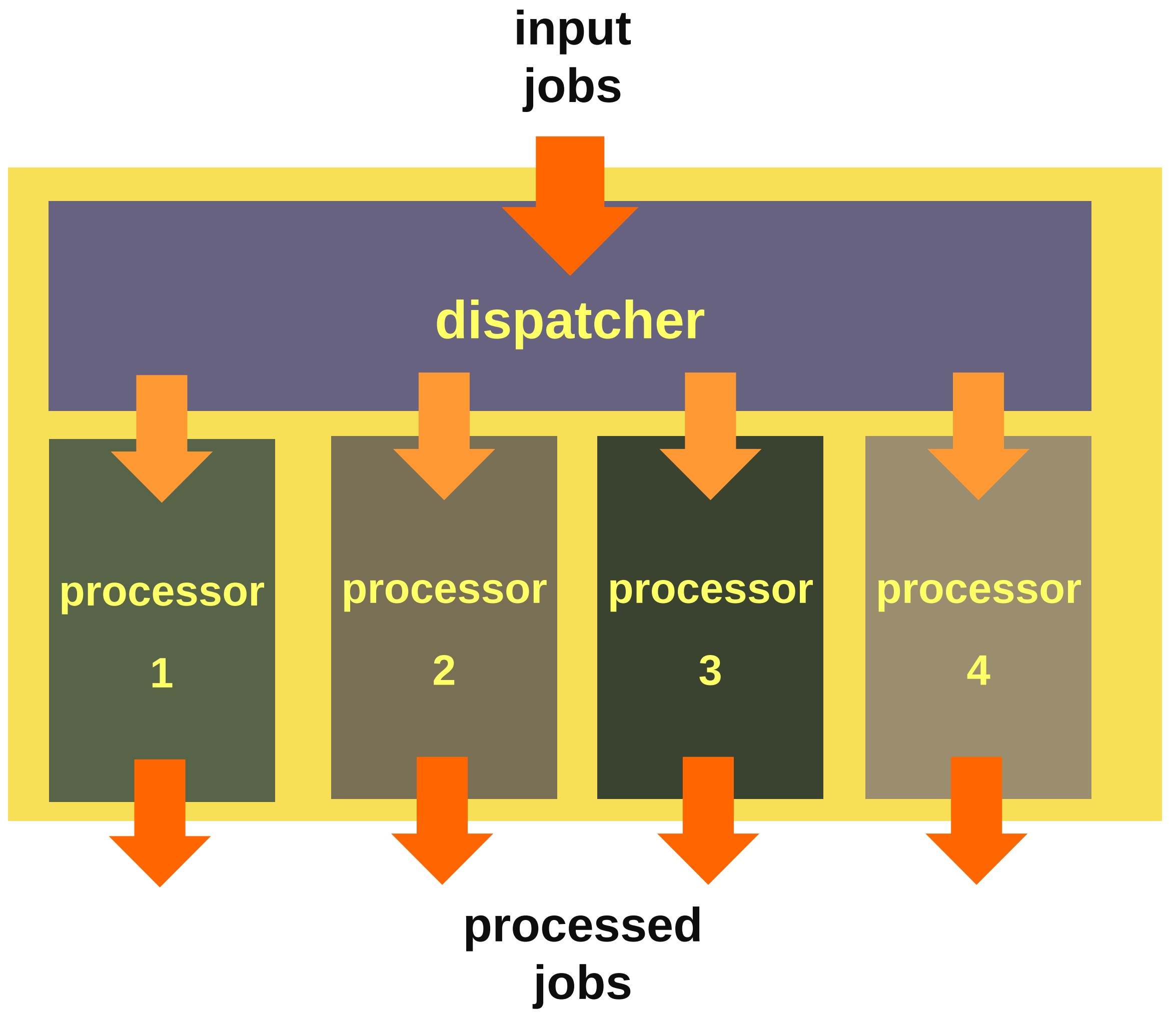
Model-Based Testing

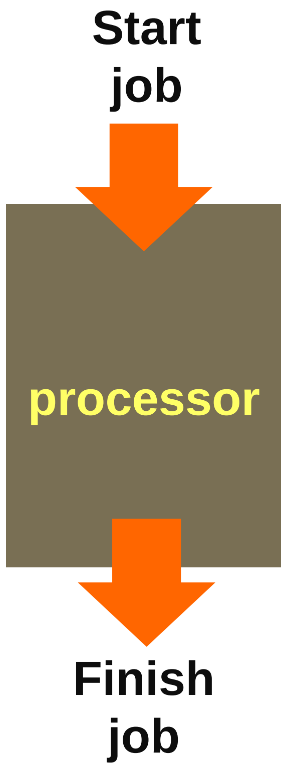
Example :

