A Skeleton for Distributed Work Pools in Eden

Mischa Dieterle[1] Jost Berthold[2] Rita Loogen[1]

[1]Fachbereich Mathematik und Informatik
Philipps Universität Marburg

[2]Datalogisk Institut
University of Copenhagen

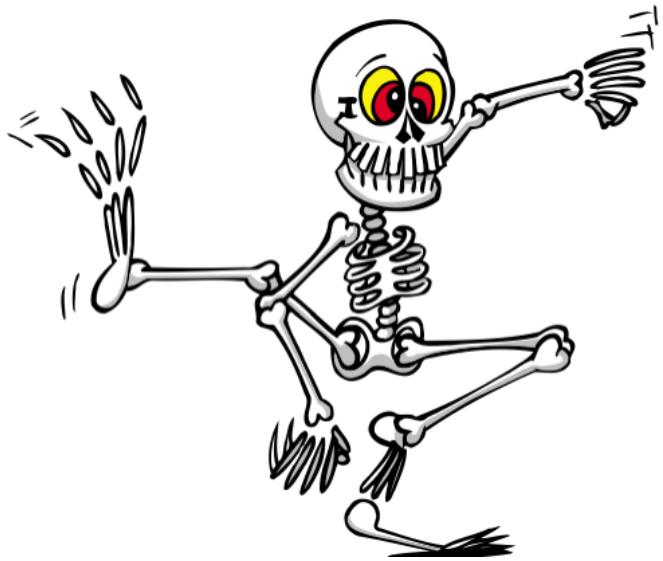
Outline

- *Introduction*
- *Implementation*
- *Applications*

Algorithmic Skeletons

Patterns of parallel computations
→ in functional languages:

- Parallel higher-order function



Typical patterns:

- Parallel maps: farm
- Work pools / master-worker systems
- Topology skeletons: pipeline, ring, torus, grid, ...
- Divide and conquer

Eden Features

- Haskell extension
- Distributed memory
- Explicitly defined processes
  - `process :: (Trans a, Trans b) ⇒ (a → b) → Process a b`
  - `spawn :: (Trans a, Trans b) ⇒ [Process a b] → [a] → [b]`
- Implicit communication
- Communication semantics: Typeclass `Trans`
  - Data sent in normal form
  - Lists sent as streams ...

Eden Features

- Haskell extension
- Distributed memory
- Explicitly defined processes
  - `process :: (Trans a, Trans b) ⇒ (a → b) → Process a b`
  - `spawn :: (Trans a, Trans b) ⇒ [Process a b] → [a] → [b]`
- Implicit communication
- Communication semantics: Typeclass `Trans`
  - Data sent in normal form
  - Lists sent as streams ...

Eden Features

- Haskell extension
- Distributed memory
- Explicitly defined processes
  - `process :: (Trans a, Trans b) ⇒ (a → b) → Process a b`
  - `spawn :: (Trans a, Trans b) ⇒ [Process a b] → [a] → [b]`
- Implicit communication
- Communication semantics: Typeclass `Trans`
  - Data sent in normal form
  - Lists sent as streams ...

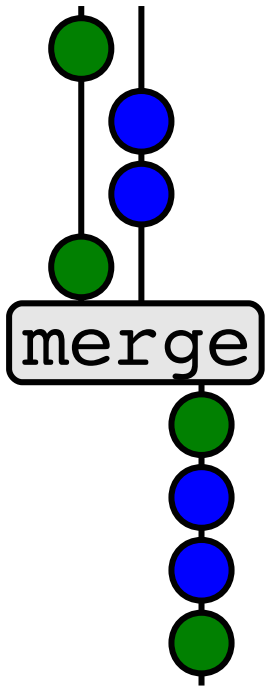
Eden Features

- Haskell extension
- Distributed memory
- Explicitly defined processes
  - `process :: (Trans a, Trans b) ⇒ (a → b) → Process a b`
  - `spawn :: (Trans a, Trans b) ⇒ [Process a b] → [a] → [b]`
- Implicit communication
- Communication semantics: Typeclass `Trans`
  - Data sent in normal form
  - Lists sent as streams ...

