Towards a parallel search for solutions of non-deterministic computations

Fabian Reck Sebastian Fischer

October 12, 2009

non-determinism in Haskell

- purely functional, lazy
- no built-in non-determinism
- non-determinism is modeled by data structures
- abstracted by monads

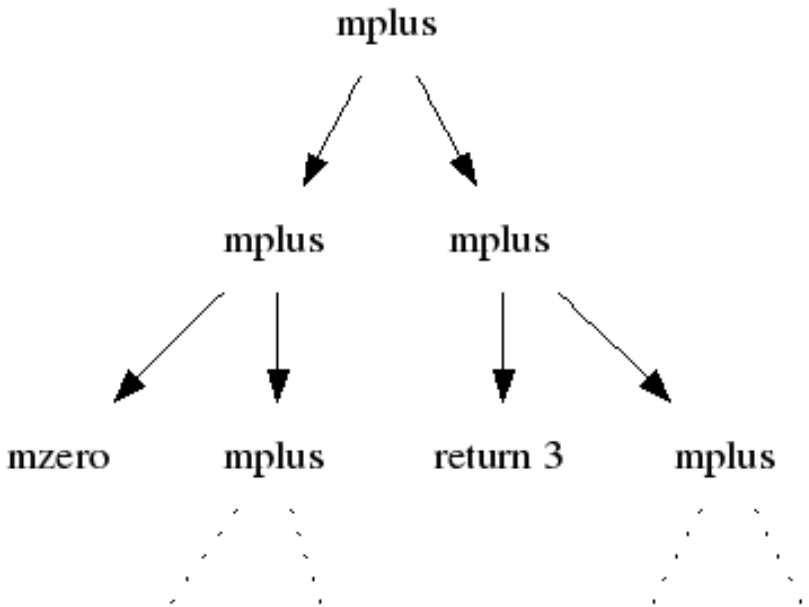
non-determinism in Haskell

- `return x`
  - a non-deterministic computation with the single result `x`
- `mzero`
  - a failing computation
- `e1 ‘mplus‘ e2`
  - combines non-deterministic computations
- `e >>= f`
  - applies the non-deterministic function `f` non-deterministically to a result of `e`

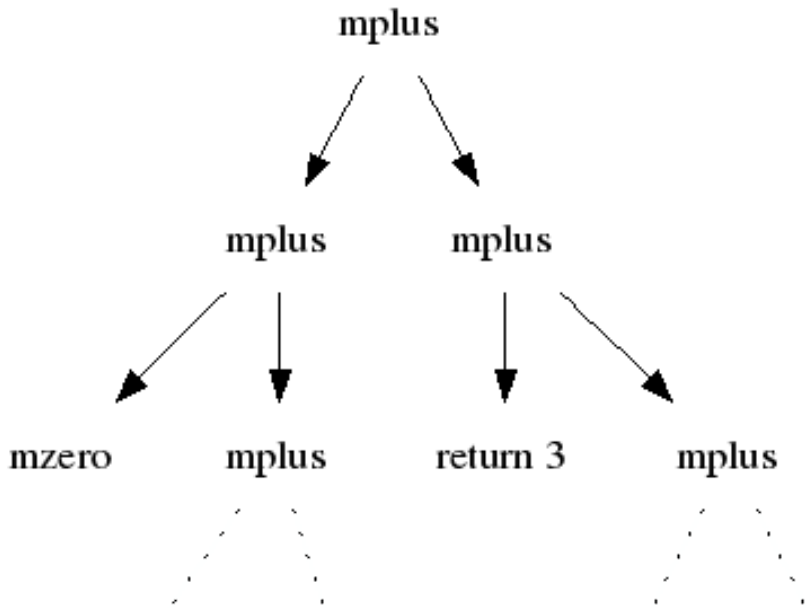
example

```
perm ::  [a] -> m [a]
perm [] = return []
perm (x:xs) = perm xs >>= insert x

insert :: a -> [a] -> m [a]
insert x [] = return [x]
insert x (y:ys) = return (x:y:ys)
                  `mplus` (insert x ys >>= (return . (y:)))
```

an alternative representation of non-determinism

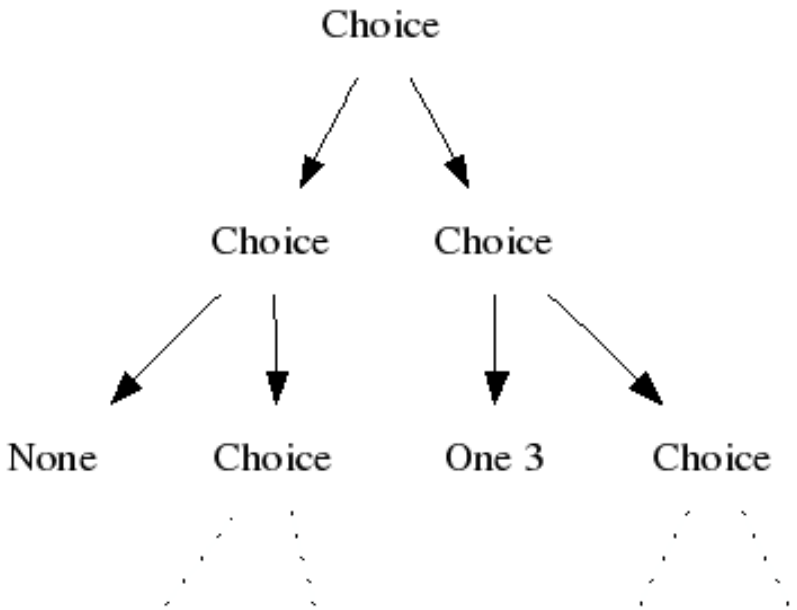


an alternative representation of non-determinism




```
data SearchTree a = None
                  | One a
                  | Choice (SearchTree a) (SearchTree a)
```