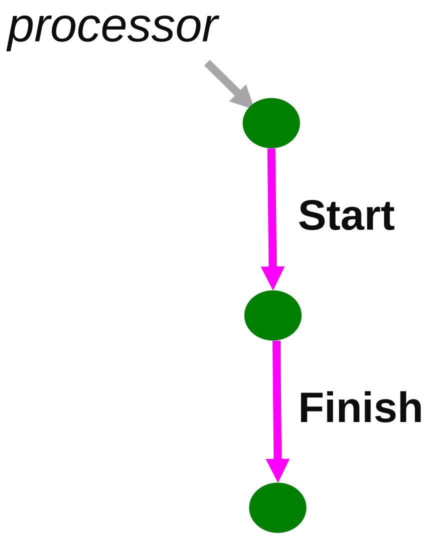
Dispatcher-Processing System

Example: Dispatcher-Processing System

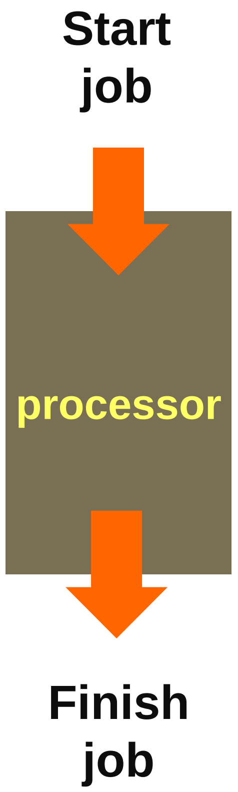


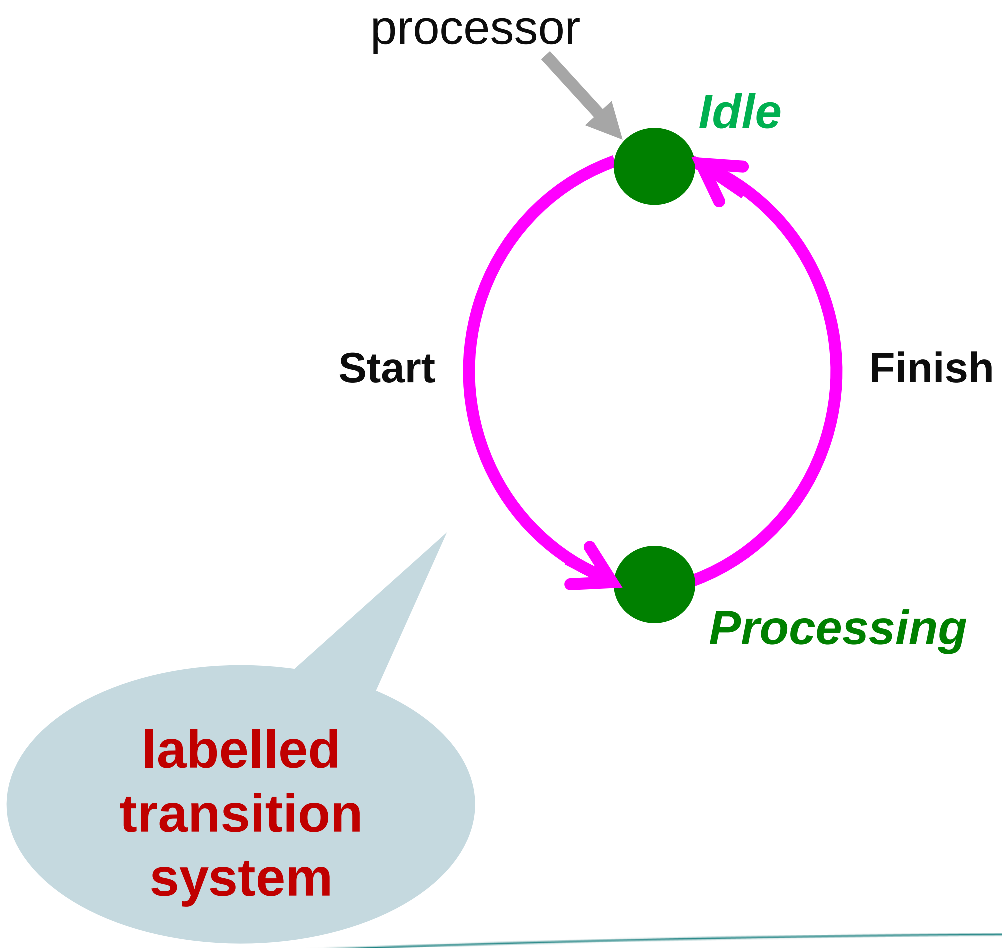
Example: Dispatcher-Processing System



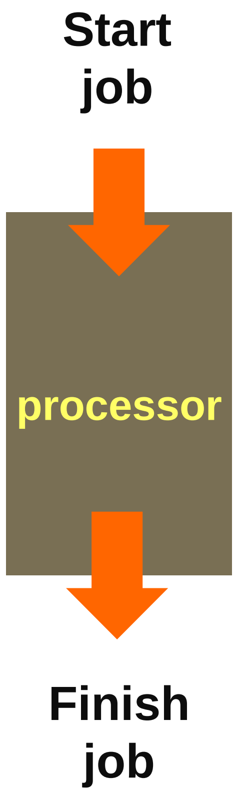


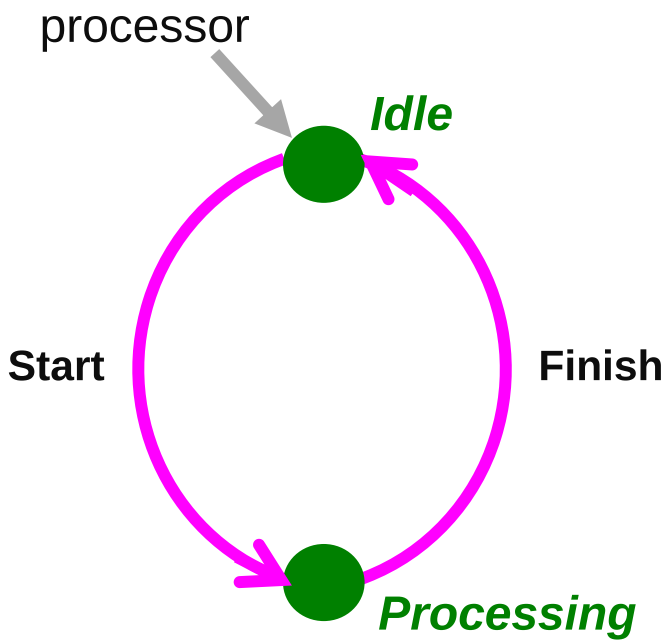
Example: Dispatcher-Processing System





Example: Dispatcher-Processing System

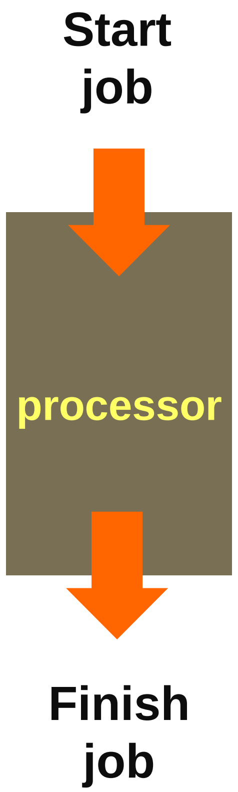


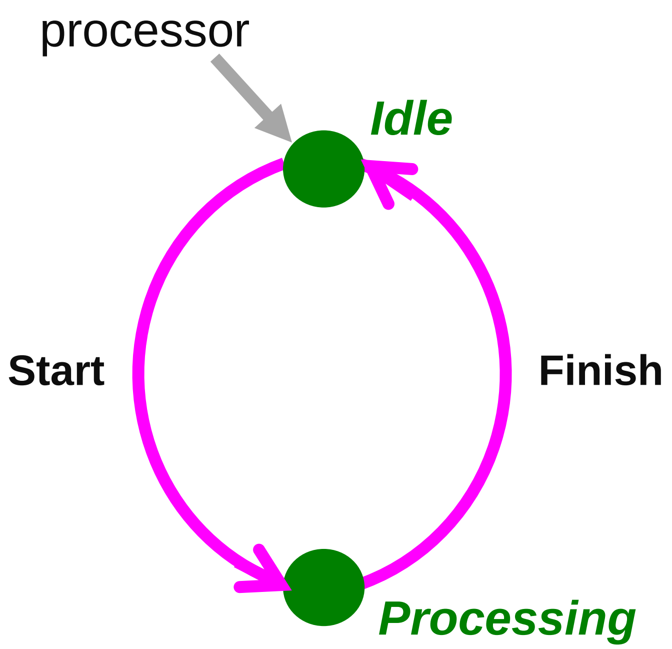


```
PROCDEF processor
  ::=
        Start   >->   Finish   >->   processor

 ENDDEF
```

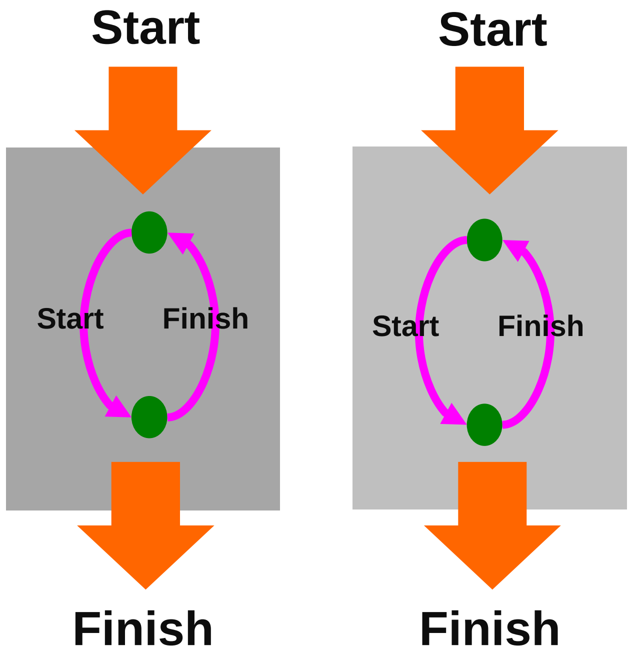
Example: Dispatcher-Processing System





```
STAUTDEF  processor
   ::=     …..
           Idle  –> Start  –> Processing
           Processing  –> Finish  –> Idle
ENDDEF
```

