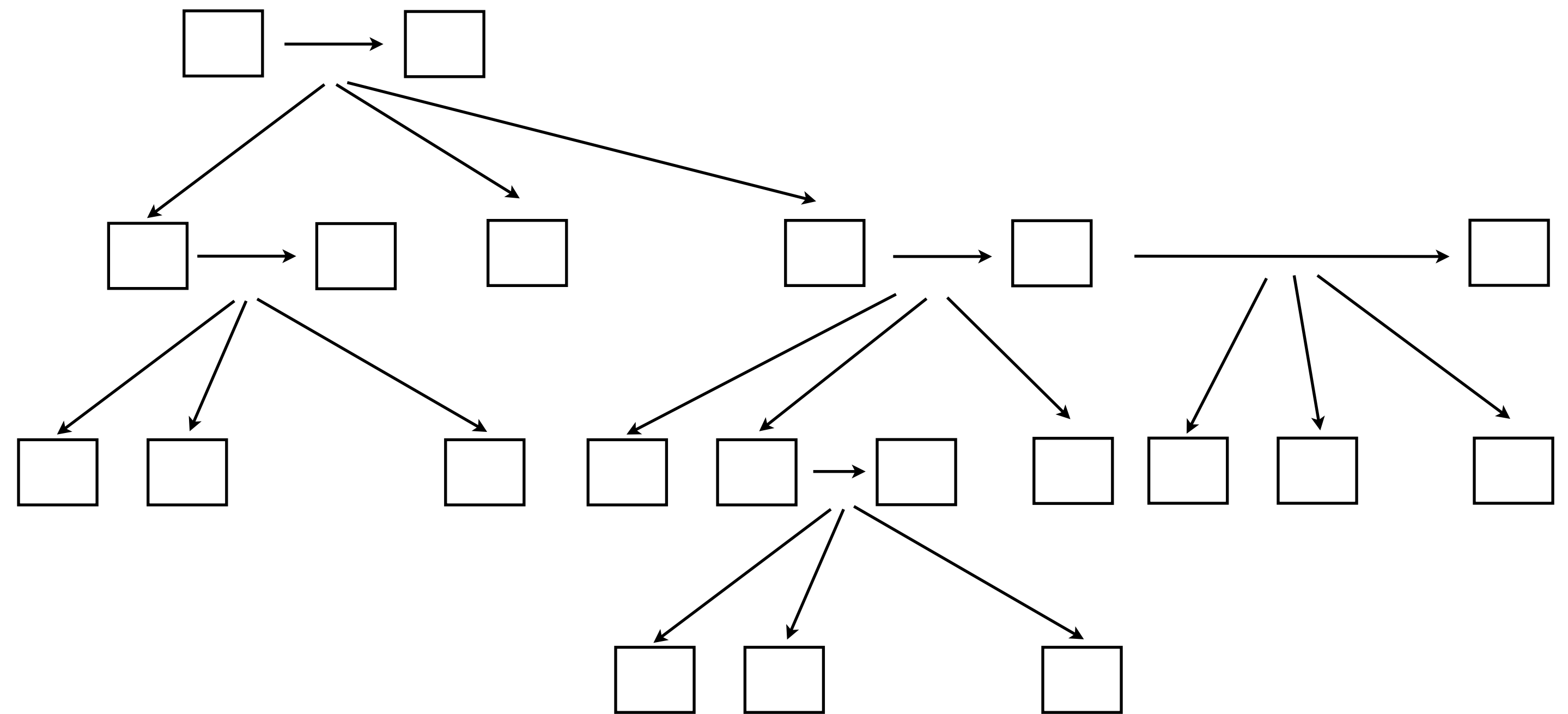
Job-tree

- Job-tree allows you to create arbitrary ‘job-trees’



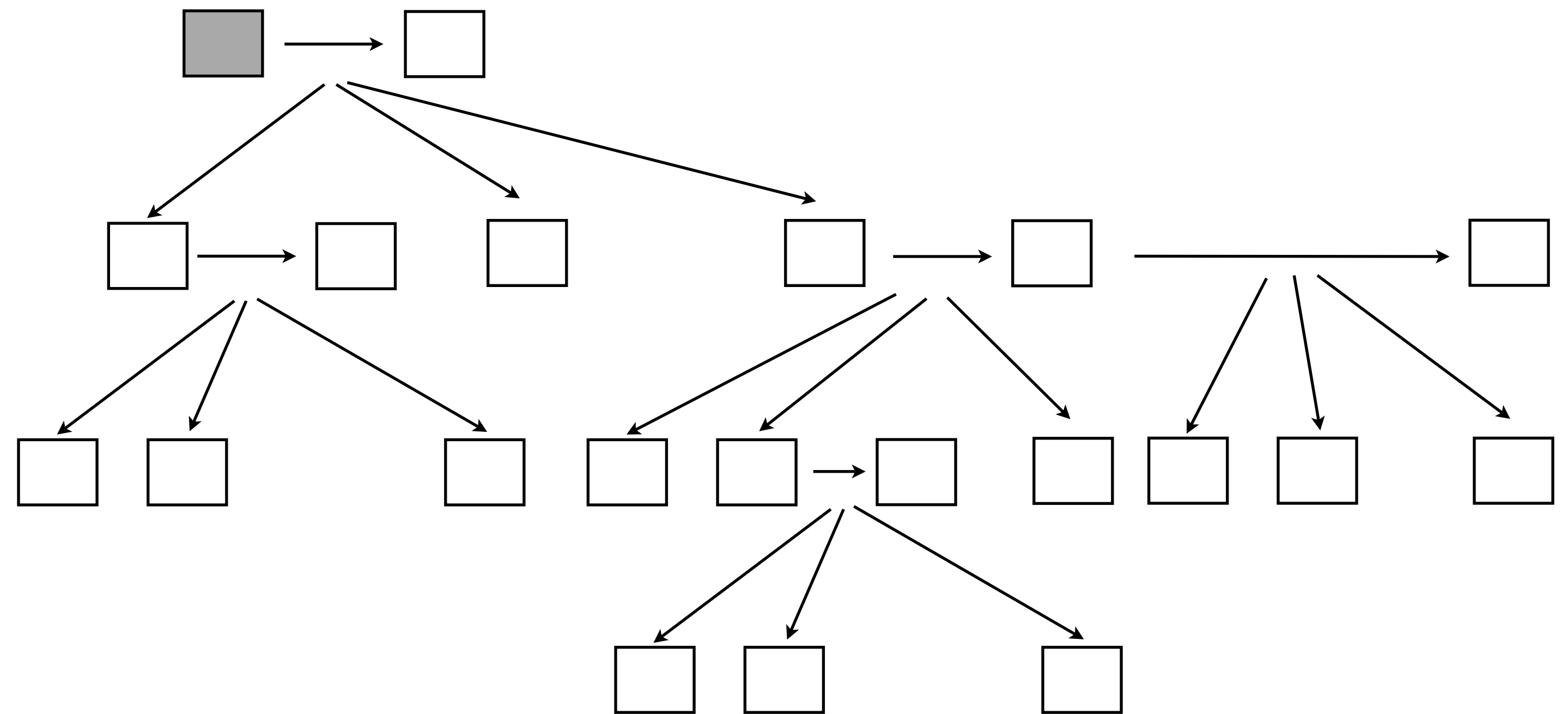
Job-tree

- Let's walk through an example.



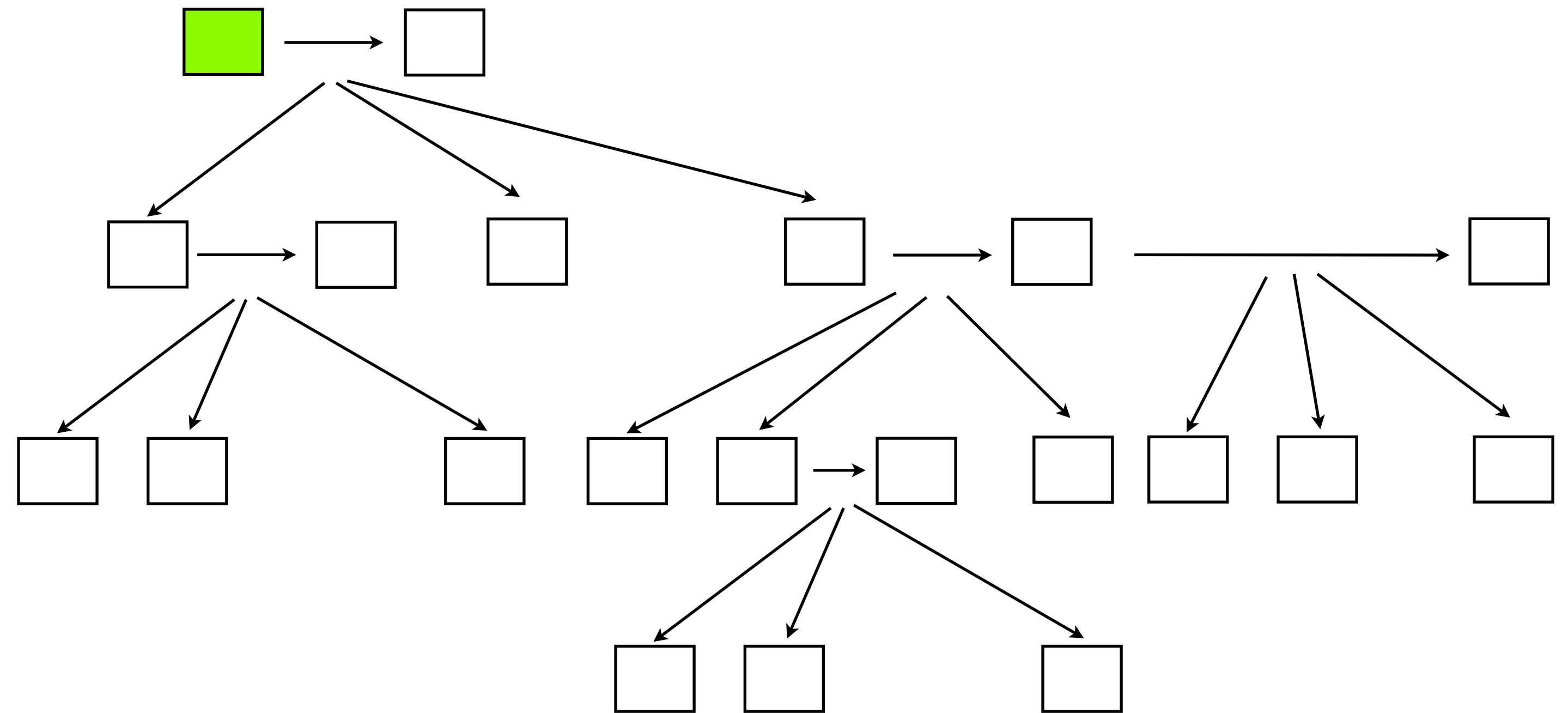
Job-tree

- The first job is issued to the system (grey)
- White boxes are jobs not yet created.