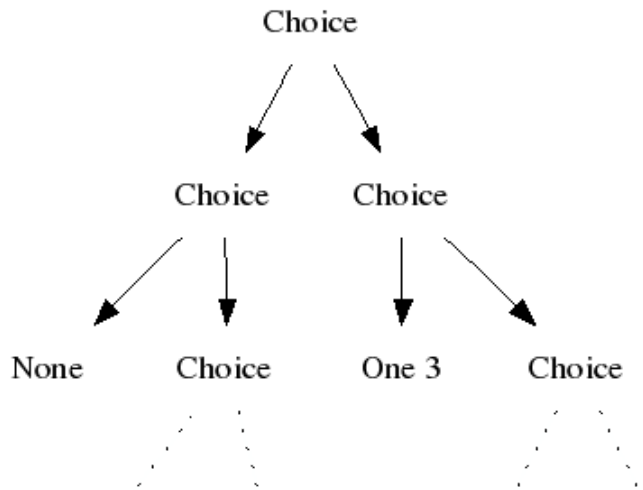
an alternative representation of non-determinism



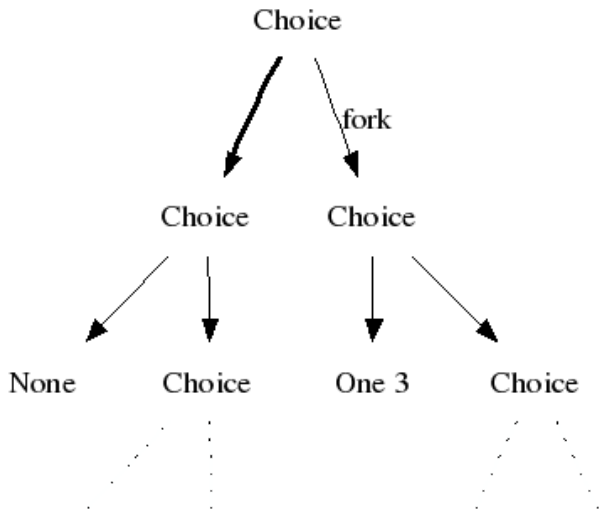

```
data SearchTree a = None
                  | One a
                  | Choice (SearchTree a) (SearchTree a)
```

```
return = One

mzero = None

mplus = Choice

None          >>= _ = None
One x         >>= f = f x
Choice t1 t2 >>= f = Choice (t1 >>= f)
                            (t2 >>= f)
```