Example: Two Parallel Processors



Example: Two Parallel Processors

processor 1

$$\begin{tikzcd}
{} \arrow[d] \\
0 \arrow[d, "S1"] \\
1 \arrow[d, "F1"] \\
2
\end{tikzcd}$$

processor 2

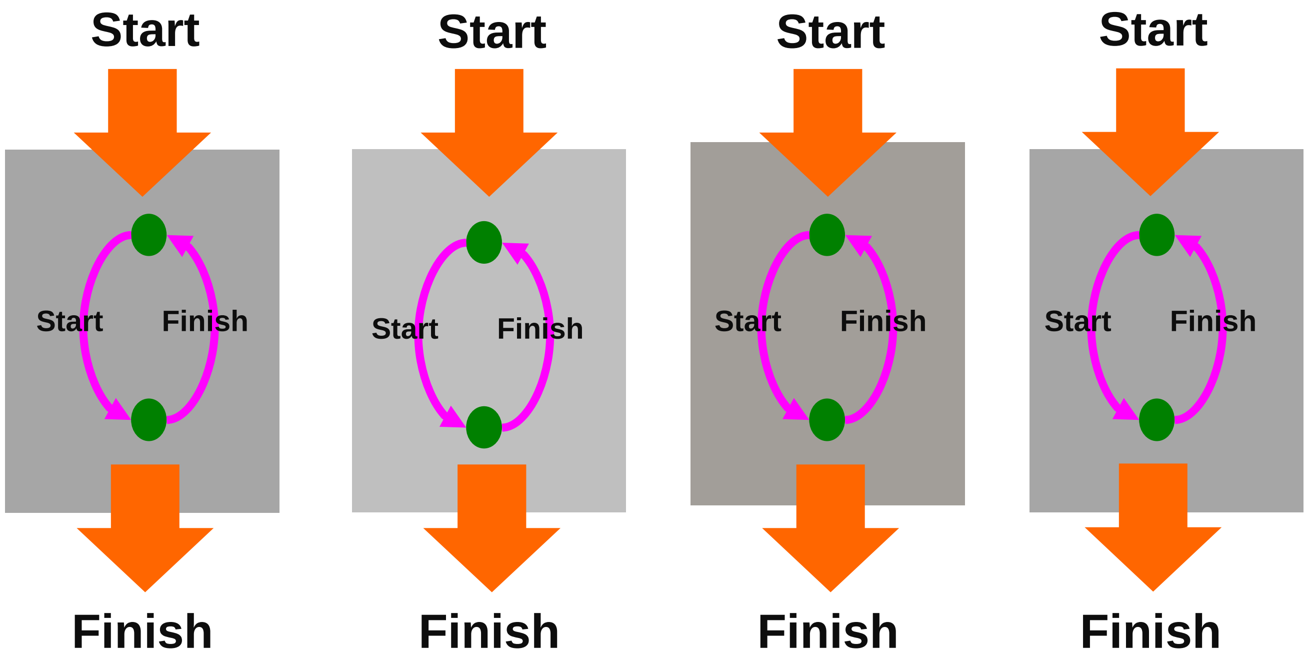
$$\begin{tikzcd}
{} \arrow[d] \\
0 \arrow[d, "S2"] \\
1 \arrow[d, "F2"] \\
2
\end{tikzcd}$$

$$\begin{tikzcd}
 & {} \arrow[dr] & & & \\
 & & 0{,}0 \arrow[dl, "S1"'] \arrow[dr, "S2"] & & \\
 & 1{,}0 \arrow[dl, "F1"'] \arrow[dr, "S2"] & & 0{,}1 \arrow[dl, "S1"'] \arrow[dr, "F2"] & \\
2{,}0 \arrow[dr, "S2"'] & & 1{,}1 \arrow[dl, "F1"'] \arrow[dr, "F2"] & & 0{,}2 \arrow[dl, "S1"] \\
 & 2{,}1 \arrow[dr, "F2"'] & & 1{,}2 \arrow[dl, "F1"] & \\
 & & 2{,}2 & &
\end{tikzcd}$$