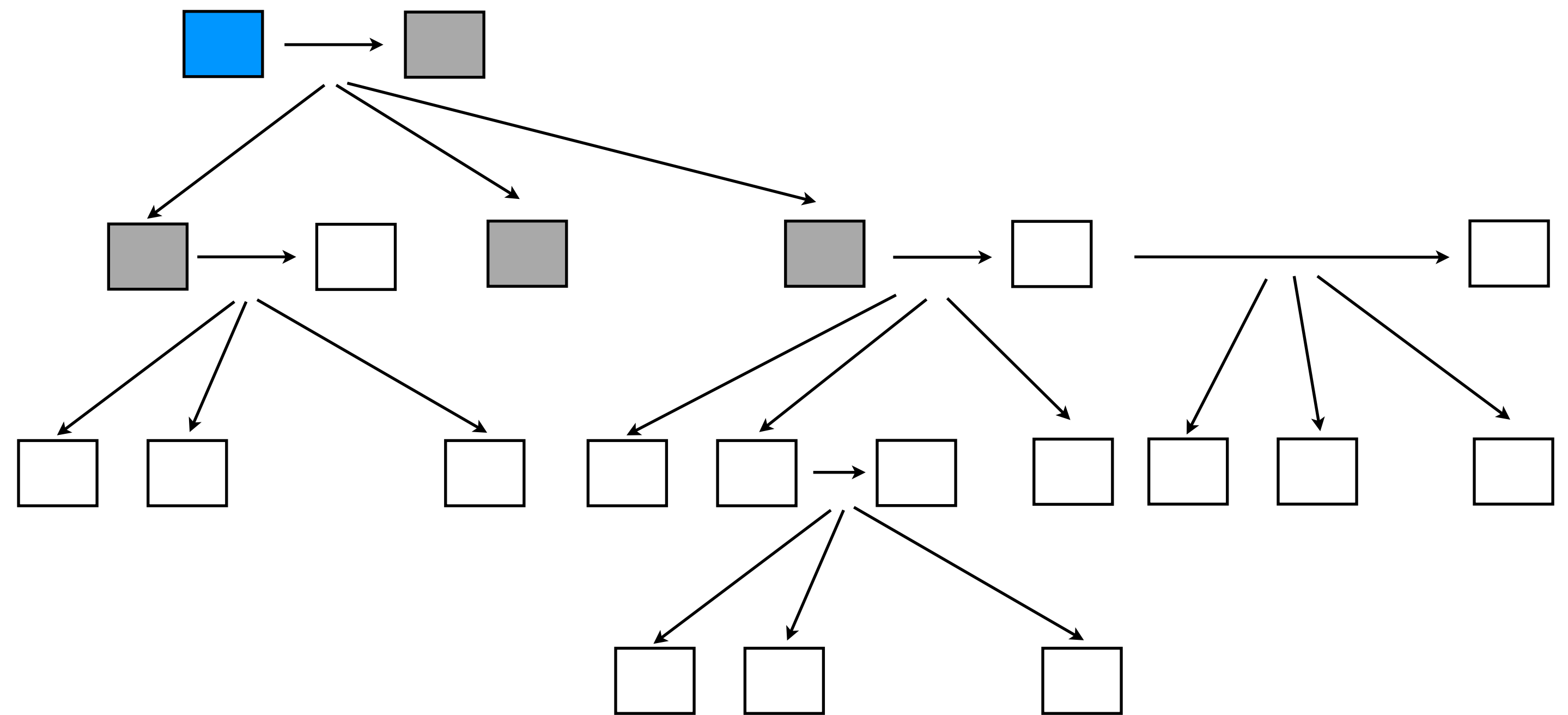
Job-tree

- The first job is run (green)



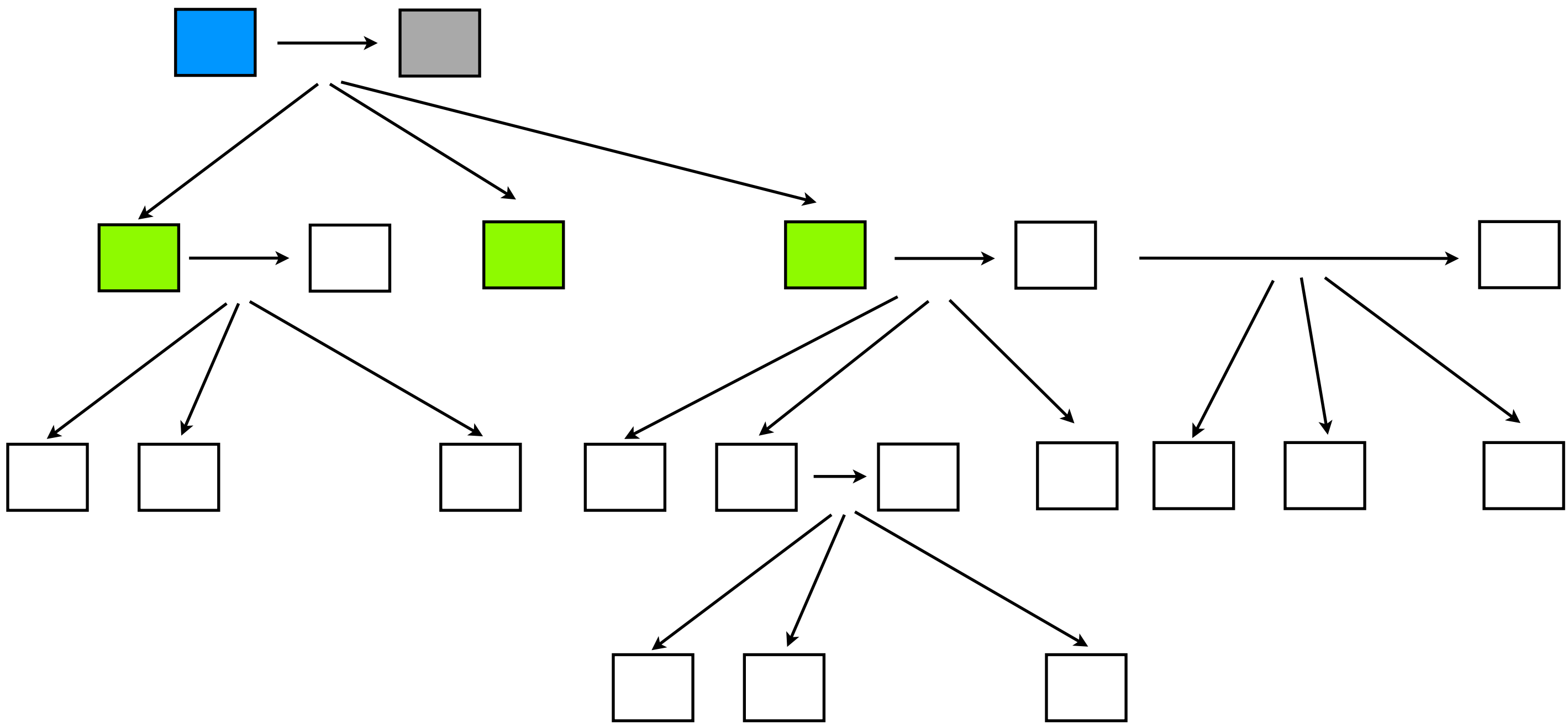
Job-tree

- It completes successively and creates some children and a follow on.
- It is blue, indicating it has children/follow ons not yet complete.



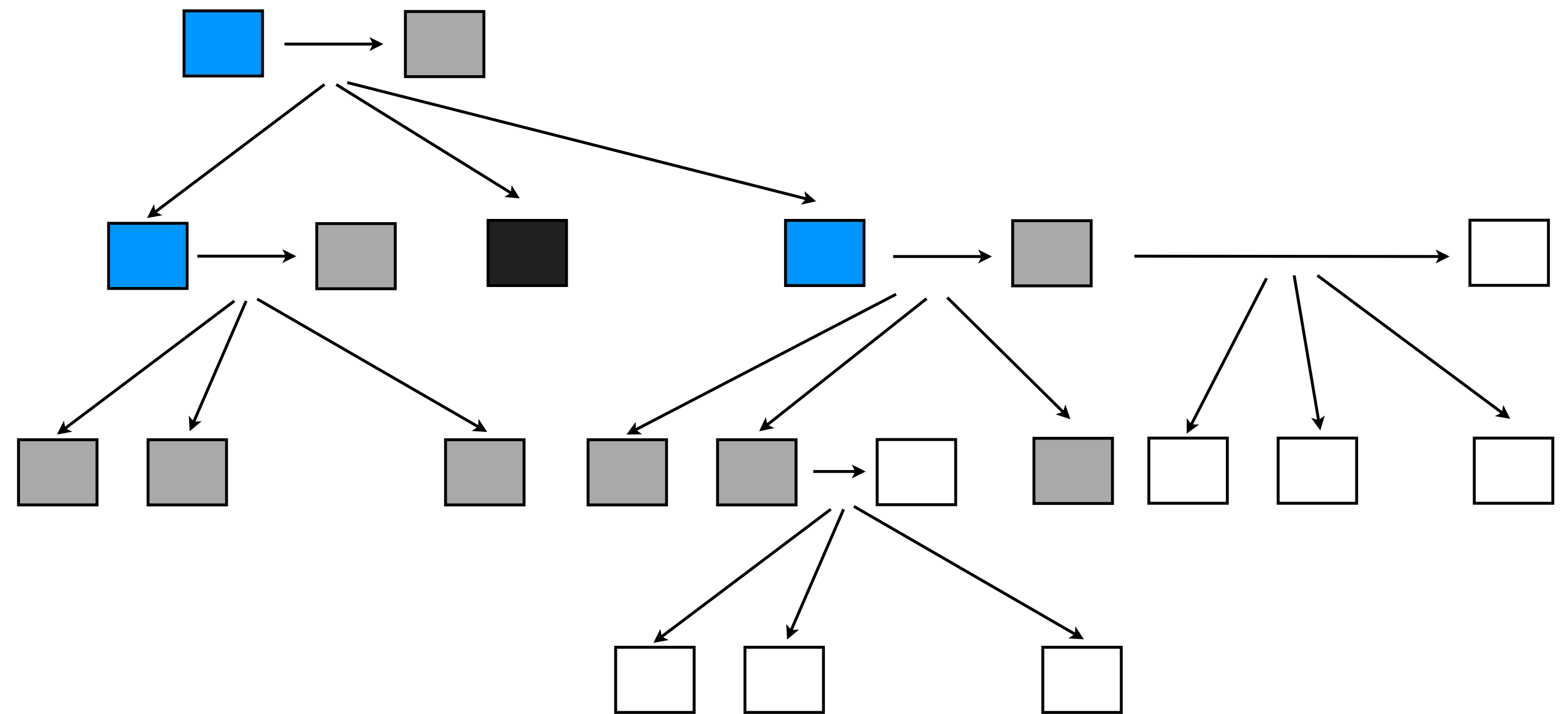
Job-tree

- The children
are run



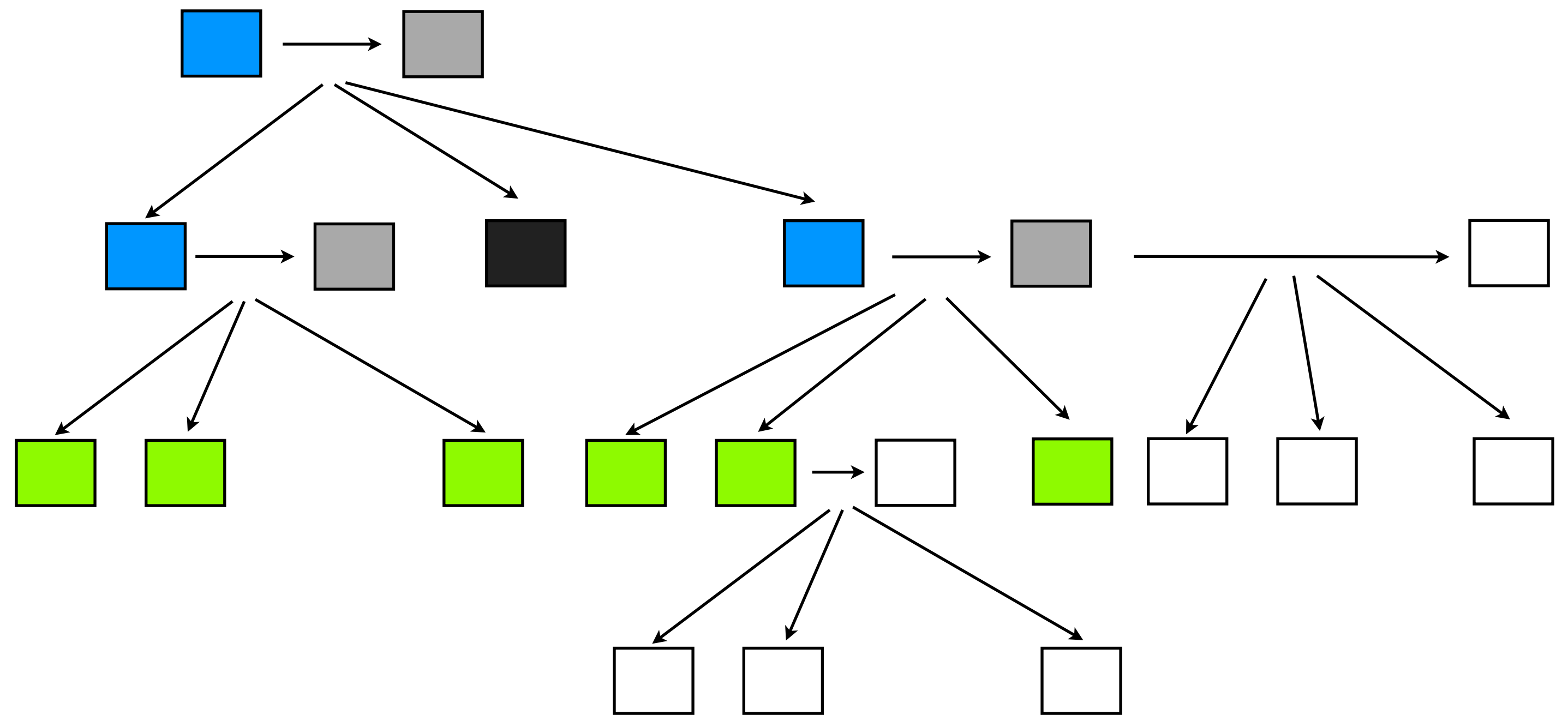
Job-tree

- Some create children and follow ons.
- Some do not, they are now complete (coloured black)