Nonfunctional Extensions

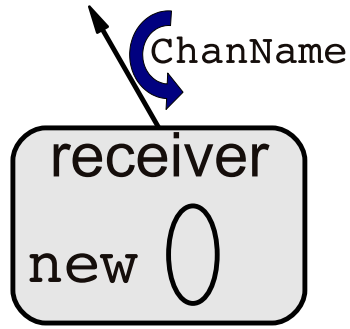
Nondeterminism:

```
merge :: [[a]] → [a]
```

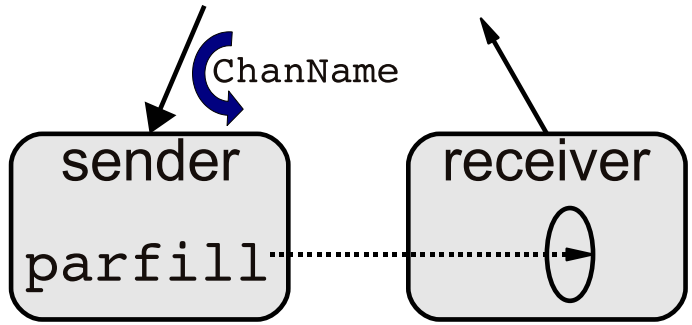


Explicit Communication:

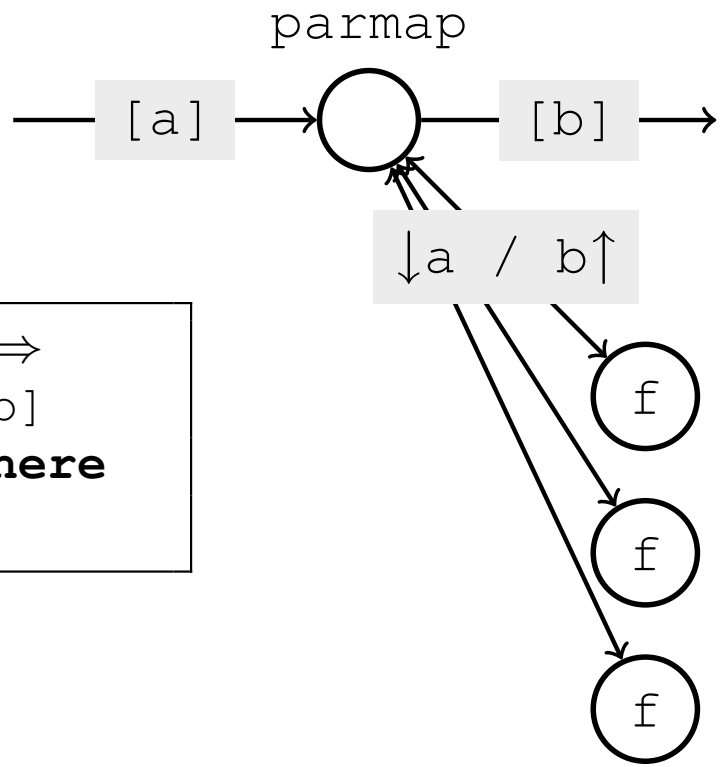
```
new :: (ChanName a → a → b) → b
```




```
parfill :: ChanName a → a → b → b
```



Algorithmic Skeletons in Eden

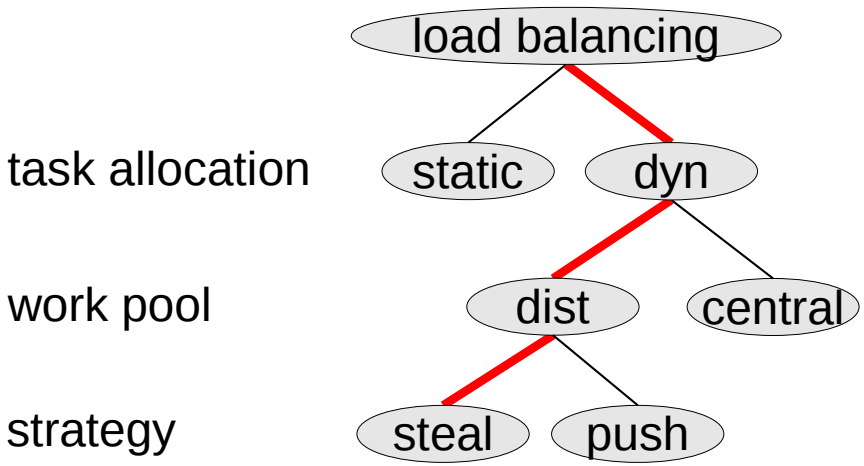


Example:

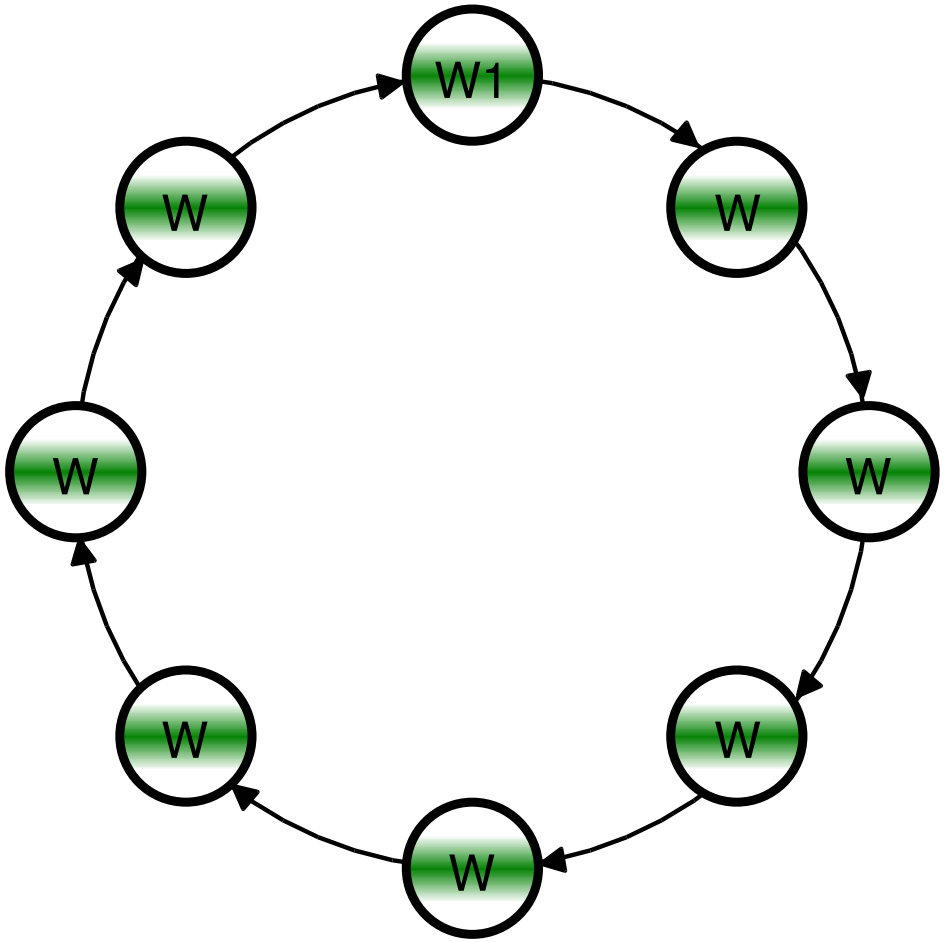
```
parmap :: (Trans a,Trans b) ⇒
          (a→b) → [a] → [b]
parmap f as = spawn pfs as where
 pfs = repeat (process f)
```