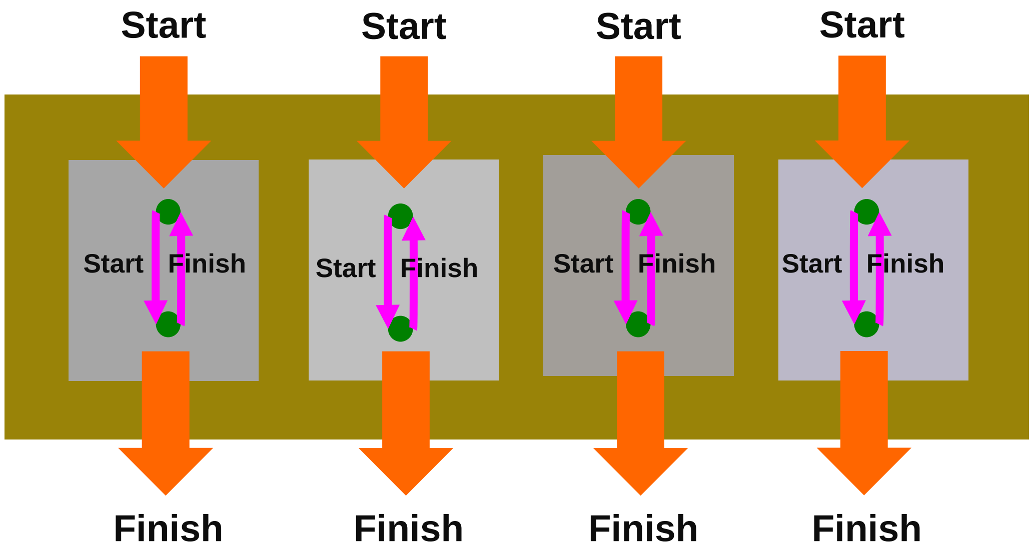
parallelism

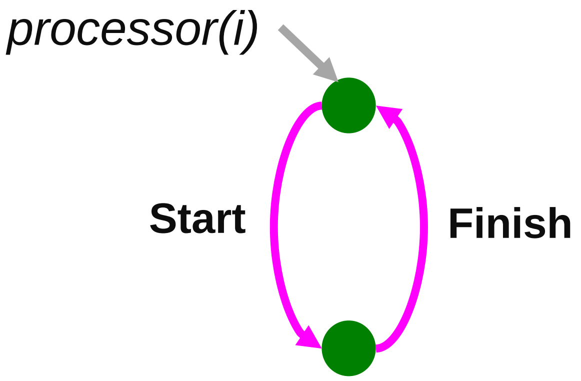
processor 1 ||| processor 2

Example: Four Parallel Processors



Example: Dispatcher-Processing System

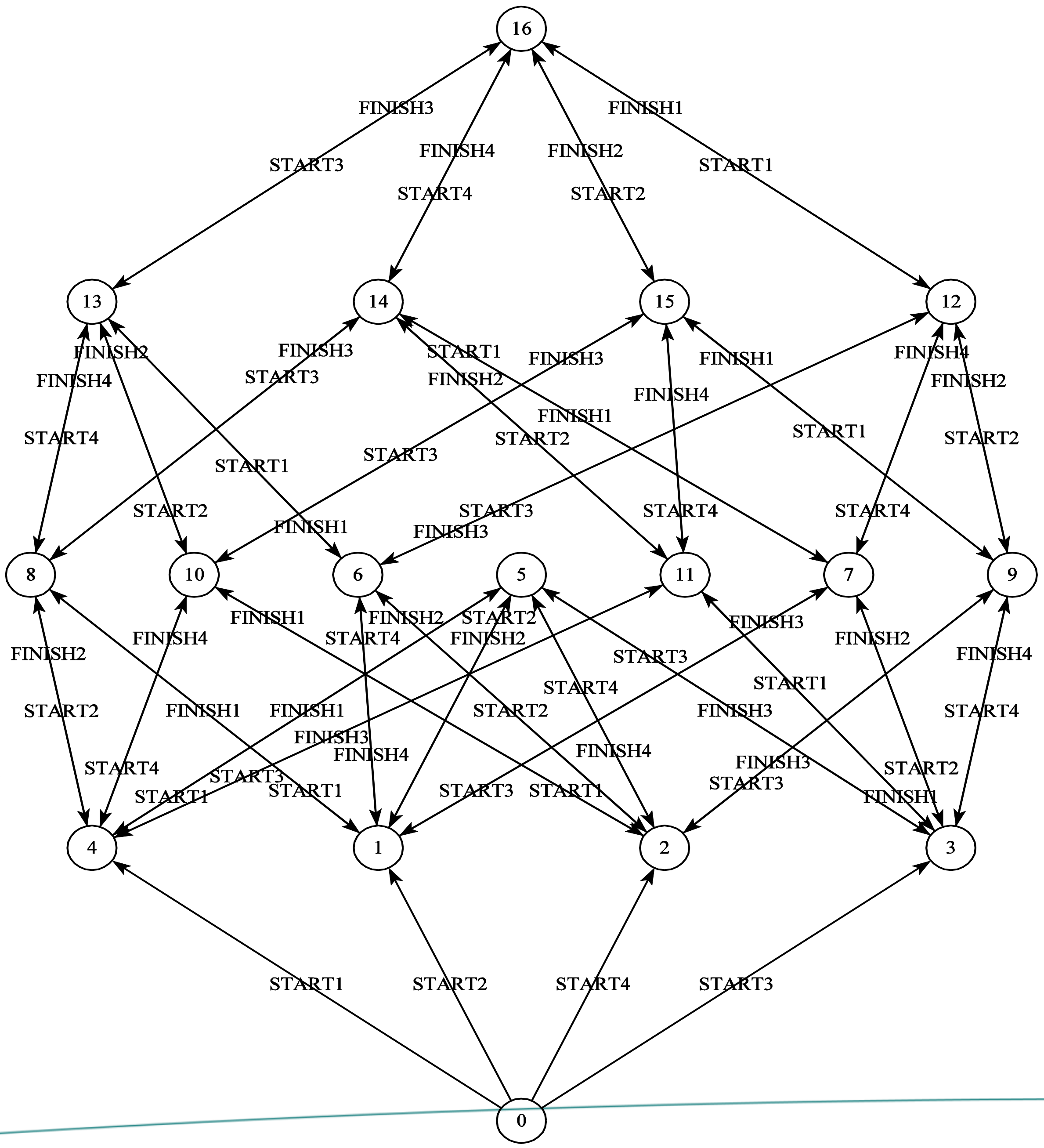




parallel composition

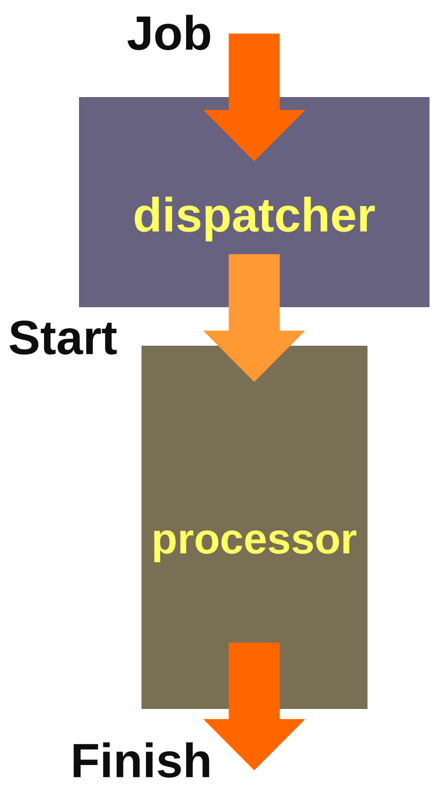
processors
::=
processor(1) ||| processor(2) ||| processor(3) ||| processor(4)