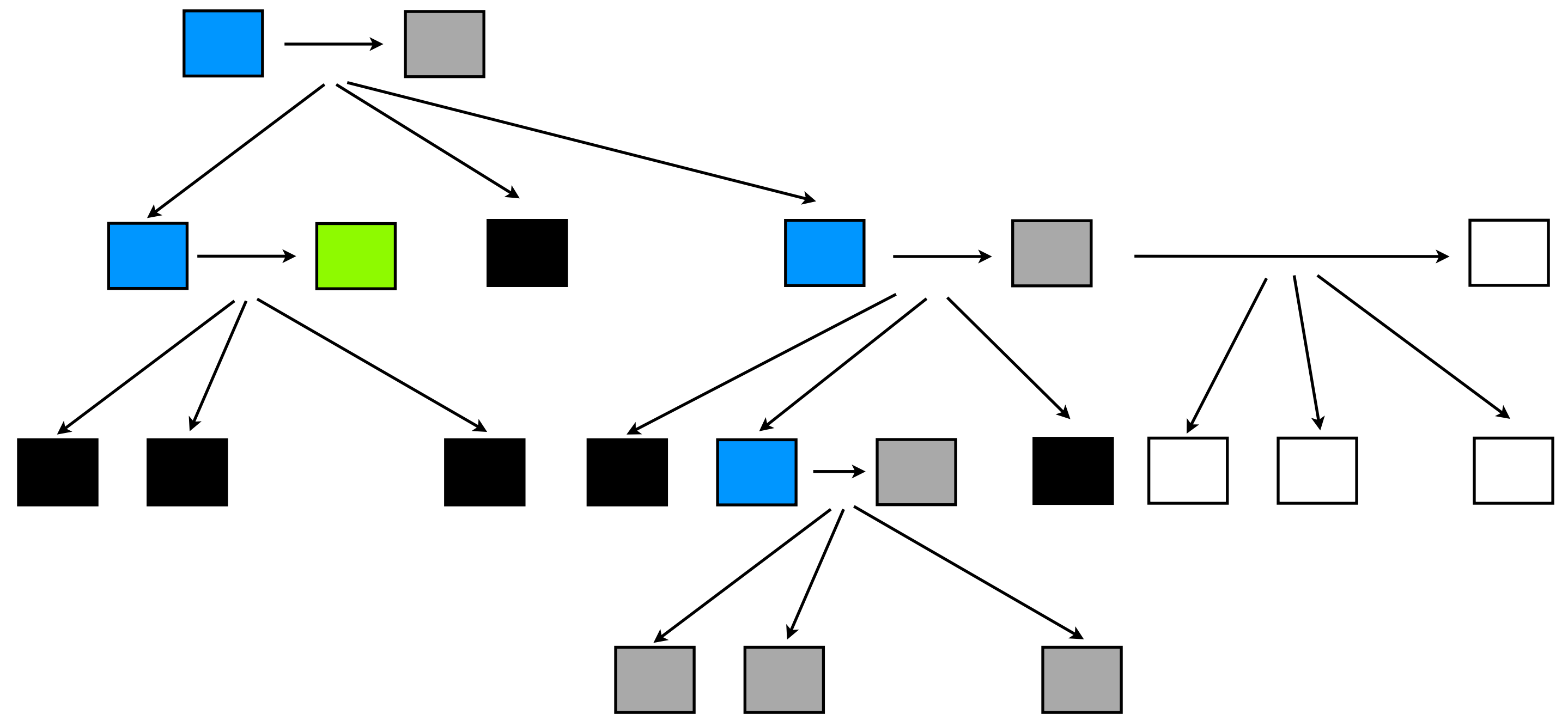
Job-tree

- More children are run



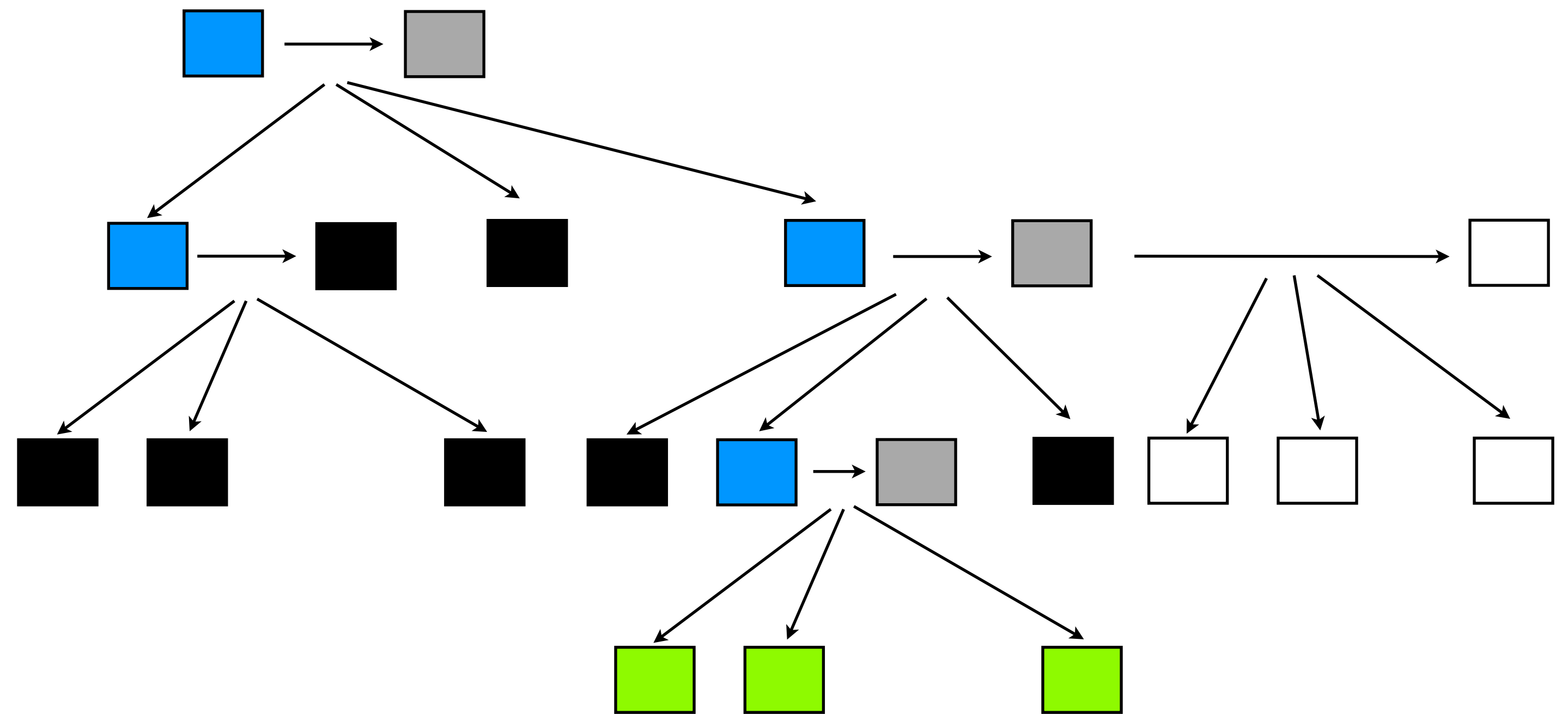
Job-tree

- Most are finished, one creates more children, and one follow can be run



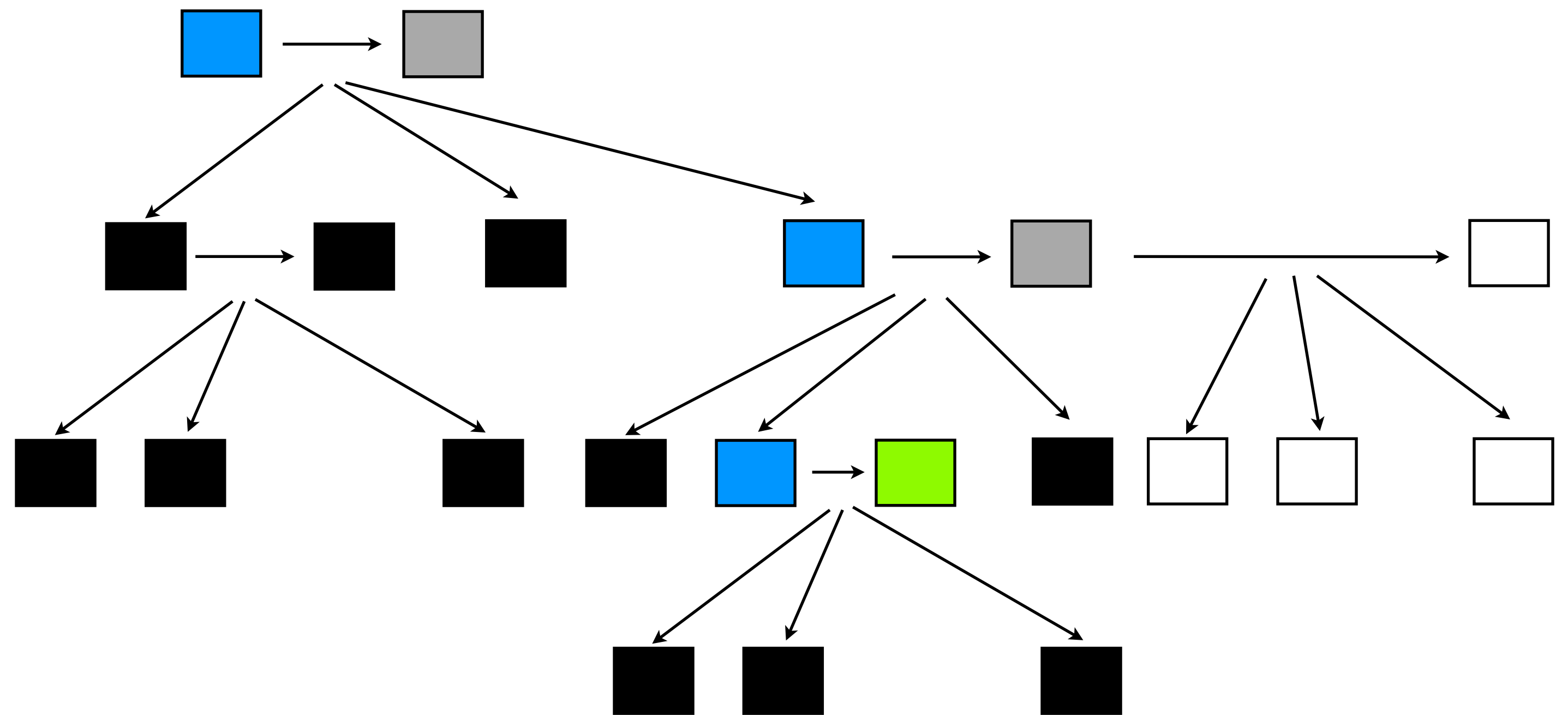
Job-tree

- etc, etc..