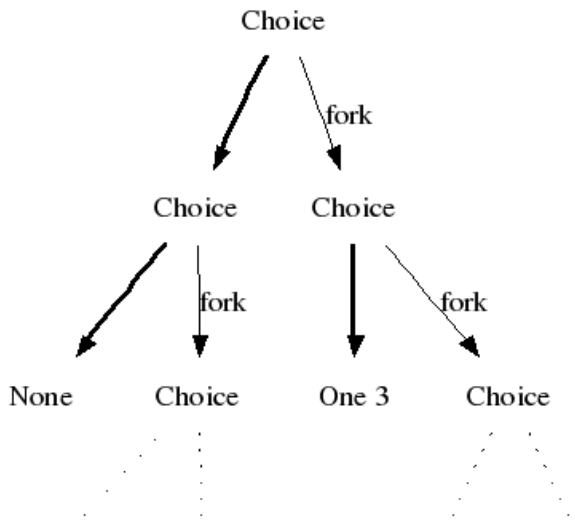
dividing the tree



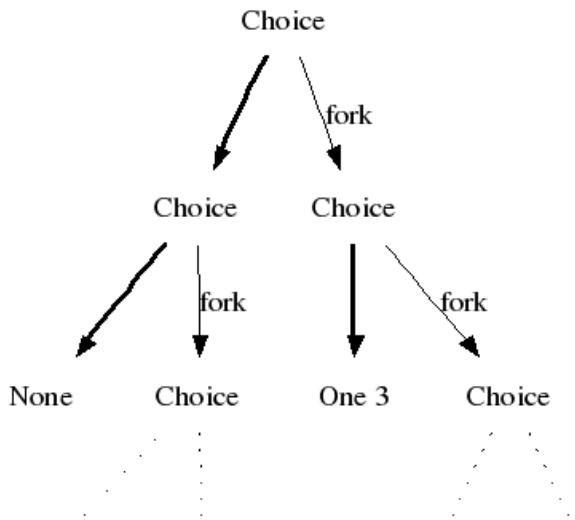
dividing the tree



dividing the tree



dividing the tree

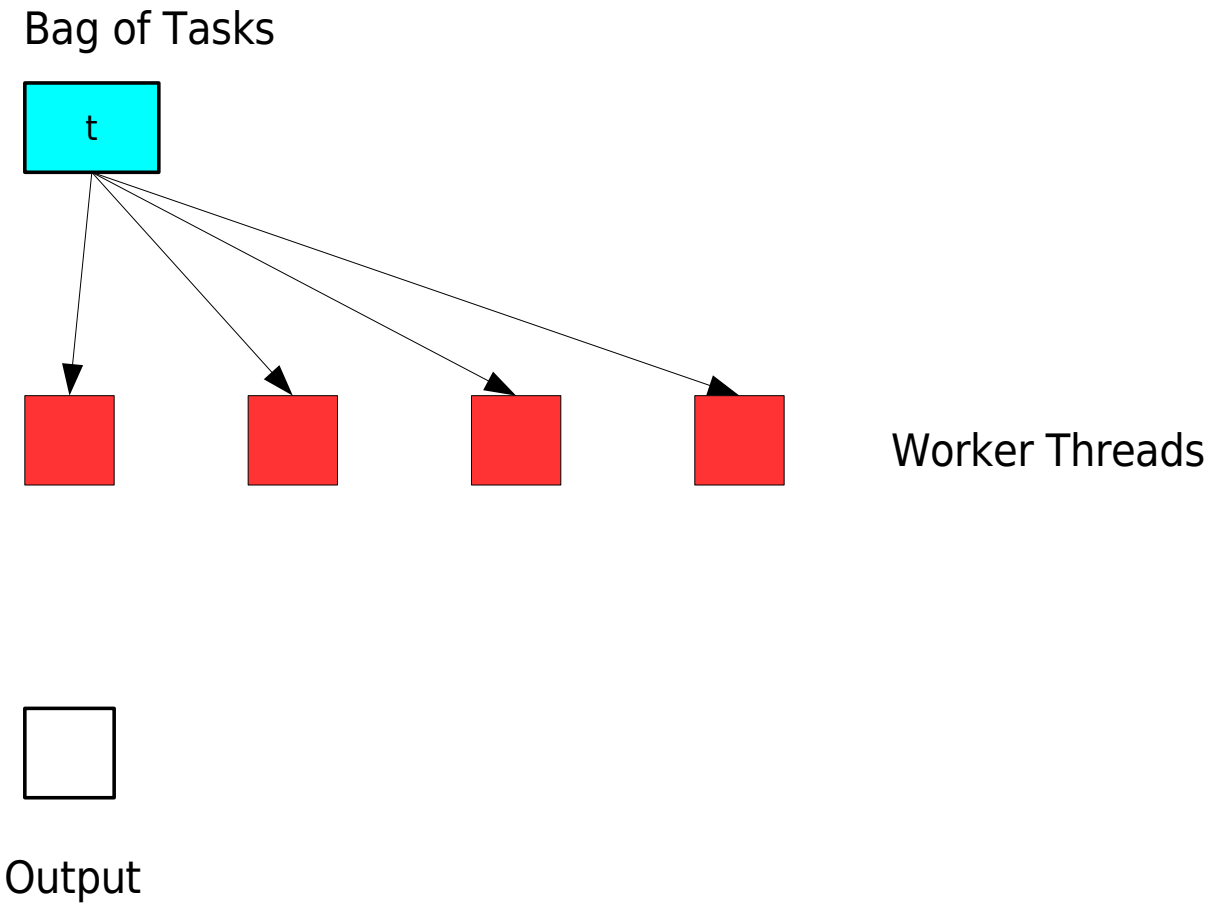


- fork new threads down to a given depth n
- the result of the threads is combined
- best choice for n depends on the number of cores and the shape of the tree
- sequential search has to be strict