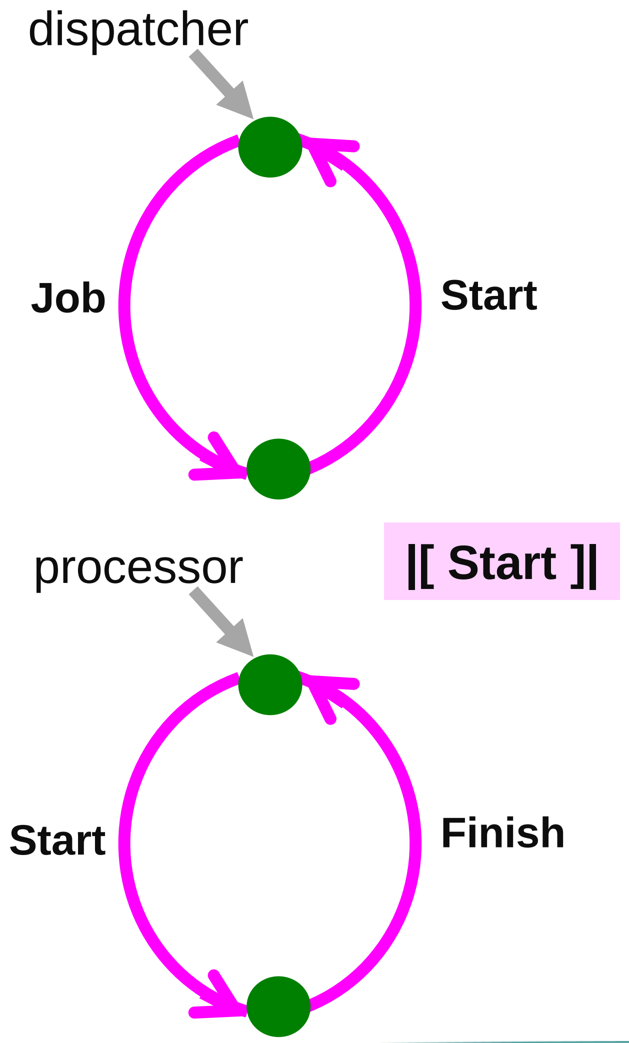
Example: Four Parallel Processors



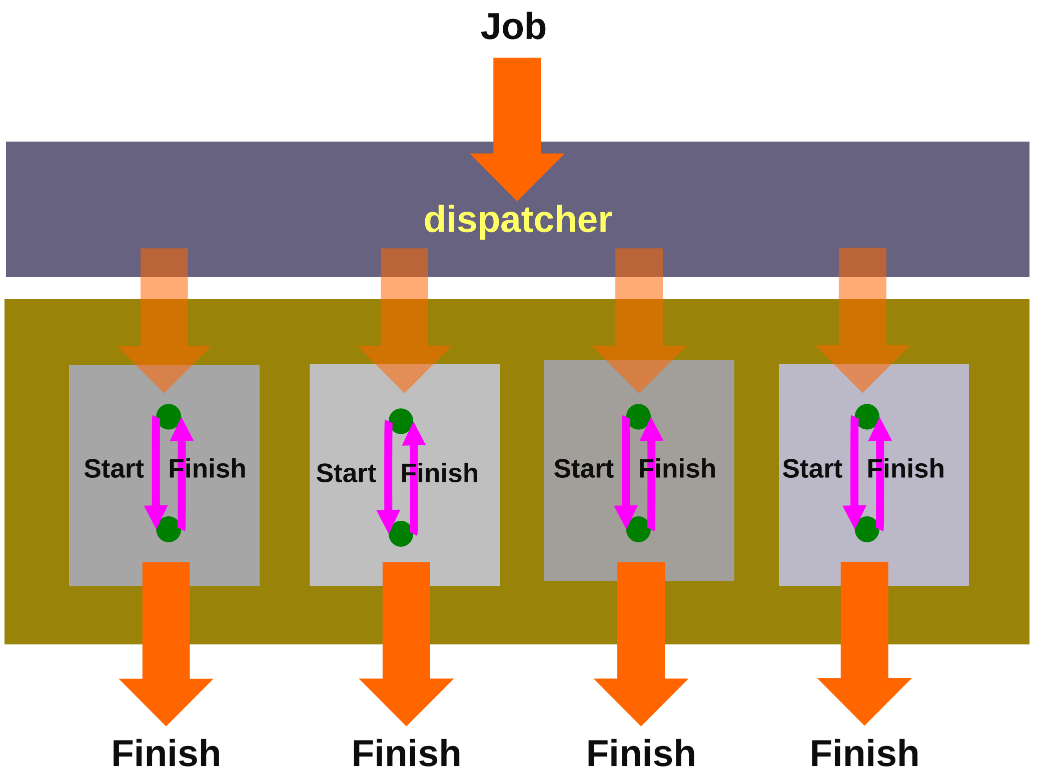
parallelism

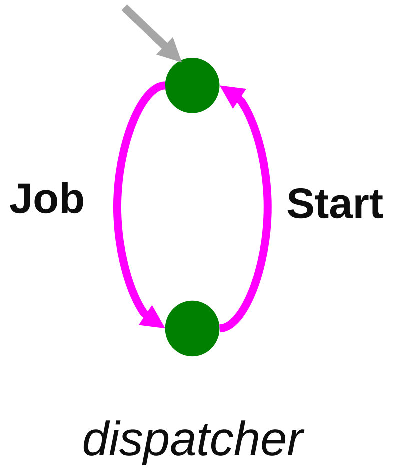
Example: Dispatcher-Processing System





Example: Dispatcher-Processing System

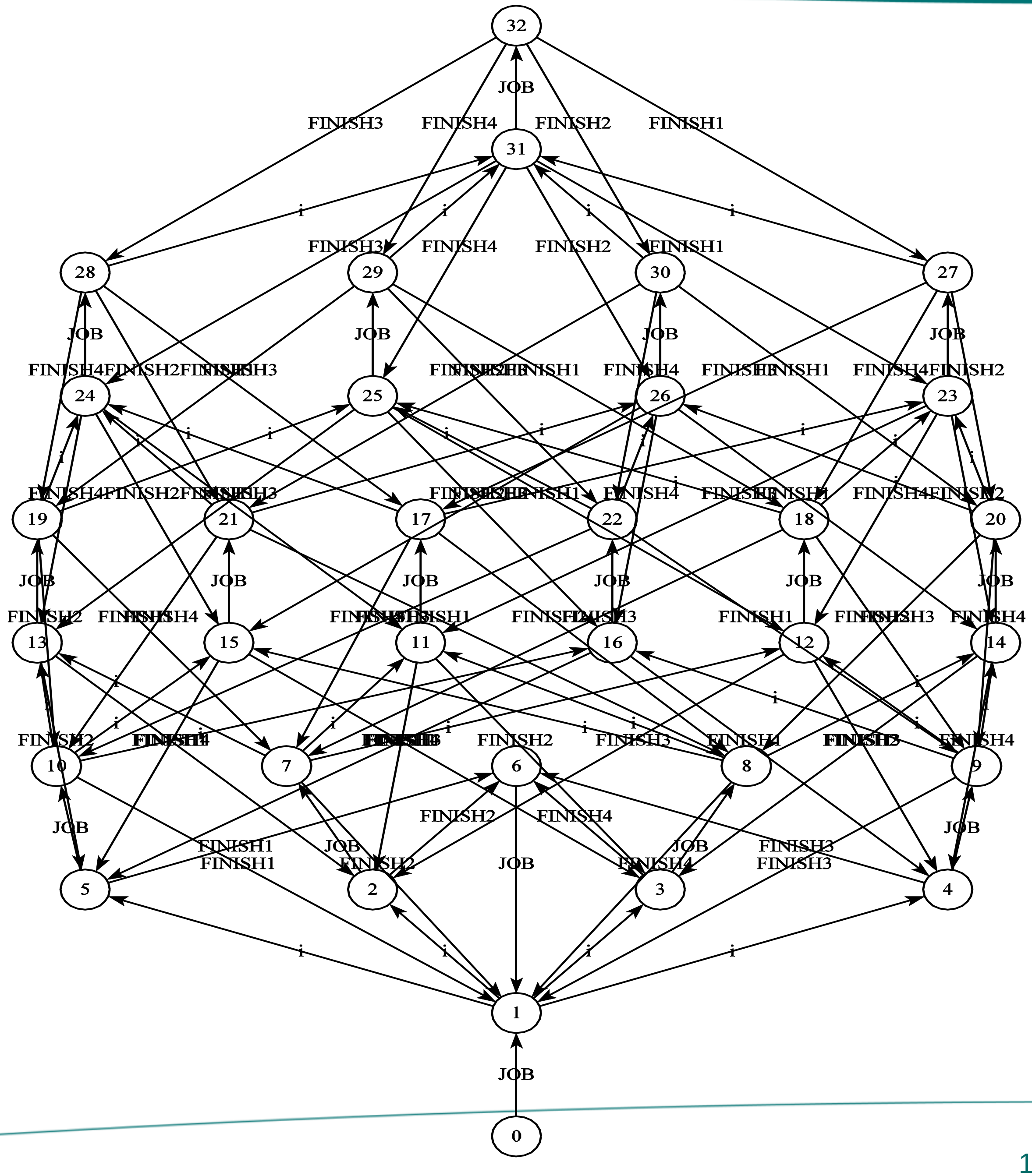




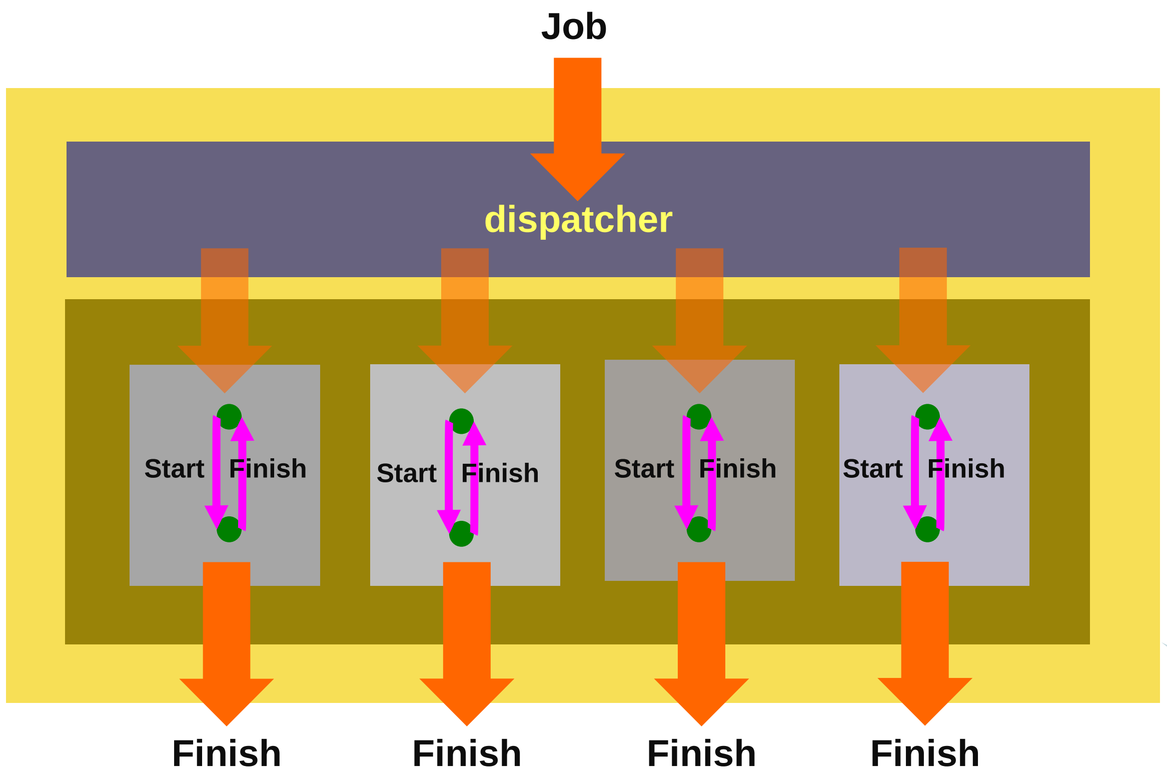