Load Balancing

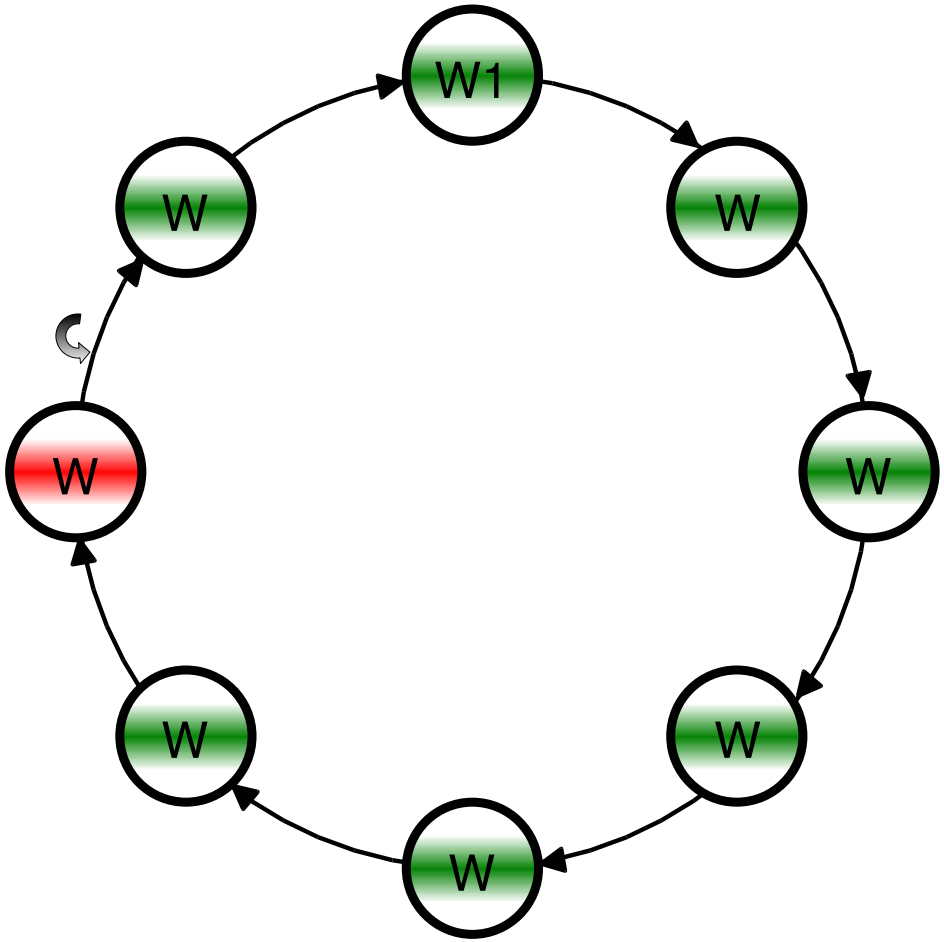
- Load imbalance
  $\longrightarrow$ idle machines
- Possible causes:
  - Heterogeneous tasks
  - Different worker performances