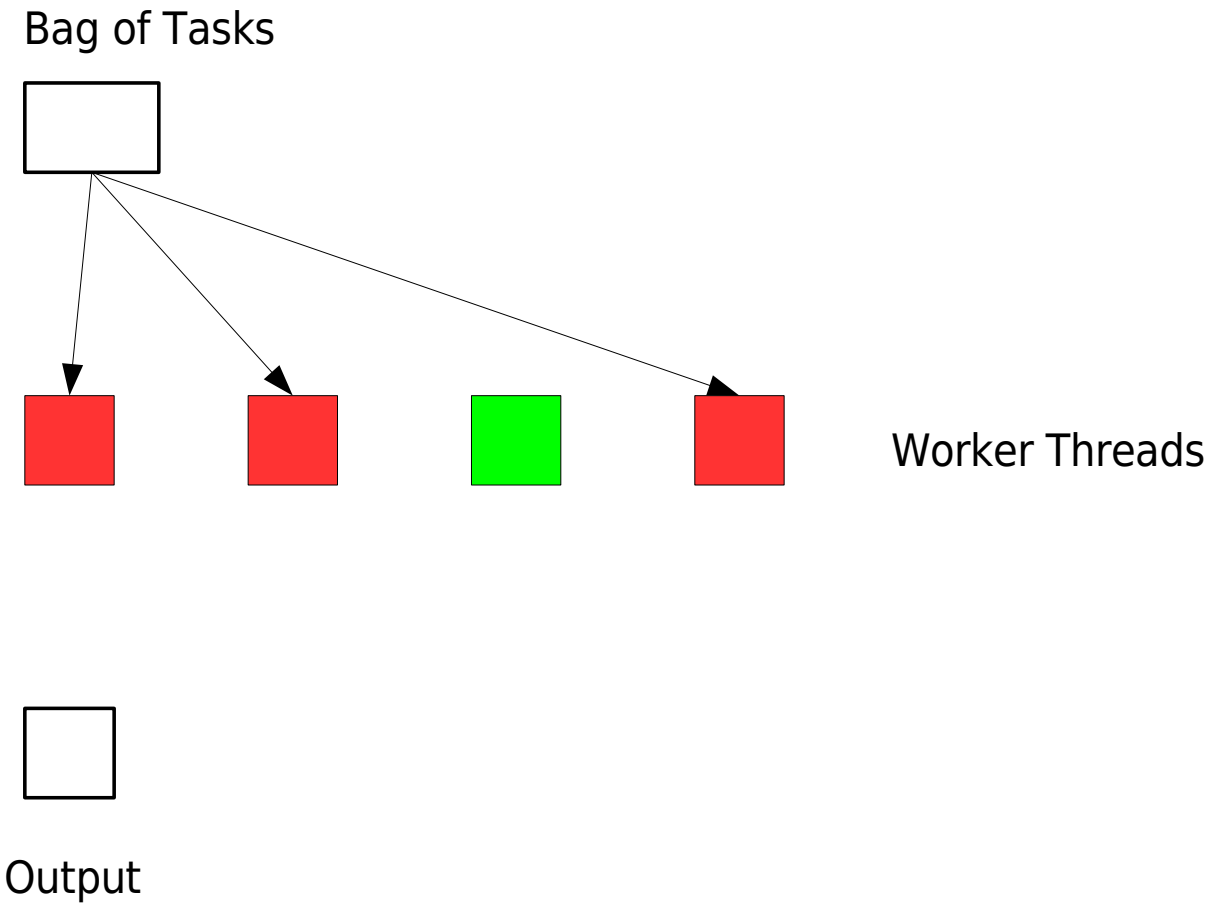
dividing the tree

- overhead is limited
- works only for finite trees

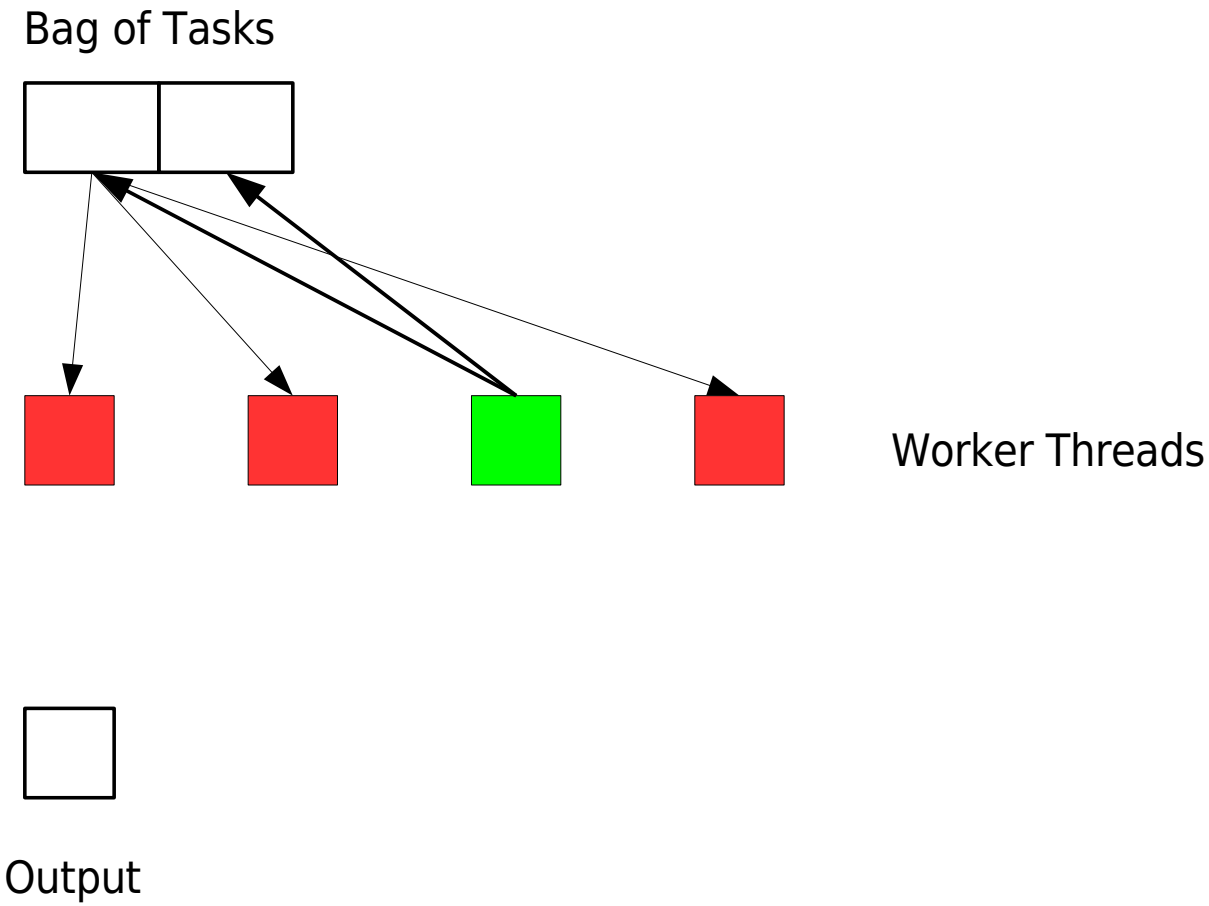
bag of tasks



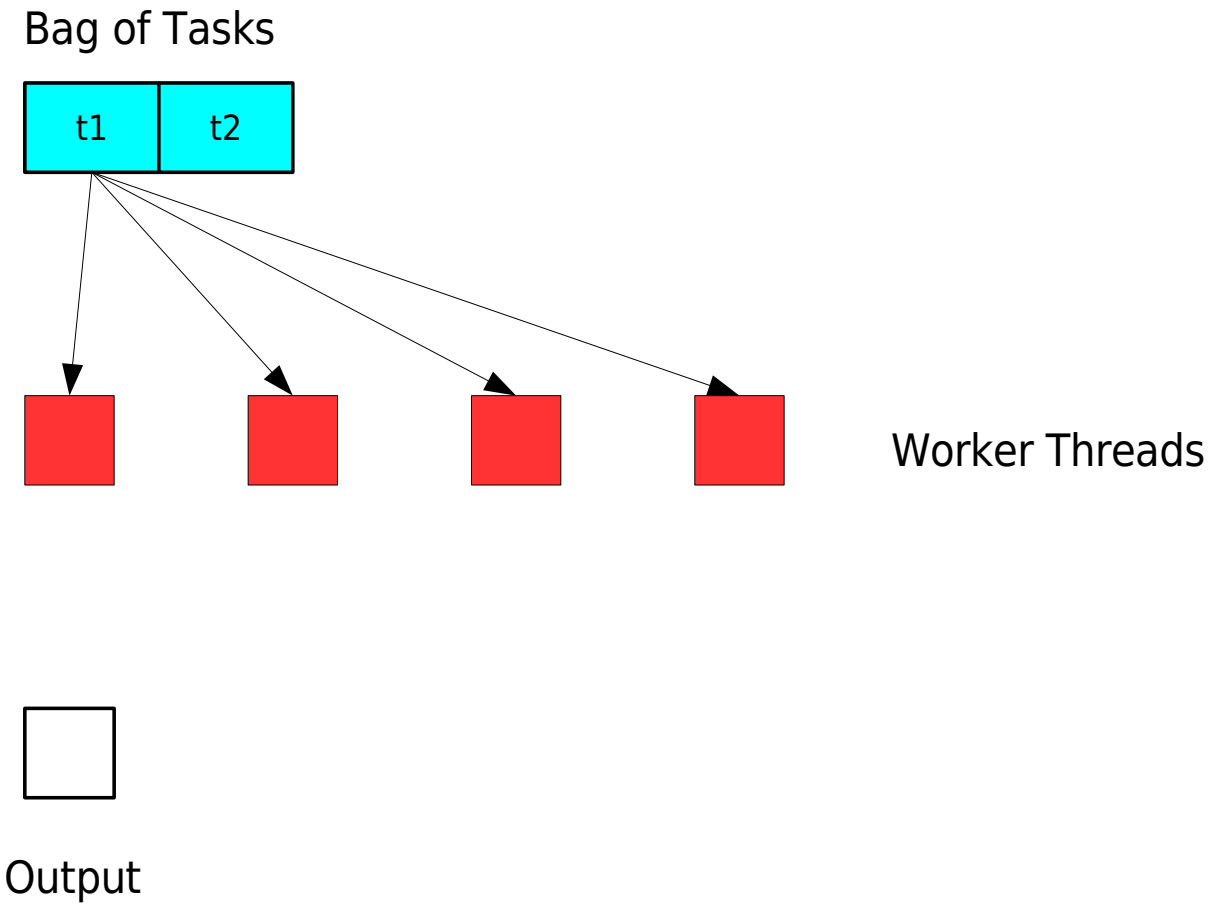
bag of tasks