dispatch_procs
::=
dispatcher |[Start]| processors

composition

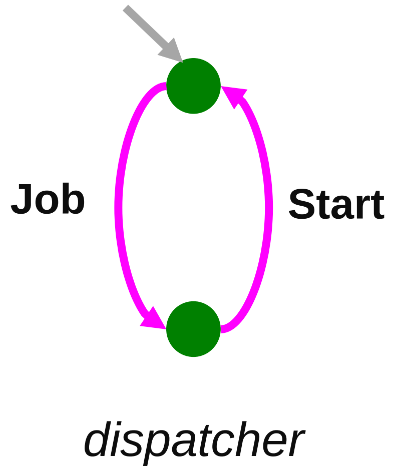
Example:
Dispatcher Processing System



Example: Dispatcher-Processing System

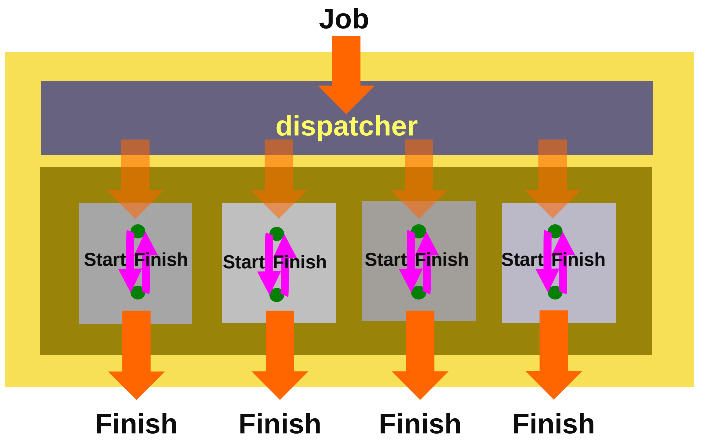


```
dispatch_procs
 ::=  HIDE  [ Start ]
      IN
           dispatcher   |[  Start  ]|   processors
      NI
```