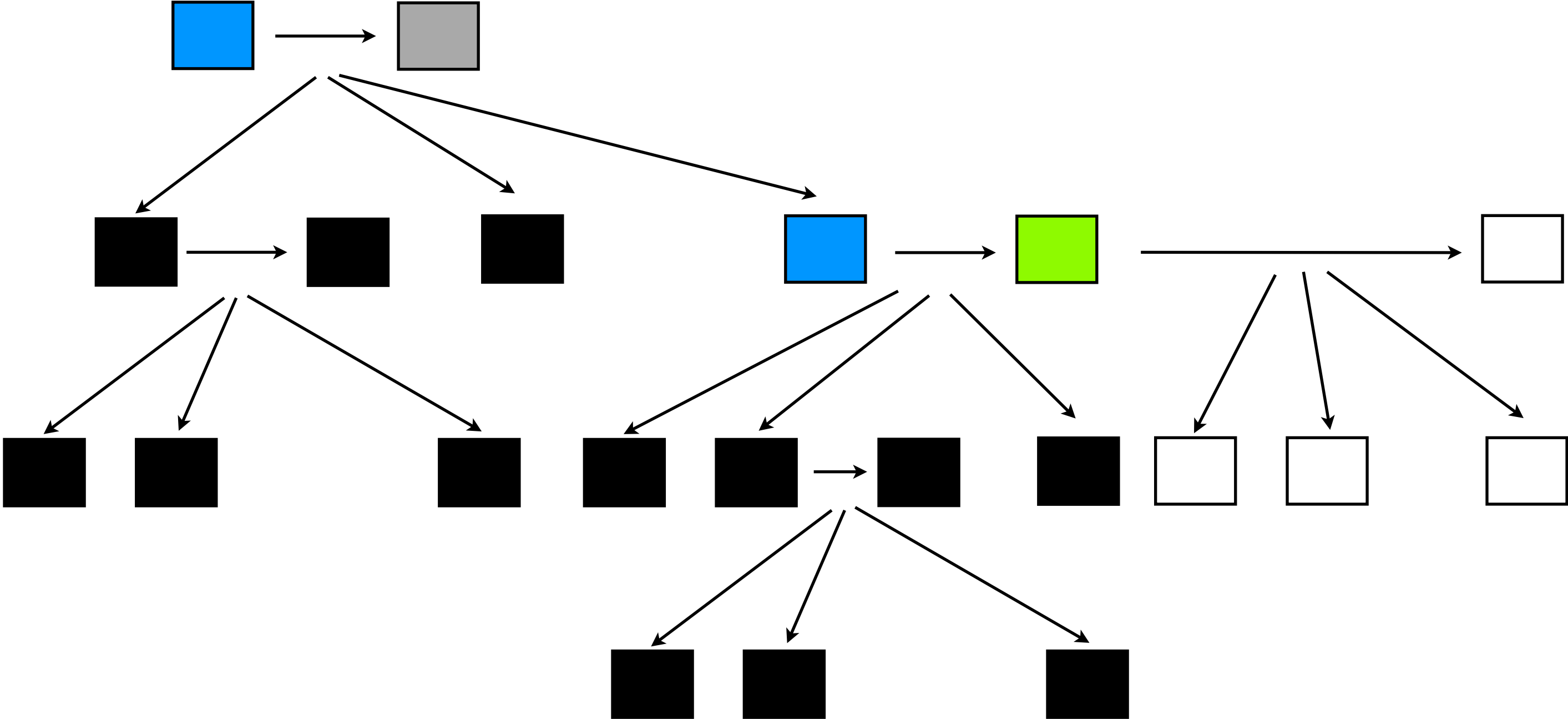
Job-tree

- etc, etc..

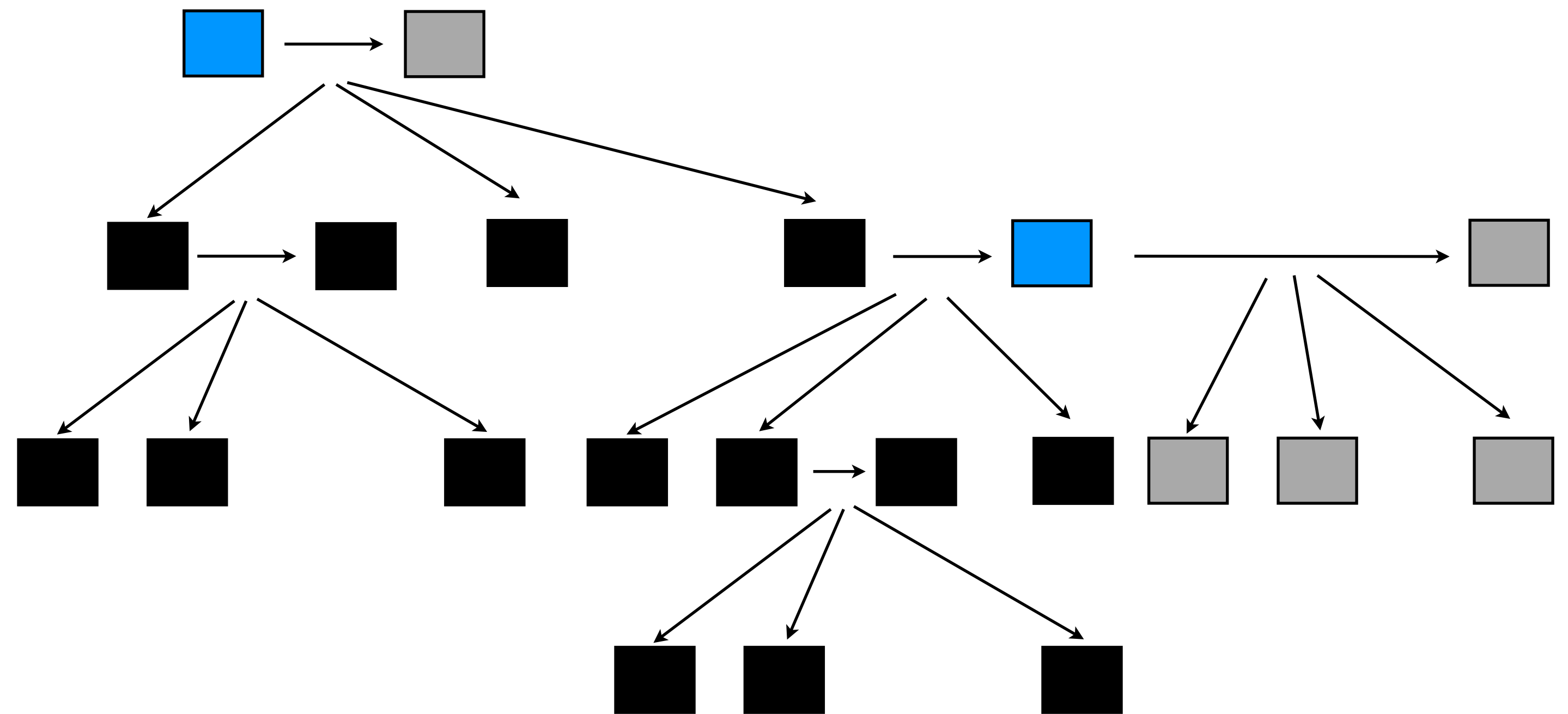


Job-tree

- etc, etc..



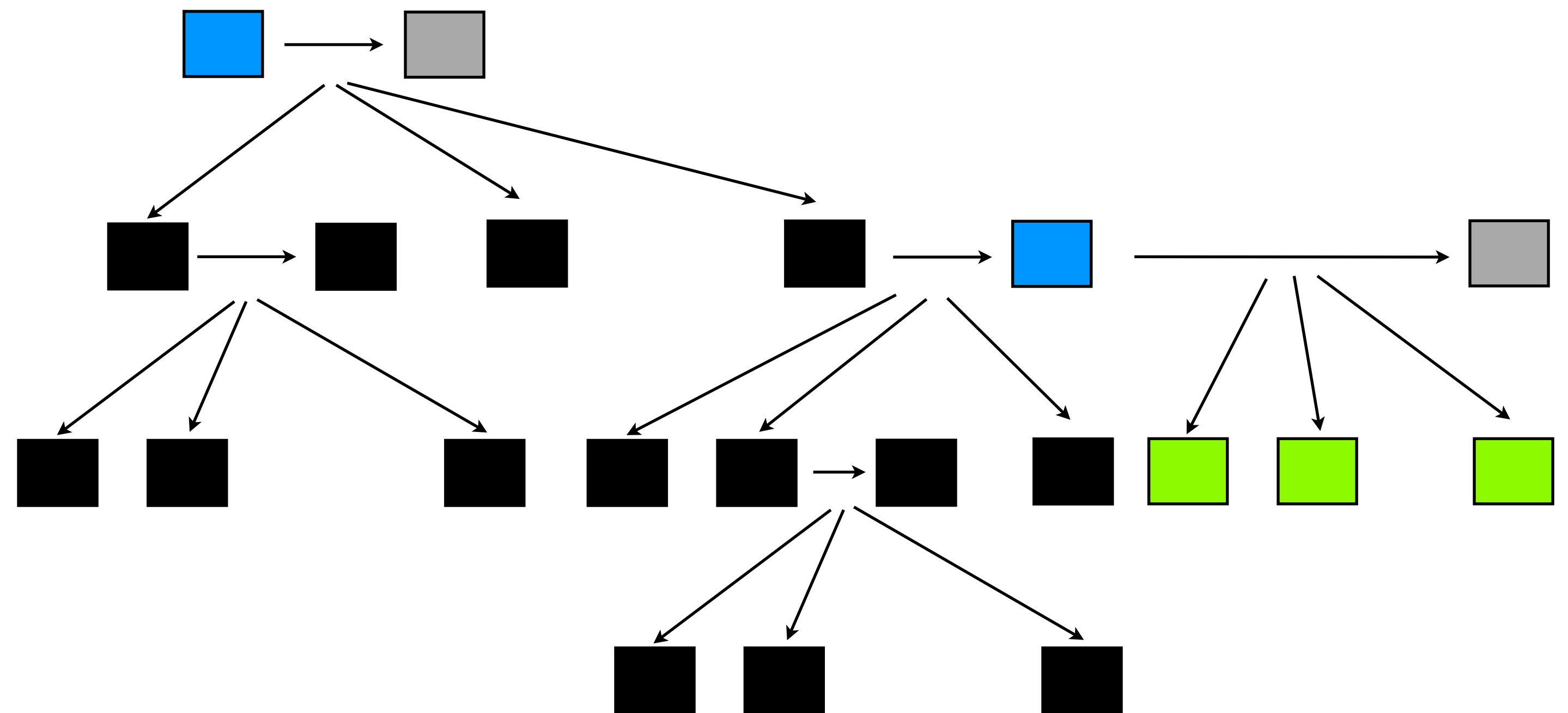
Job-tree



- etc, etc..

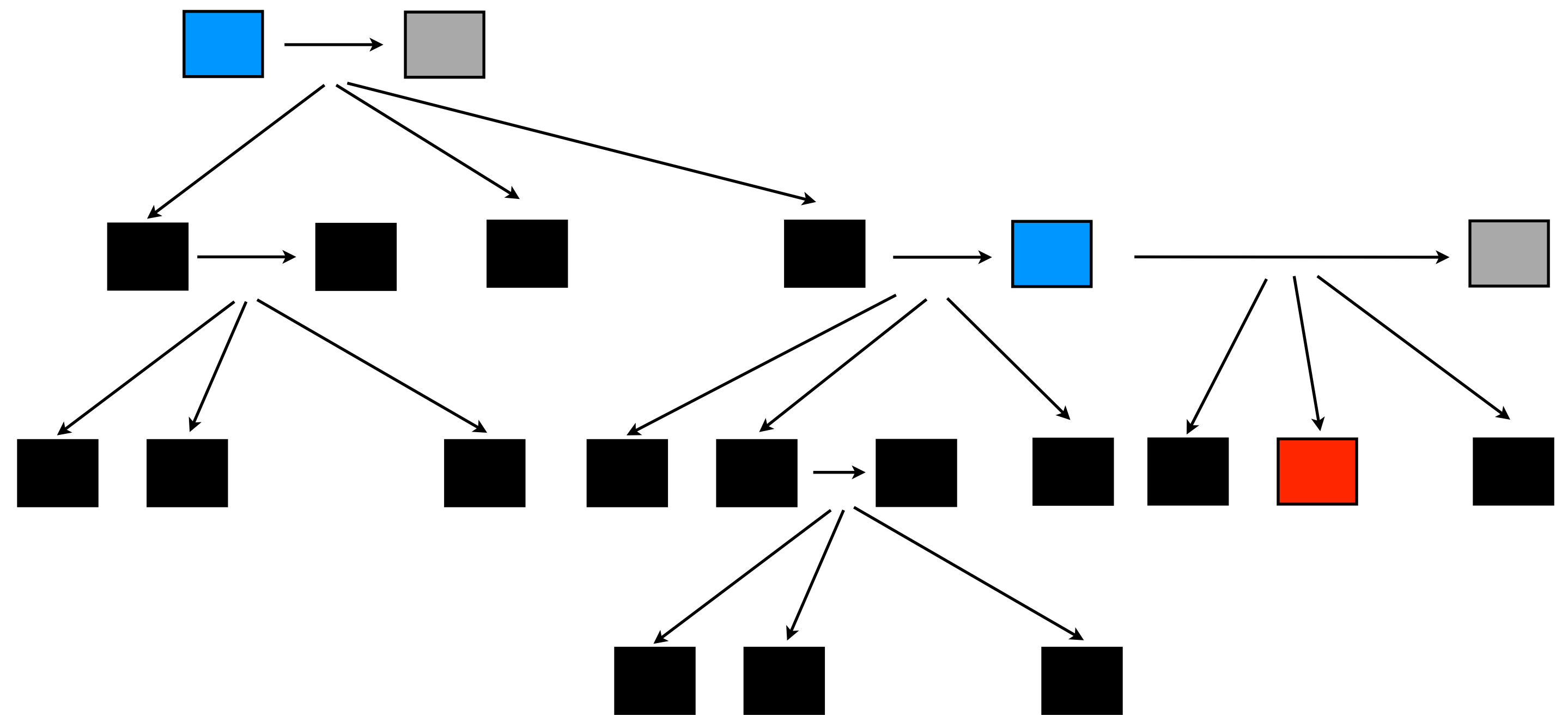
Job-tree

- etc, etc..