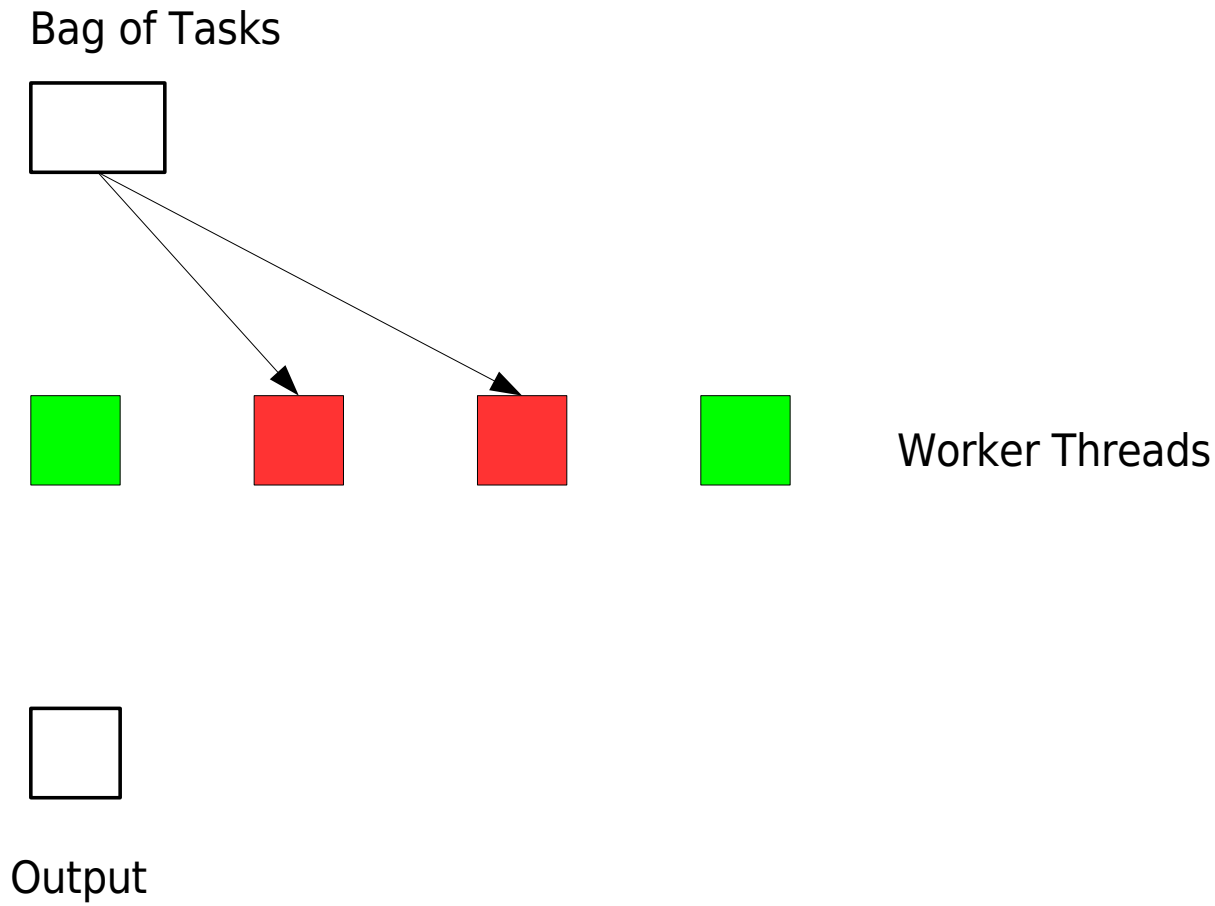
bag of tasks



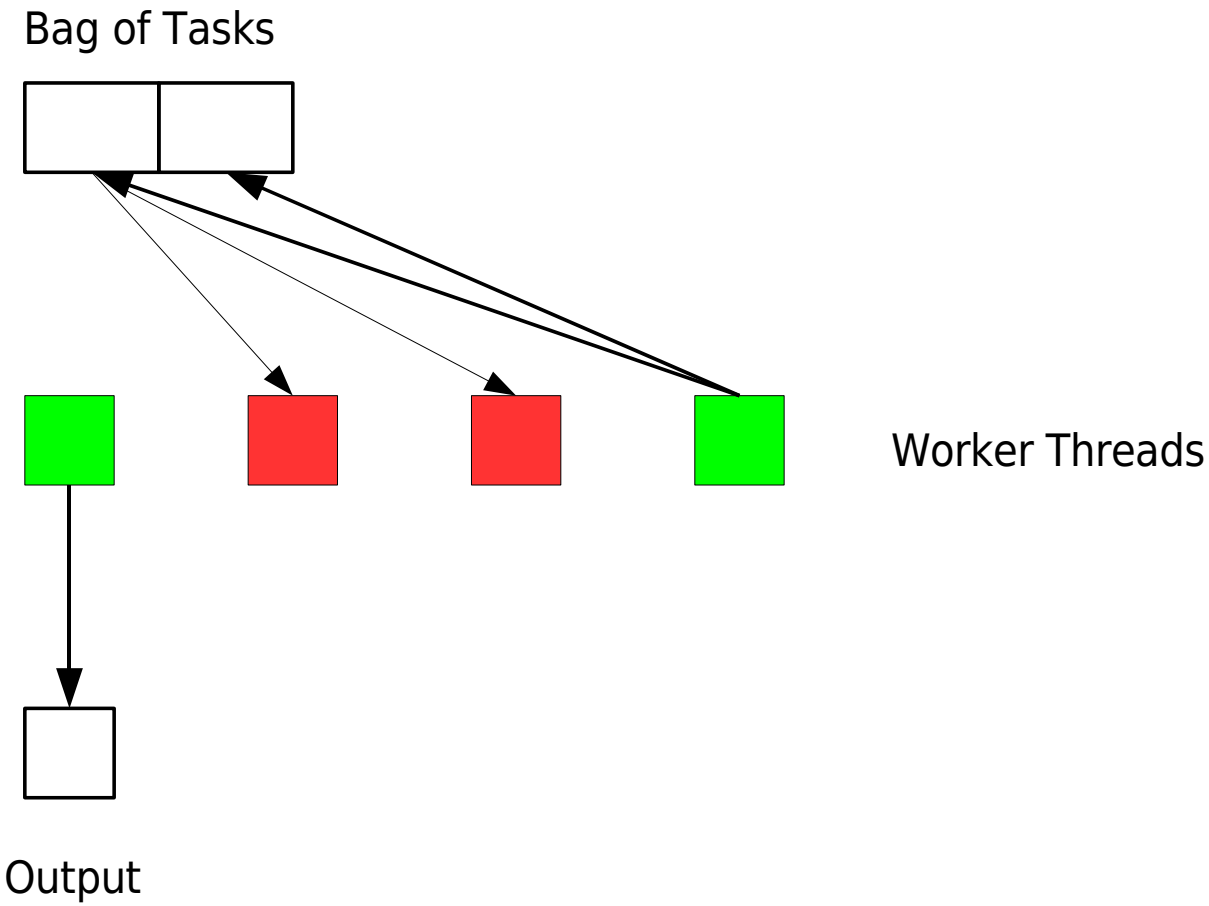
bag of tasks



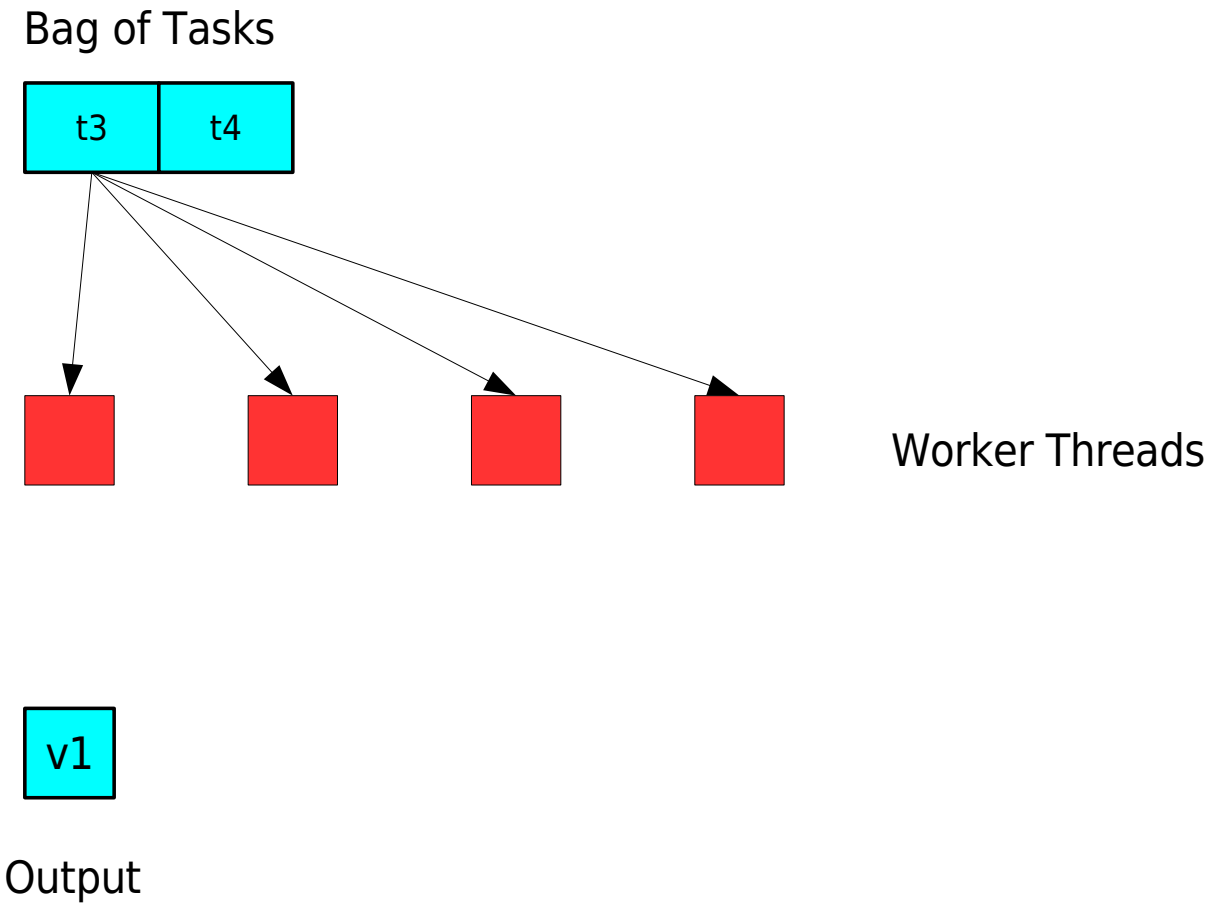
bag of tasks



bag of tasks



bag of tasks



bag of tasks

- works for infinite trees (until memory is exhausted)