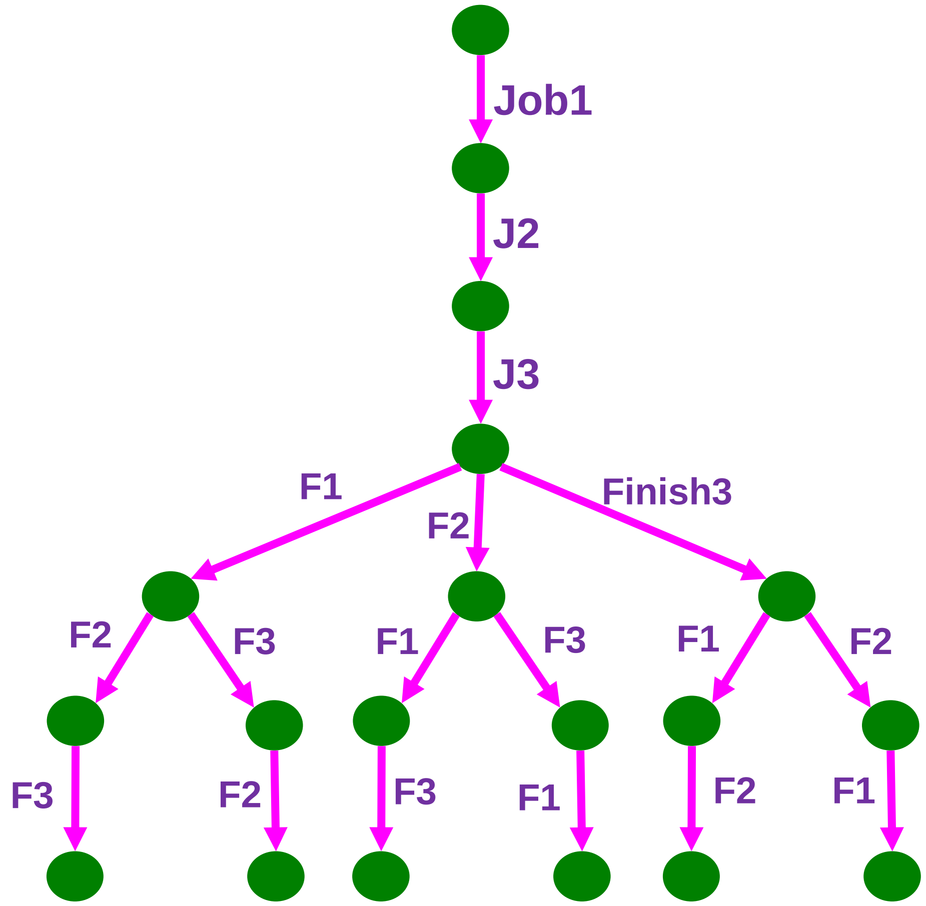


abstraction

Example: Dispatcher-Processing System

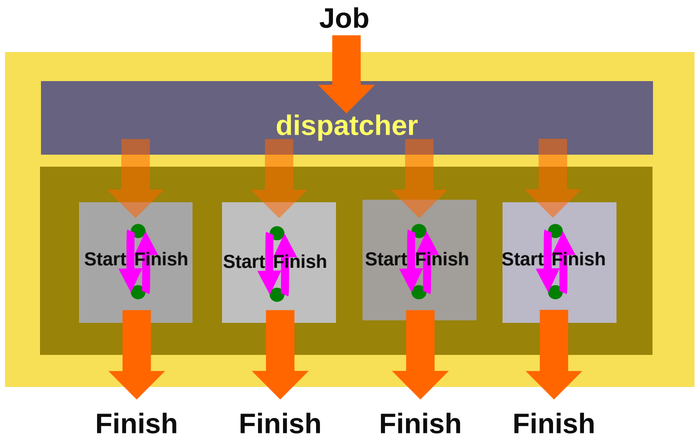


Inputs: Job1, Job2, Job3:

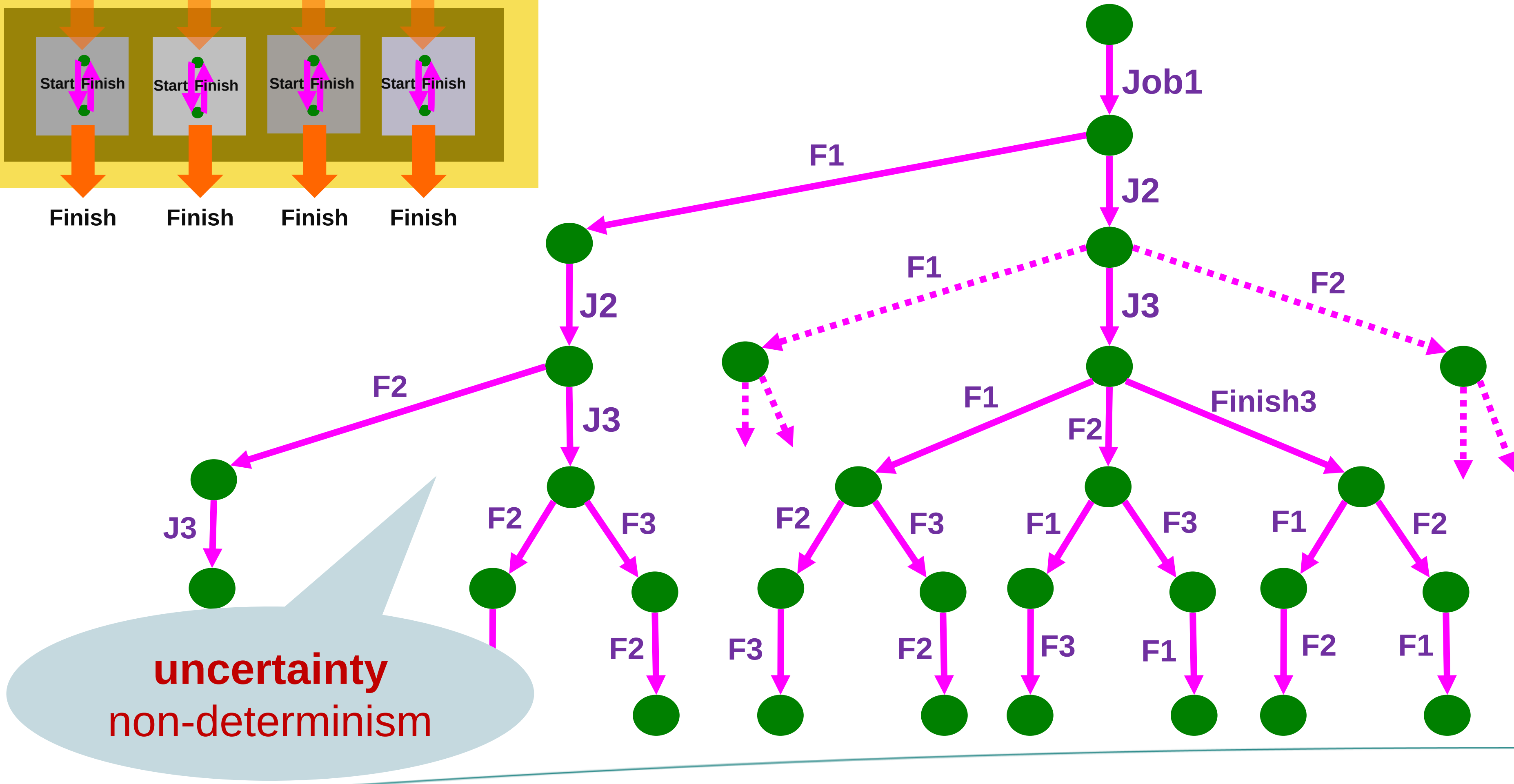


uncertainty
no unique expected result

Example: Dispatcher-Processing System



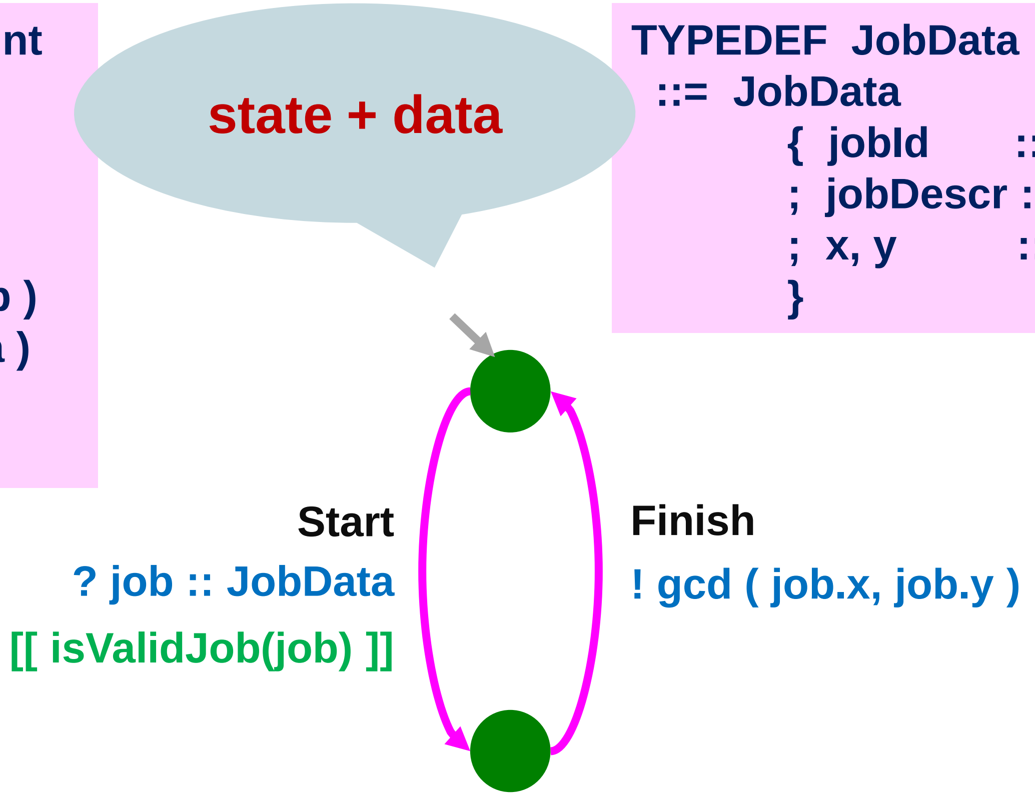
Inputs: Job1, Job2, Job3:

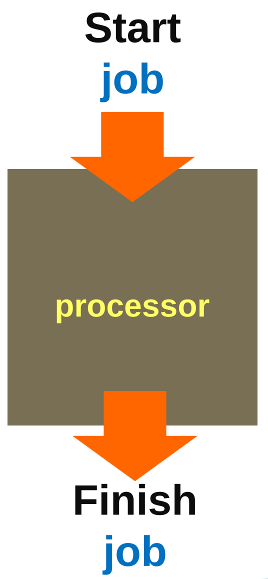


Example: Dispatcher-Processing System

```
FUNCDEF  gcd ( a, b :: Int ) :: Int
 ::=
      IF a == b
      THEN  a
      ELSE  IF a > b
              THEN  gcd ( a - b, b )
              ELSE  gcd ( a, b - a )
              FI
      FI
```

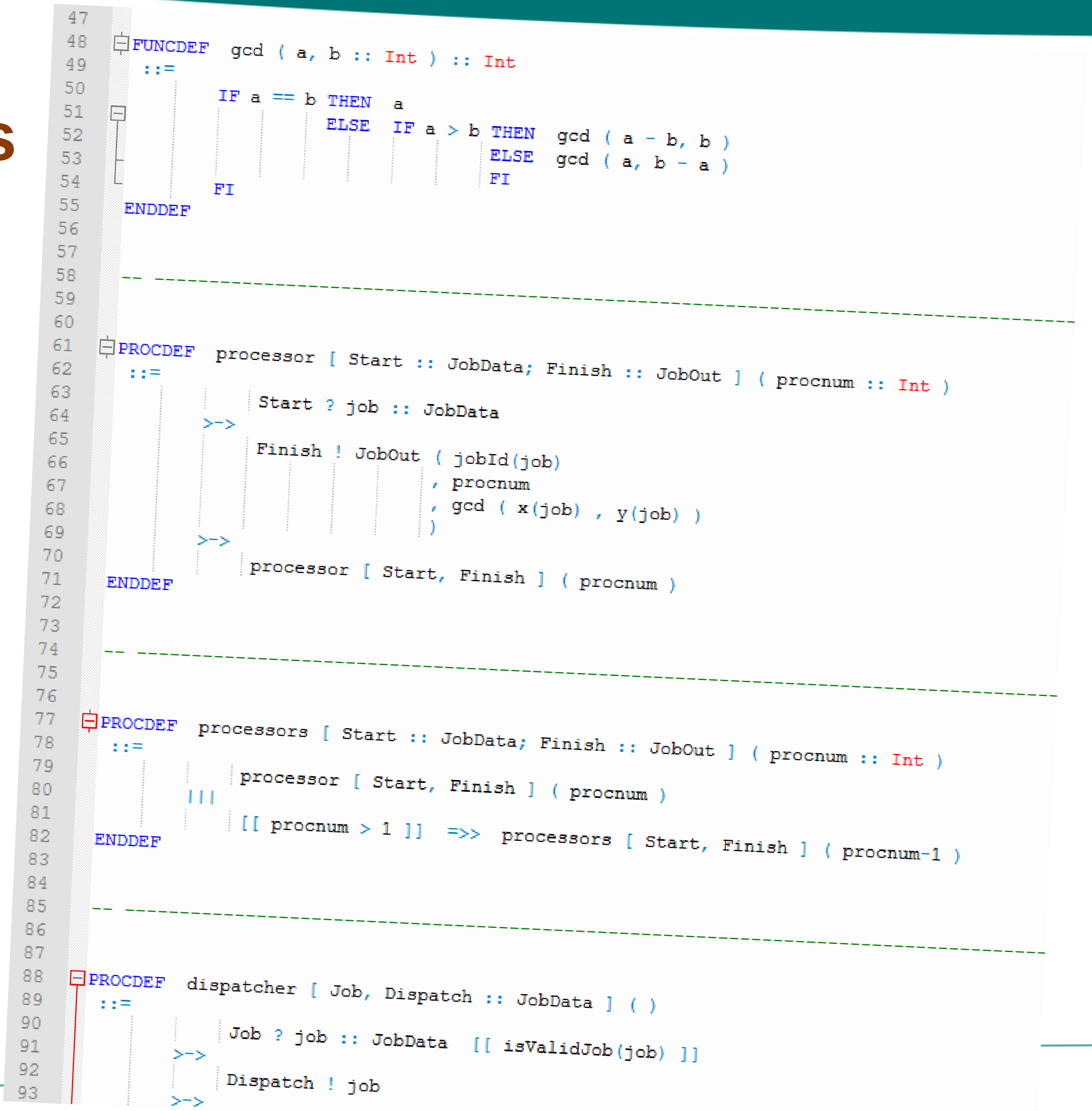
```
TYPEDEF  JobData
  ::=  JobData
          {  jobId       :: Int
          ;  jobDescr :: String
          ;  x, y          :: Int
          }
```





```
FUNCDEF  isValidJob ( jobdata :: JobData )  :: Bool
  ::=
        jobdata.jobId > 0
    /\  strinre ( jobdata.jobDescr, REGEX('[A-Z][0-9]{2}[a-z]+') )
    /\  . . . . .
```

TorXakis Model



```
FUNCDEF  gcd ( a, b :: Int ) :: Int
  ::=
        IF a == b THEN  a
                  ELSE  IF a > b THEN  gcd ( a - b, b )
                                 ELSE  gcd ( a, b - a )
                                 FI
        FI
ENDDEF

-- ----------------------------------------------------------------------

PROCDEF  processor [ Start :: JobData; Finish :: JobOut ] ( procnum :: Int )
  ::=
           Start ? job :: JobData
      >->
           Finish ! JobOut ( jobId(job)
                           , procnum
                           , gcd ( x(job) , y(job) )
                           )
      >->
           processor [ Start, Finish ] ( procnum )
ENDDEF

-- ----------------------------------------------------------------------

PROCDEF  processors [ Start :: JobData; Finish :: JobOut ] ( procnum :: Int )
  ::=
           processor [ Start, Finish ] ( procnum )
      |||
           [[ procnum > 1 ]]  =>>  processors [ Start, Finish ] ( procnum-1 )
ENDDEF

-- ----------------------------------------------------------------------

PROCDEF  dispatcher [ Job, Dispatch :: JobData ] ( )
  ::=
           Job ? job :: JobData  [[ isValidJob(job) ]]
      >->
           Dispatch ! job
      >->
```

TorXakis

Demo