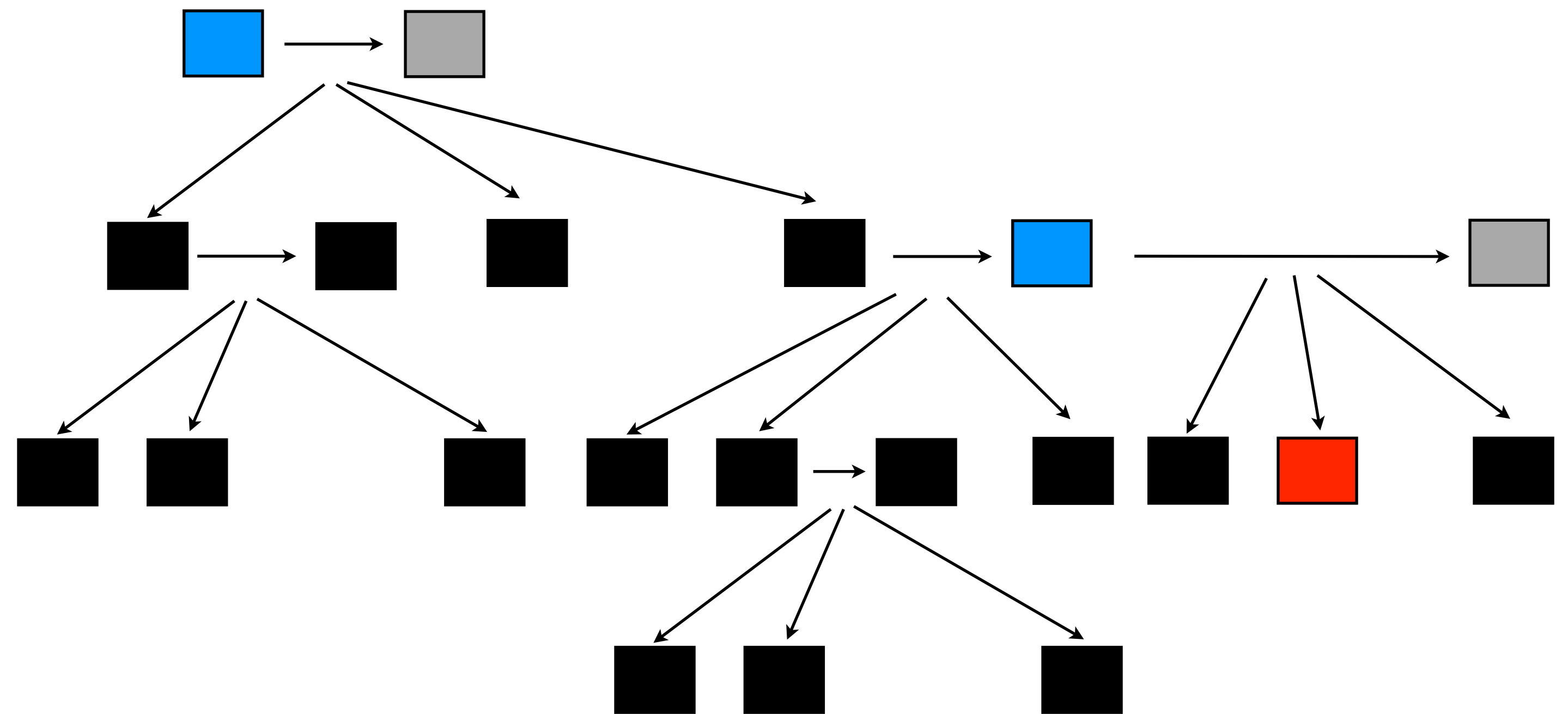
Job-tree

- Oh no! A job failed (coloured red)
- It will be restarted a preset number of times.



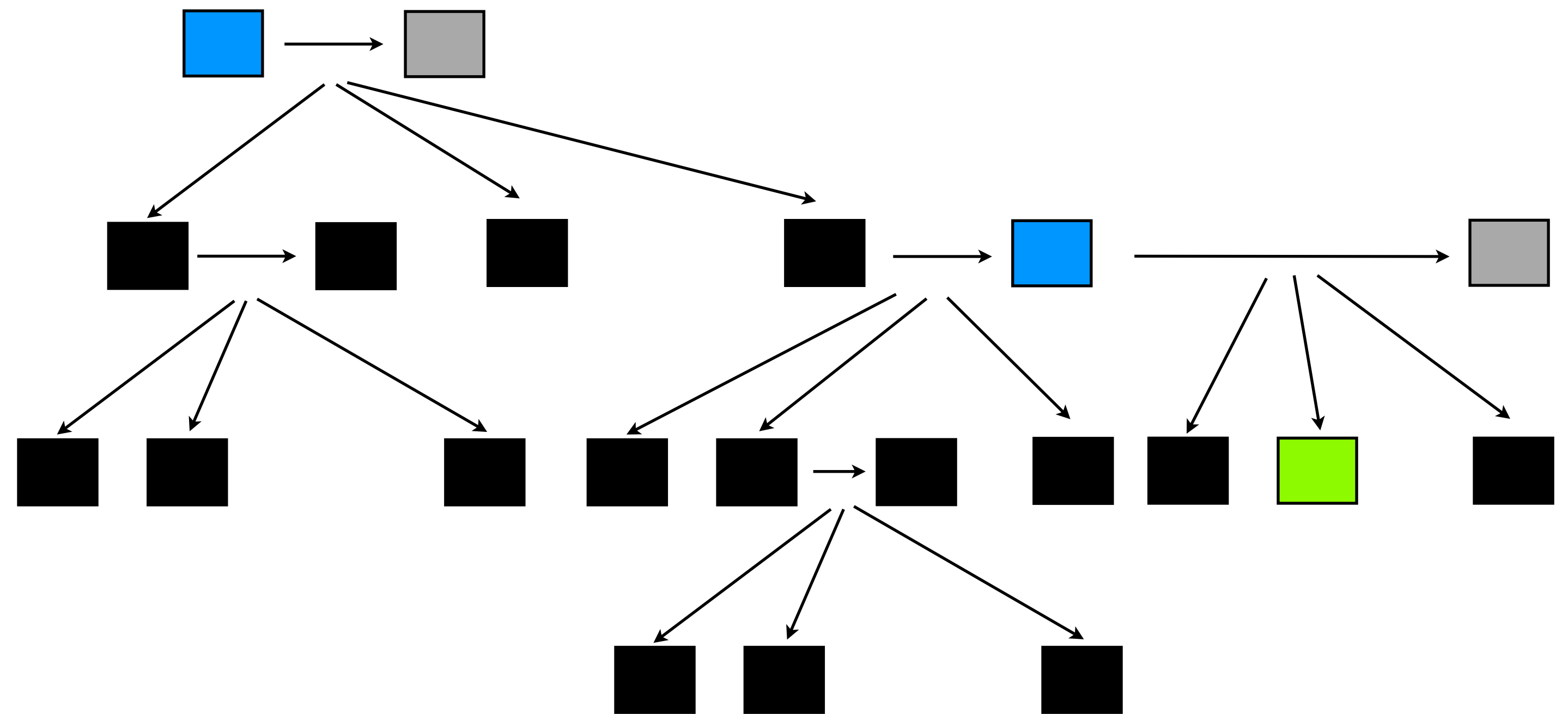
Job-tree

- If it still fails
Job-tree will
continue as
far as it can,
then return,
so you can fix
the job.
- Job-tree will
then re-start
from that
point.



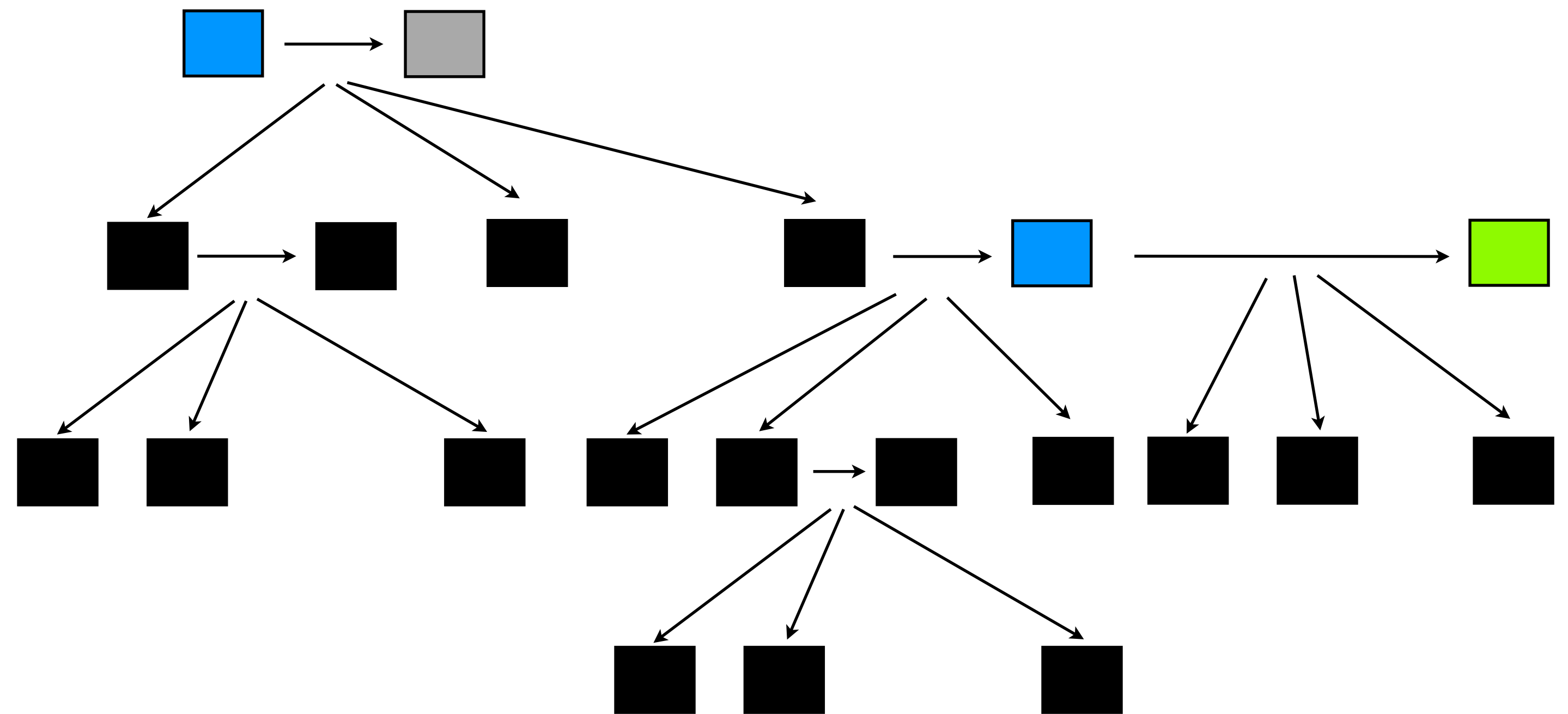
Job-tree

- So now we're running again..