- search is complete
- much synchronisation
- exponential space complexity

glasgow parallel haskell

```
par :: a -> b -> b
```

- returns its second argument
- first argument is stored in a *spark pool*
- *sparks* are evaluated by idle processors

search with GPH

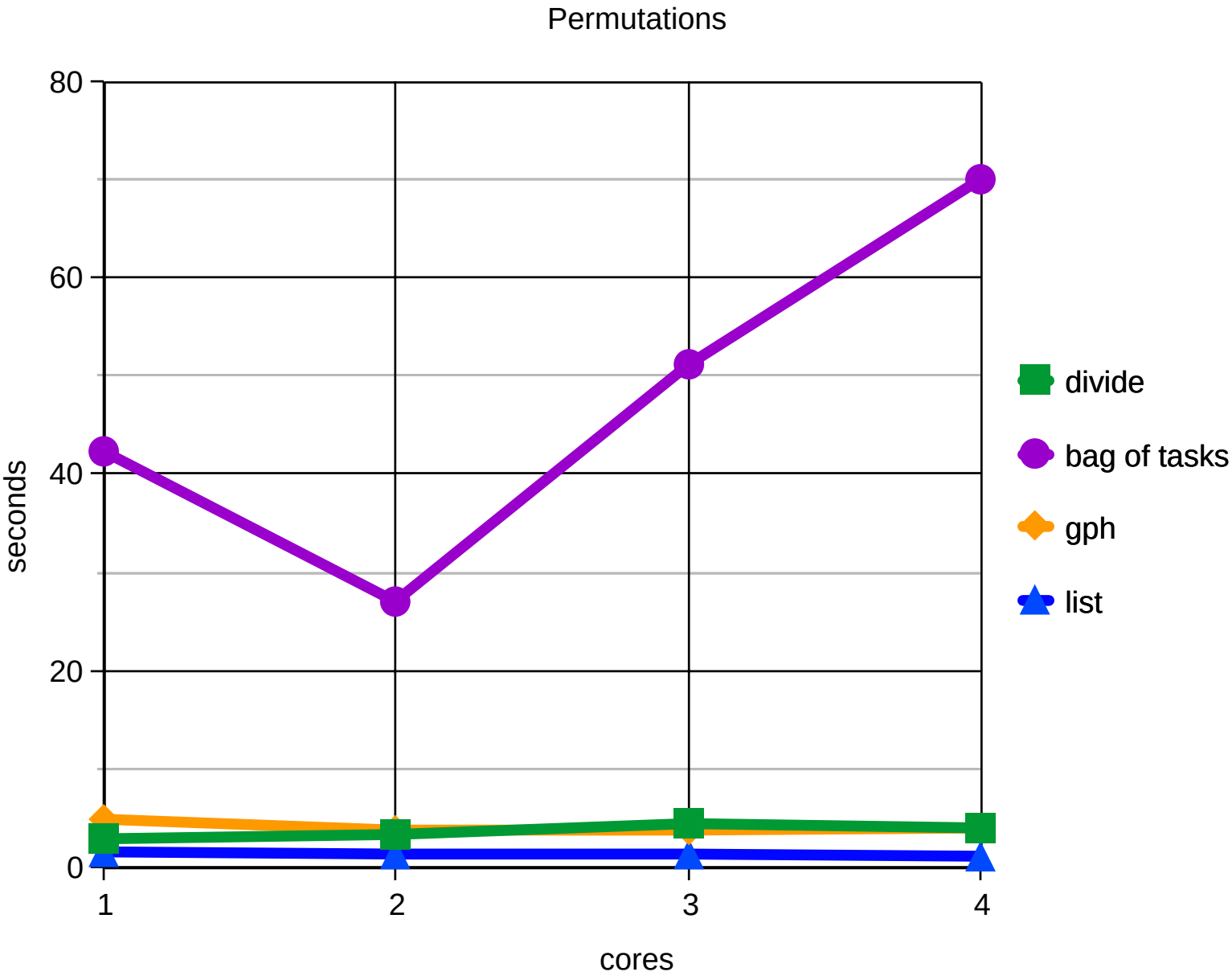
```
search :: SearchTree a -> [a]
search None      = []
search (One x) = [x]
search (Choice l r) = rs `par` (search l ++ rs)
  where rs = search r
```