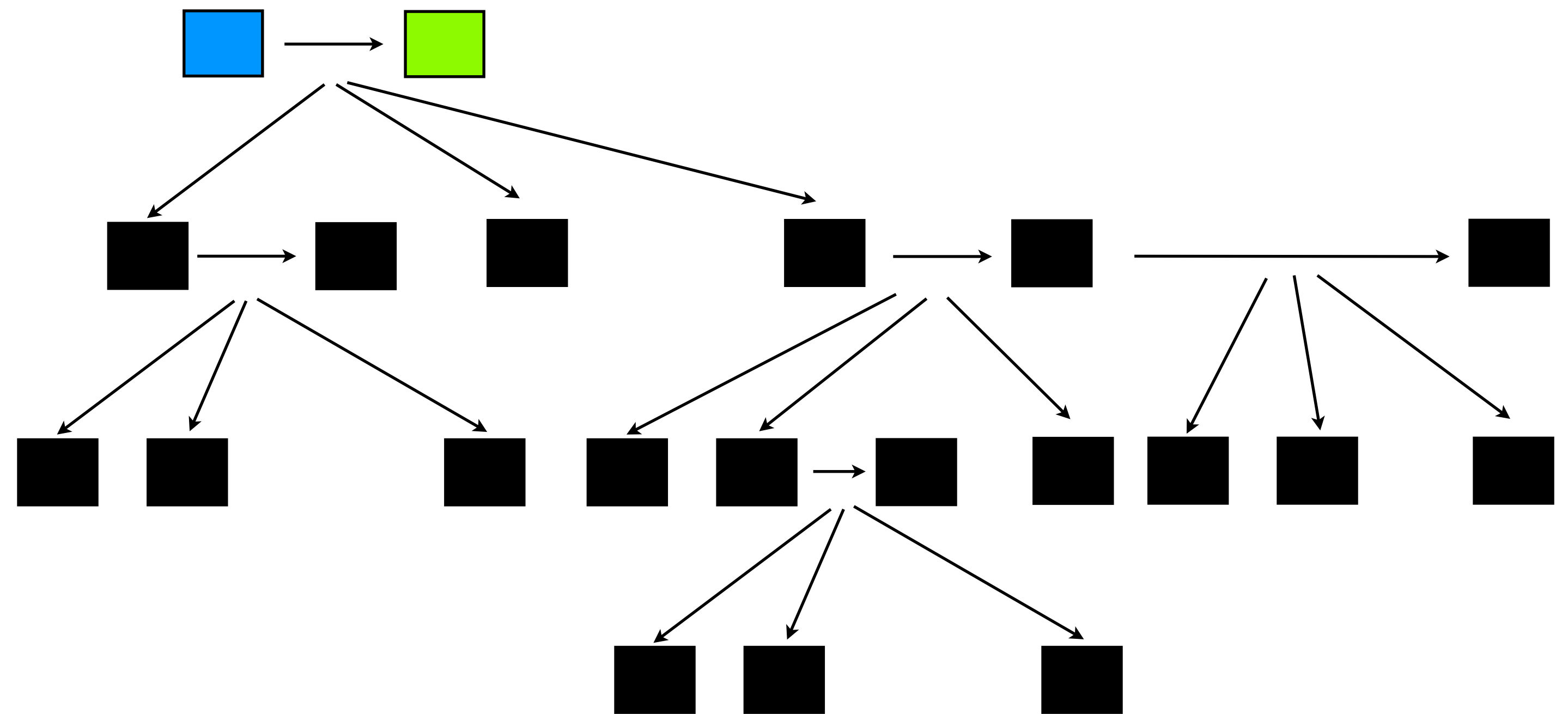


Job-tree

- etc, etc..



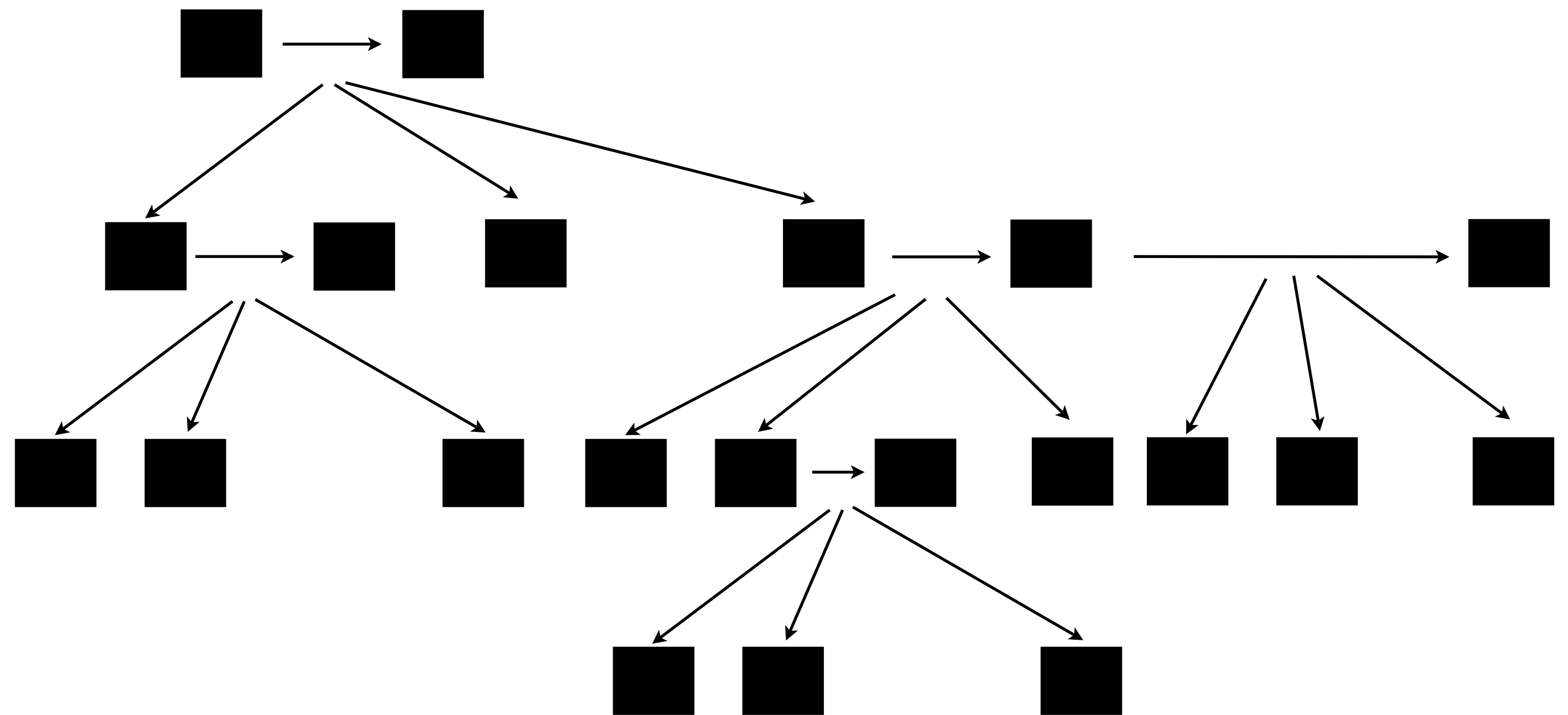
Job-tree



- etc, etc..

Job-tree

- And we're done.



Job-tree/Script-tree

- jobTree communicates by clunky XML files. Each job is passed a an xml file, which it edits and when the job is complete this file is processed.
- scriptTree removes this pain, you just inherit a 'Target' python class, as follows (we'll look at example for doing parallel merge sort).

Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target

class Setup(Target):
    """Sets up the sort.
    """
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N

    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

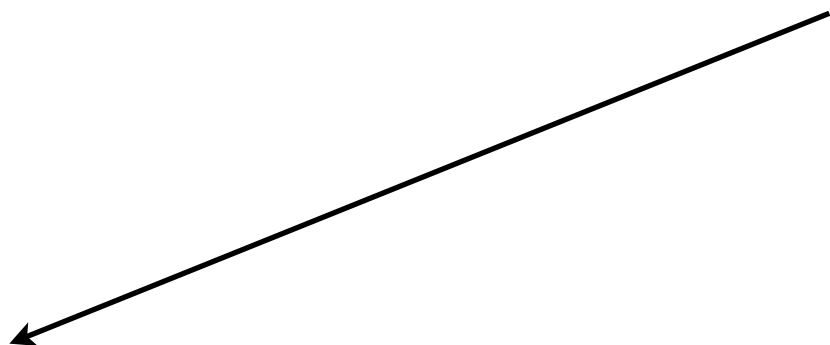
Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target

class Setup(Target):
    """Sets up the sort.
    """
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N

    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

Estimated runtime lets the
meta-scheduler be more
efficient



Job-tree/Script-tree

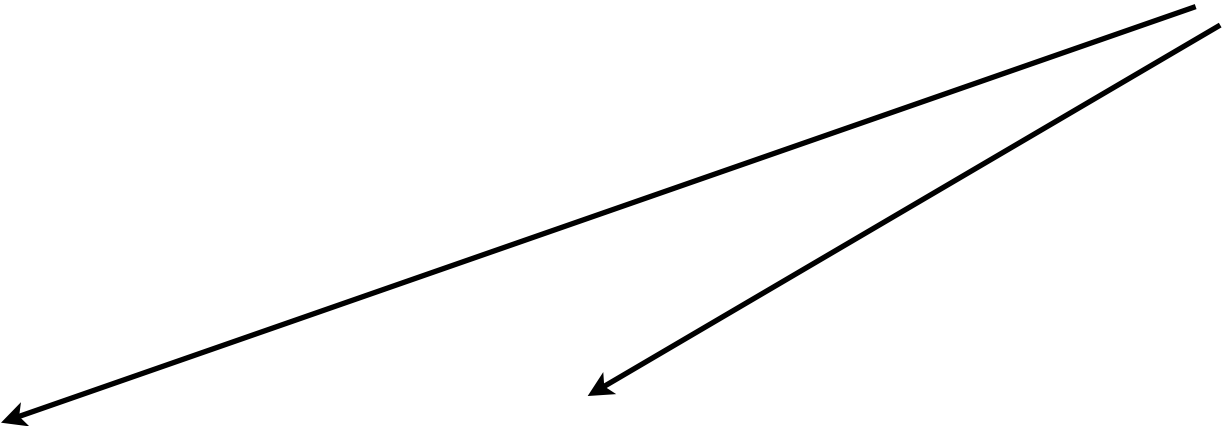
```
import os
from workflow.jobTree.scriptTree.target import Target
```

```
class Setup(Target):
    """Sets up the sort.
    """
```

```
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N
```

```
    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

Memory (bytes) and cpu
requirements can be
specified



Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target
```

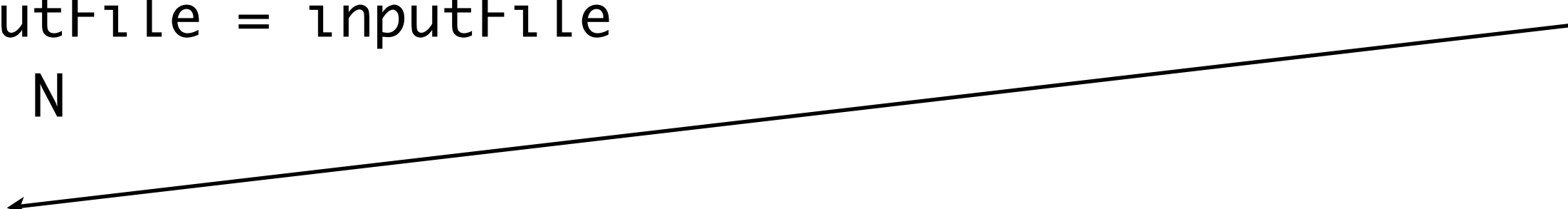
```
class Setup(Target):
```

```
    """Sets up the sort.
    """
```

```
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N
```

```
    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

Run, where you create
children, a follow on and
do work



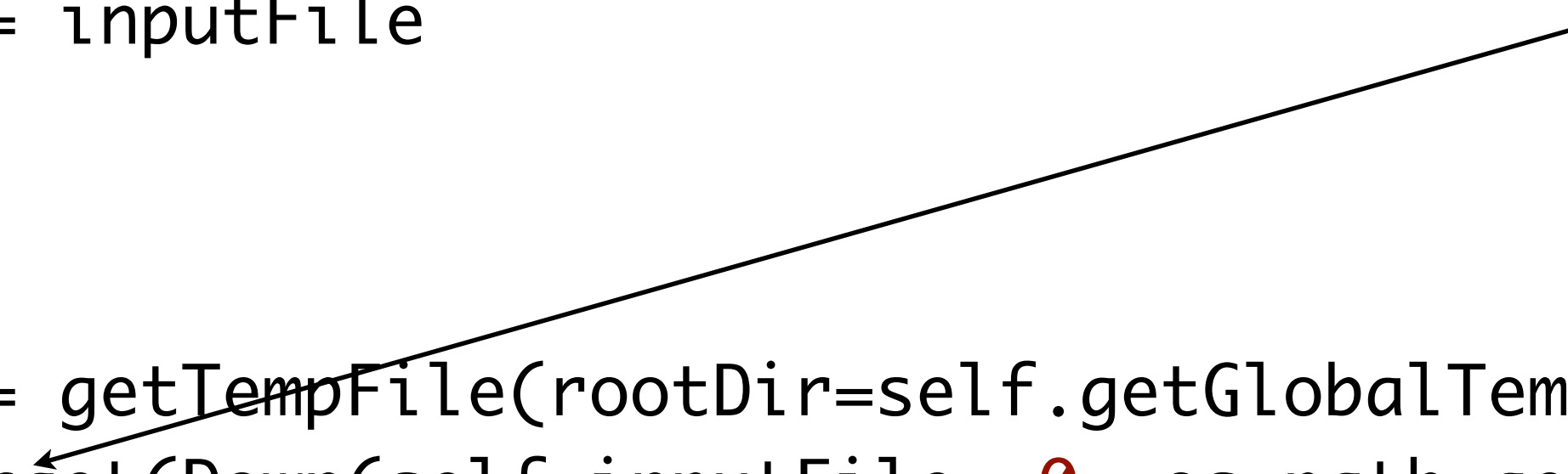
Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target

class Setup(Target):
    """Sets up the sort.
    """
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N

    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

Creating a child



Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target
```

```
class Setup(Target):
```

```
    """Sets up the sort.
    """
```

```
    def __init__(self, inputFile, N):
```

```
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N
```

```
    def run(self):
```

```
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

Creating the follow on



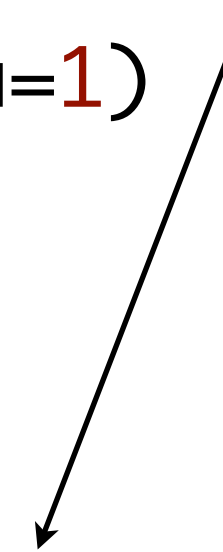
Job-tree/Script-tree

```
import os
from workflow.jobTree.scriptTree.target import Target

class Setup(Target):
    """Sets up the sort.
    """
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N

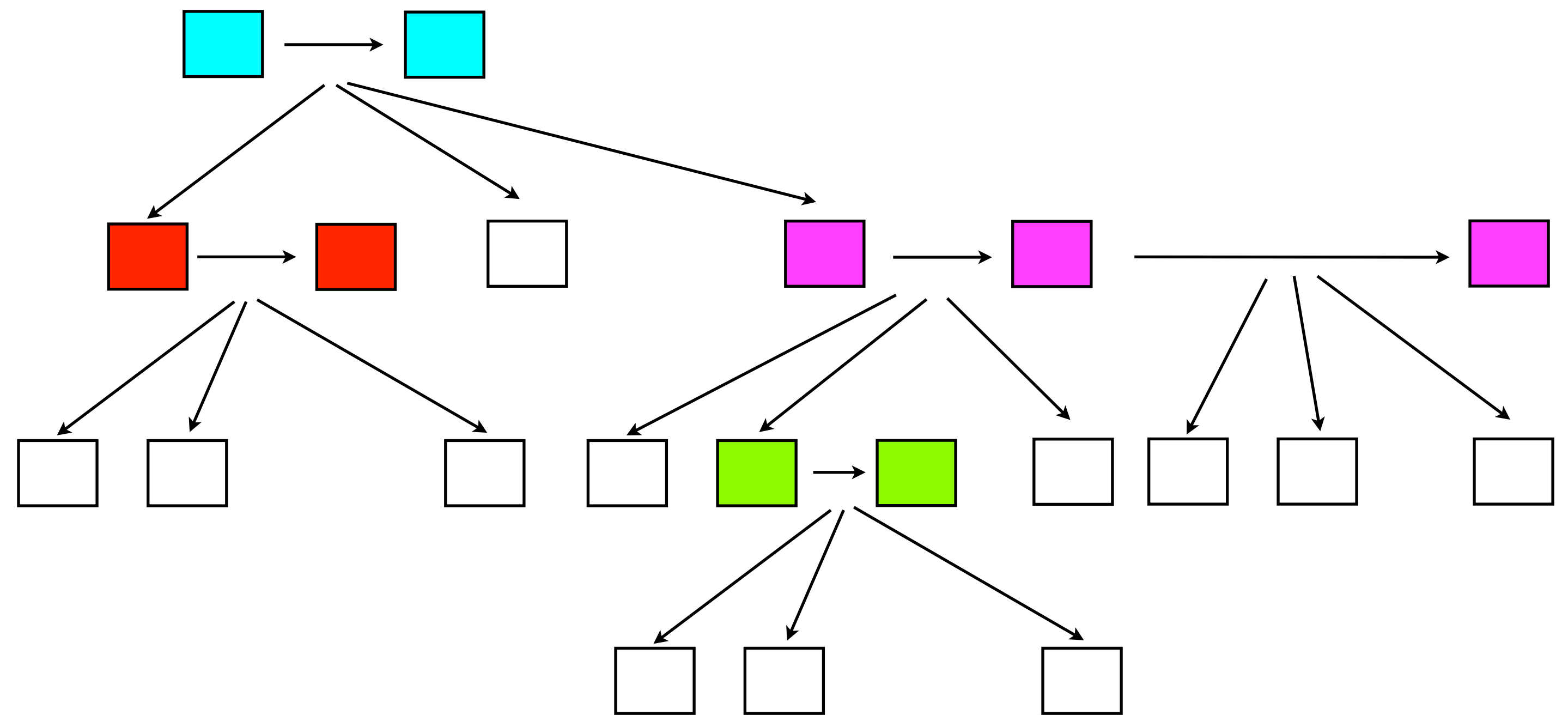
    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile),
self.N, tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

A global (visible to all machines on the cluster) temporary directory that exists for the life of the job, its setup jobs and its follow ons.



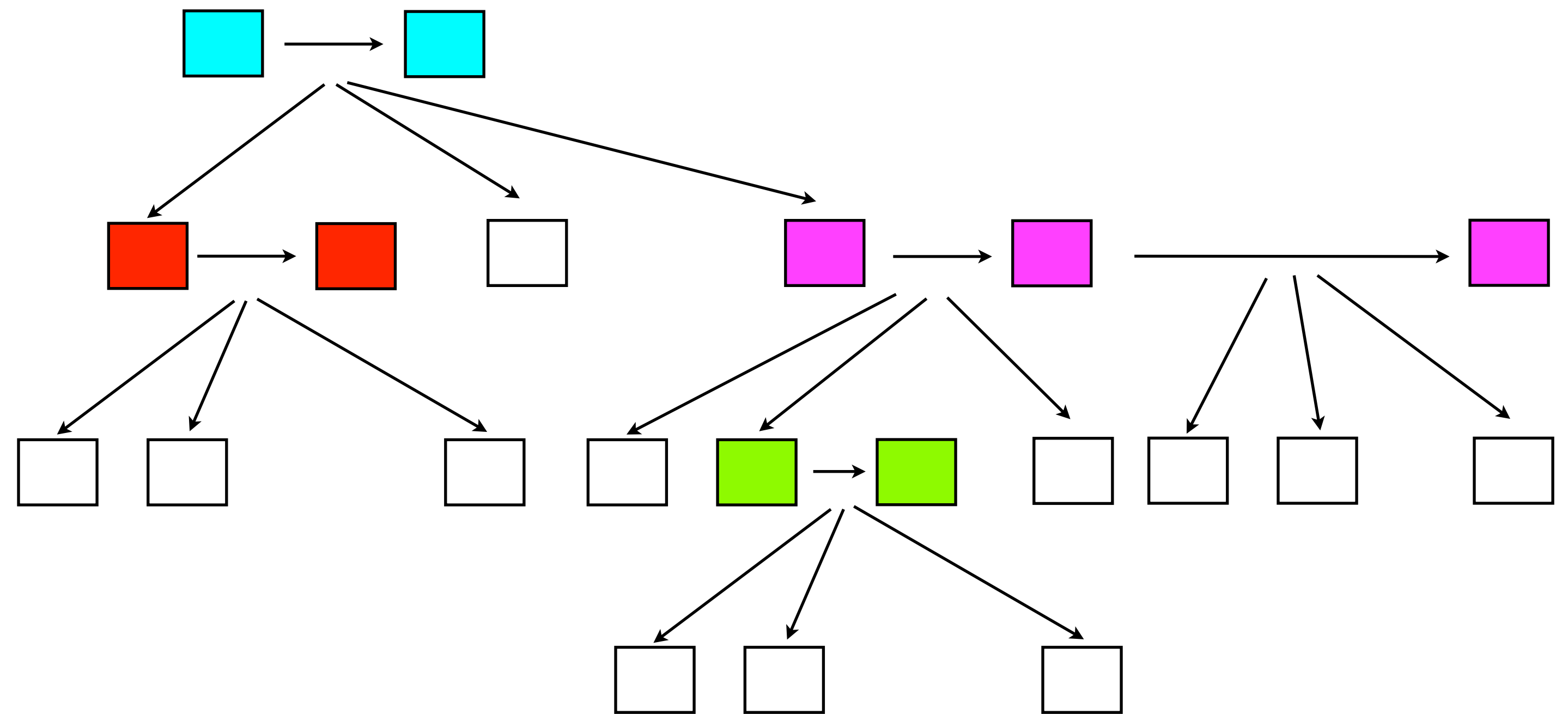
GlobalTempDir

- A chain consists of a founder, a sequence of successive follow ons and a closer.
- Non-trivial chains are coloured in example



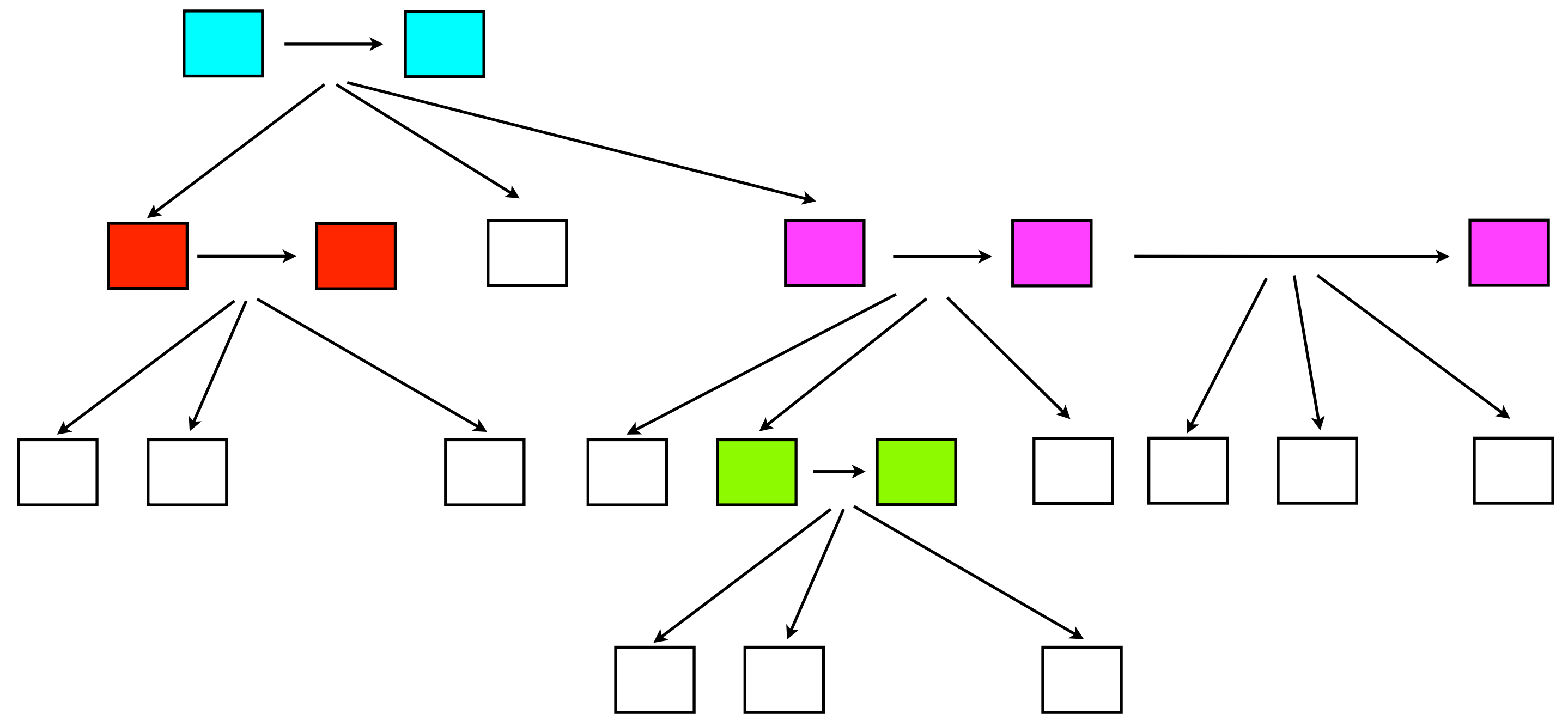
GlobalTempDir

- A 'globalTempDir' temporary directory is created for each chain.



GlobalTempDir

- It is created just before the founder and deleted just after the closer.
- It allows files to be passed around between follow ons and their children.