- no explicit threads needed
- search is incomplete

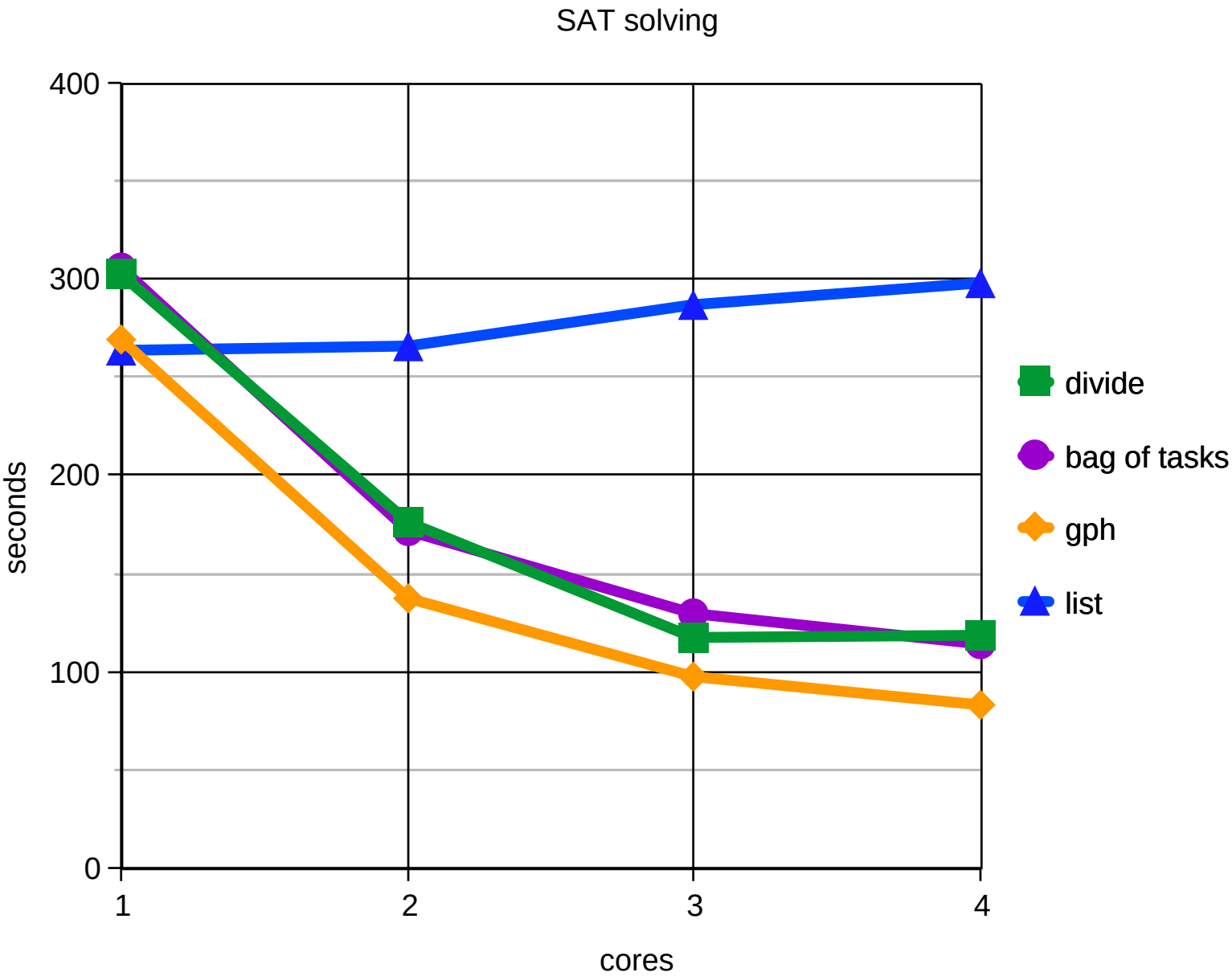
benchmarks

1. permutations
   - large number of results
   - little effort to compute a node in the tree
   - no failures
2. SAT solving with Davis-Putnam-Logemann-Loveland
   - no results (with the tested instance)
   - some effort to compute a node in the tree
   - large number of failures

benchmarks


Permutations
80
60
40
20
0
seconds
1
2
3
4
cores
divide
bag of tasks
gph
list

benchmarks


SAT solving
400
300
200
100
0
seconds
1
2
3
4
cores
divide
bag of tasks
gph
list

summary

- three approaches to parallel search
  1. dividing the tree
  2. bag of tasks
  3. glasgow parallel haskell
- significant speedups
- room for improvements