Job-tree/Script-tree

```
#!/usr/bin/env python
```

```
"""A demonstration of jobTree. Sorts the lines of a file into ascending order by doing a parallel merge sort.
"""
```

```
from optparse import OptionParser
import os
import shutil
from sonLib.bioio import getTempFile
from workflow.jobTree.scriptTree.target import Target
from workflow.jobTree.scriptTree.stack import Stack
from workflow.jobTree.test.sort.lib import merge, sort, copySubRangeOfFile, getMidPoint
```

```
class Setup(Target):
    """Sets up the sort.
    """
    def __init__(self, inputFile, N):
        Target.__init__(self, time=1, memory=1000000, cpu=1)
        self.inputFile = inputFile
        self.N = N

    def run(self):
        tempOutputFile = getTempFile(rootDir=self.getGlobalTempDir())
        self.addChildTarget(Down(self.inputFile, 0, os.path.getsize(self.inputFile), self.N,
tempOutputFile))
        self.setFollowOnTarget(Cleanup(tempOutputFile, self.inputFile))
```

```
class Cleanup(Target):
    """Copies back the temporary file to input once we've successfully sorted the temporary file.
    """
    def __init__(self, tempOutputFile, outputFile):
        Target.__init__(self)
        self.tempOutputFile = tempOutputFile
        self.outputFile = outputFile

    def run(self):
        shutil.copyfile(self.tempOutputFile, self.outputFile)
```

```
class Down(Target):
    """Input is a file and a range into that file to sort and an output location in which
    to write the sorted file.
    If the range is larger than a threshold N the range is divided recursively and
    a follow on job is then created which merges back the results else
    the file is sorted and placed in the output.
    """
    def __init__(self, inputFile, fileStart, fileEnd, N, outputFile):
        assert fileStart >= 0
        assert fileStart <= fileEnd
        Target.__init__(self, time=0.05)
        self.inputFile = inputFile
        self.fileStart = fileStart
        self.fileEnd = fileEnd
        self.N = N
        self.outputFile = outputFile

    def run(self):
        length = self.fileEnd - self.fileStart
        assert length >= 0
        if length > self.N:
            midPoint = getMidPoint(self.inputFile, self.fileStart, self.fileEnd)
            assert midPoint >= self.fileStart
            assert midPoint+1 < self.fileEnd
            #We will subdivide the file
            tempFile1 = getTempFile(rootDir=self.getGlobalTempDir())
            tempFile2 = getTempFile(rootDir=self.getGlobalTempDir())
            self.addChildTarget(Down(self.inputFile, self.fileStart, midPoint+1, self.N, tempFile1))
            self.addChildTarget(Down(self.inputFile, midPoint+1, self.fileEnd, self.N, tempFile2)) #Add
one to avoid the newline
            self.setFollowOnTarget(Up(tempFile1, tempFile2, self.outputFile))
        else:
            #We can sort this bit of the file
            copySubRangeOfFile(self.inputFile, self.fileStart, self.fileEnd, self.outputFile)
            sort(self.outputFile)
```

```
class Up(Target):
    """Merges the two files and places them in the output.
    """
    def __init__(self, inputFile1, inputFile2, outputFile):
        Target.__init__(self, time=0.05)
        self.inputFile1 = inputFile1
        self.inputFile2 = inputFile2
        self.outputFile = outputFile
```

```
    def run(self):
        merge(self.inputFile1, self.inputFile2, self.outputFile)
```


Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

benedict\$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt

Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

Create a parser for the script and
add the Job-tree options to it.



```
benedict$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt
```

Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

Parse the options and args



```
benedict$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt
```

Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

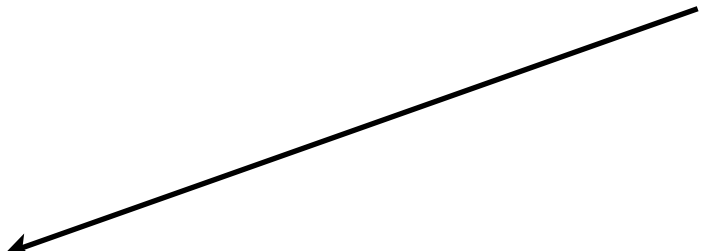
    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

Run Job-tree (alternatively
you can pass in the options to jobtre
manually and use your own options/args
parser)



benedict\$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt

Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

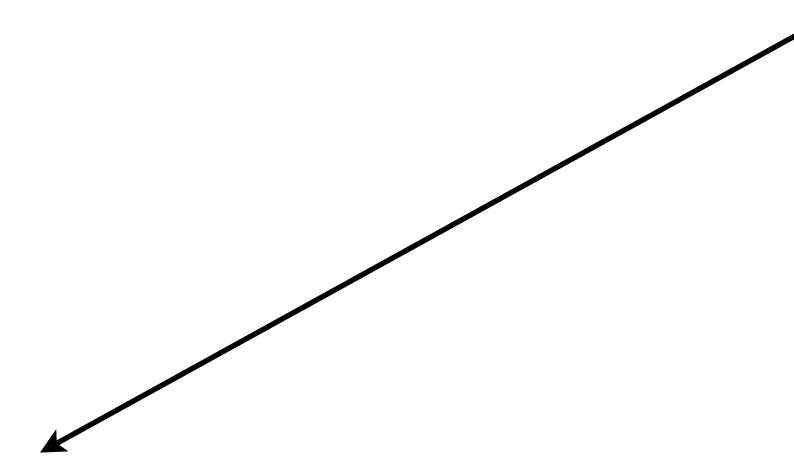
    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

The command line



```
benedict$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt
```

Running Job-tree/Script-tree

```
def main():
    parser = OptionParser()
    Stack.addJobTreeOptions(parser)

    parser.add_option("--fileToSort", dest="fileToSort",
                      help="The file you wish to sort")

    options, args = parser.parse_args()

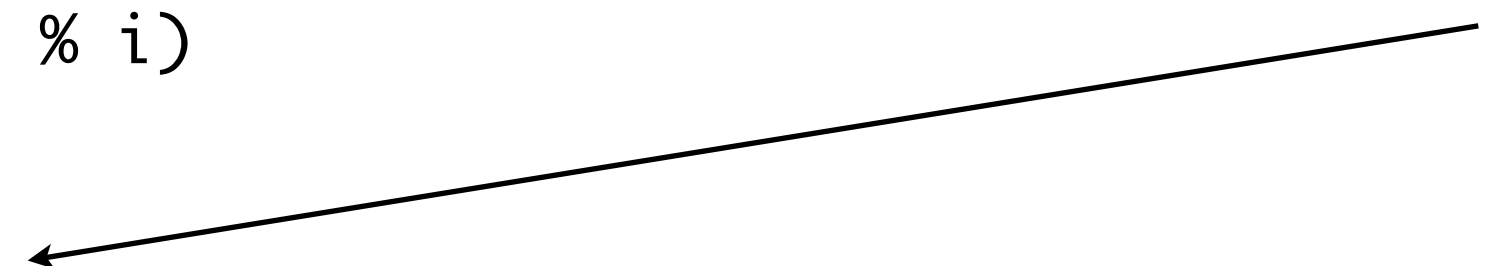
    if options.fileToSort == None:
        raise RuntimeError("No file to sort given")

    #Now we are ready to run
    i = Stack(Setup(options.fileToSort, int(options.N))).startJobTree(options)

    if i:
        raise RuntimeError("The jobtree contained %i failed jobs" % i)

if __name__ == '__main__':
    from workflow.jobTree.test.sort.Script-treeTest_Sort import *
    main()
```

Oh, and a little voodoo
required by
the pickler



benedict\$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt

Restarting a job-tree

- A job-tree is run initially by executing the script.
- If it fails call:
 - `jobTree --jobTree FOO`
 - Where foo is the location of the jobTree
- The job-tree will restart from where it failed.

Cleaning up

- Once a job-tree finishes up you can check it completed okay by running:
- `jobTreeStatus --jobTree FOO`
- Where FOO is the jobTree
- Adding the `--verbose` flag will print out any error log files associated with failed jobs
- To finish up, simply delete the job-tree.

Job-tree/Script-tree Misc.

- Your environment variables are inherited from the executing shell, so you can use relative path names and program names without stressing.
- Can run on a parasol cluster, a grid engine cluster, or in single machine mode (using multiple threads), so you can test on a workstation before you push your pipeline to the cluster.
- Is (theoretically) easily extended to work on another batch system - just inherit the abstract batch system class.

Job-tree Summary

- Allows you to dynamically create arbitrarily parallelised batches of jobs.
- Provides other nice features.
- Is stable and used by me, Dent, Krish, Charlie, Daniel, Ngan and run by others, including Wendy, Ted, Bernard, etc.
- Unfortunately, the code is pretty dense and prototype-y, and there are some clunky edges.

Utilities

- These next slides detail the utilities for job-tree

jobTreeStats

- Run as `jobTreeStats --jobTree foo`
- Reports status of tree, importantly telling you how many jobs are failed etc.
- Verbose flag will report log files of failed jobs
- Output could be improved!

jobTreeKill

- To kill a jobtree, first kill the master process, then run:
- `jobTreeKill --jobTree foo`
- Kills all queued and running jobs associated with the job-tree on the batch system (only works with parasol currently, not grid engine).
- Does not remove the batch (can be restarted again)

jobTreeStats: Balancing job-trees

- Your jobs on the cluster should run for some ‘ideal’ time in order to efficient.
- Job-tree will attempt to agglomerate your short running jobs to avoid paying scheduling costs (which may be a few seconds of latency per job!)
- Running job-tree with --stats option allows you to run jobTreeStats

```
benedict$ Script-treeTest_Sort.py --jobTree foo/Job-tree --logDebug --fileToSort bar.txt --stats  
benedict$ jobTreeStats --jobTree foo/Job-tree
```


jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252699852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

Example from
script-
treeTest_Sort.py

jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252639852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

Total runtime and
stats about job
tree.

jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252699852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

Times on targets,
which are
agglomerated into
jobs

jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252699852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

Times on individual targets, showing how close your estimated jobs came to the actual run-times

jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252699852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

This example ran
8193 targets, in a
single thread in
8.5 seconds!

jobTreeStats: Balancing job-trees

```
<collated_stats batch_system="single_machine" default_cpu="1" default_memory="2147483648" job_time="0.5" max_jobs="922337203
6854775807" max_threads="1" total_run_time="8.50252699852">
  <slave average_time="0.55012343824" max_time="1.75700092316" median_time="0.429350137711" min_time="0.0157799720764" total
_number="16" total_time="8.80197501183"/>
  <job average_number_per_slave="2.5625" average_time="0.19536111413" max_number_per_slave="10" max_time="0.69593000412" med
ian_number_per_slave="1" median_time="0.134316921234" min_number_per_slave="1" min_time="0.0079870223999" total_number="41"
total_time="8.00980567932"/>
  <target average_number_per_job="199.829268293" average_time="0.000488620646544" max_number_per_job="1112" max_time="0.0799
999237061" median_number_per_job="32" median_time="0.000387191772461" min_number_per_job="0" min_time="0.000193119049072" to
tal_number="8193" total_time="4.00326895714"/>
  <target_types>
    <Down average_time="0.00047725673919" estimated_time="0.00045" max_time="0.0799999237061" median_time="0.000394821166992
" min_time="0.000314950942993" total_number="6142" total_time="2.93131089211"/>
    <Setup average_time="0.00031590461731" estimated_time="0.00025" max_time="0.00031590461731" median_time="0.0003159046173
1" min_time="0.00031590461731" total_number="1" total_time="0.00031590461731"/>
    <Cleanup average_time="0.00308609008789" estimated_time="0.0031" max_time="0.00308609008789" median_time="0.003086090087
89" min_time="0.00308609008789" total_number="1" total_time="0.00308609008789"/>
    <ParallelFollowOnTarget average_time="0.000471115112305" estimated_time="0.0" max_time="0.000471115112305" median_time="
0.000471115112305" min_time="0.000471115112305" total_number="2" total_time="0.00094223022461"/>
    <Up average_time="0.000521550483685" estimated_time="0.0007" max_time="0.0595879554749" median_time="0.000261068344116"
min_time="0.000193119049072" total_number="2047" total_time="1.0676138401"/>
  </target_types>
</collated_stats>
```

Times that the slaves ran 'jobs', in this case we've asked for a job-runtime of 0.5 seconds, the actual average was 0.55 seconds

Grid-engine cluster

- PK cluster + memk + (shortly) kolossus
- Heterogenous cluster, with machines:
 - from 2 to 64 cpus.
 - from 4 to 1024 gigs of memory
 - ~400 nodes
- Within the cancer firewall, still working out the kinks.