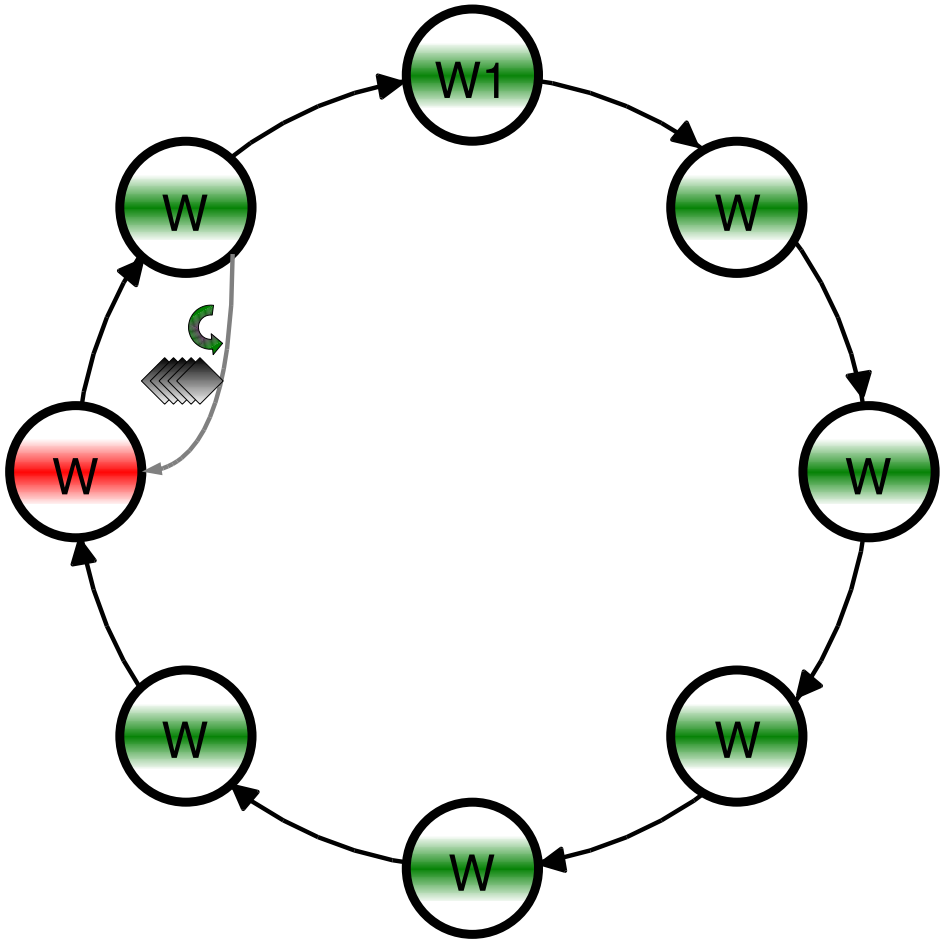
Global Functionality



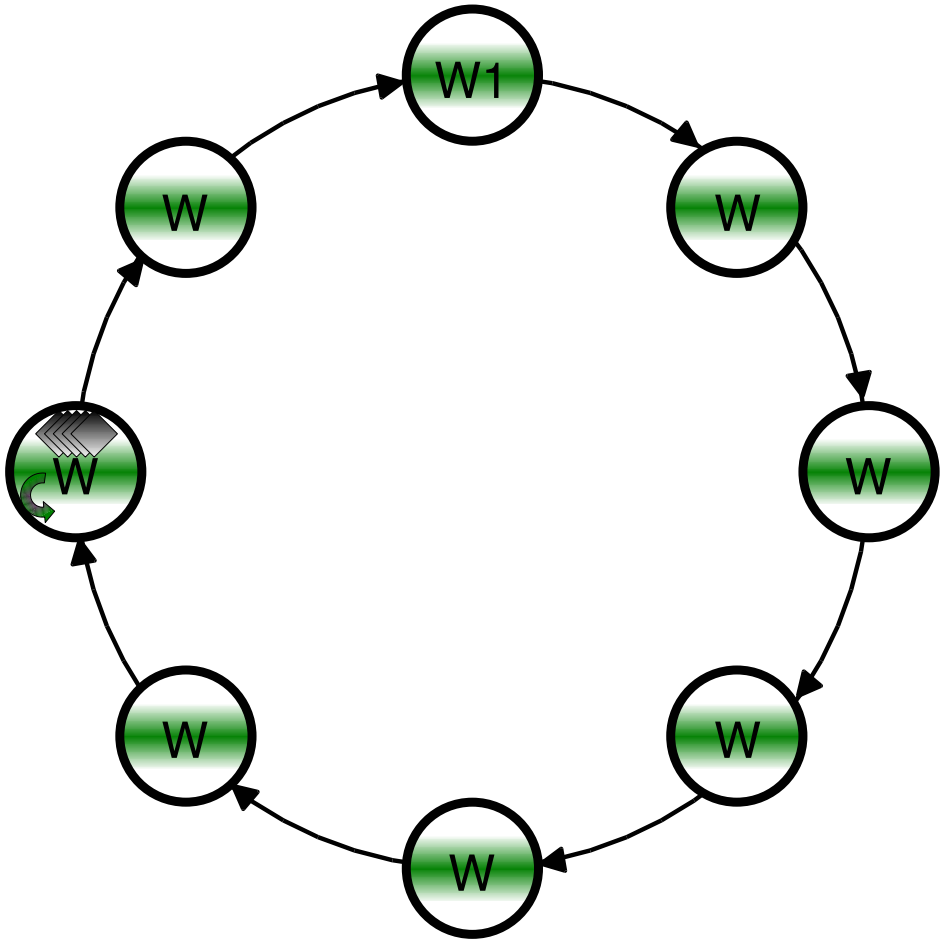
Global Functionality



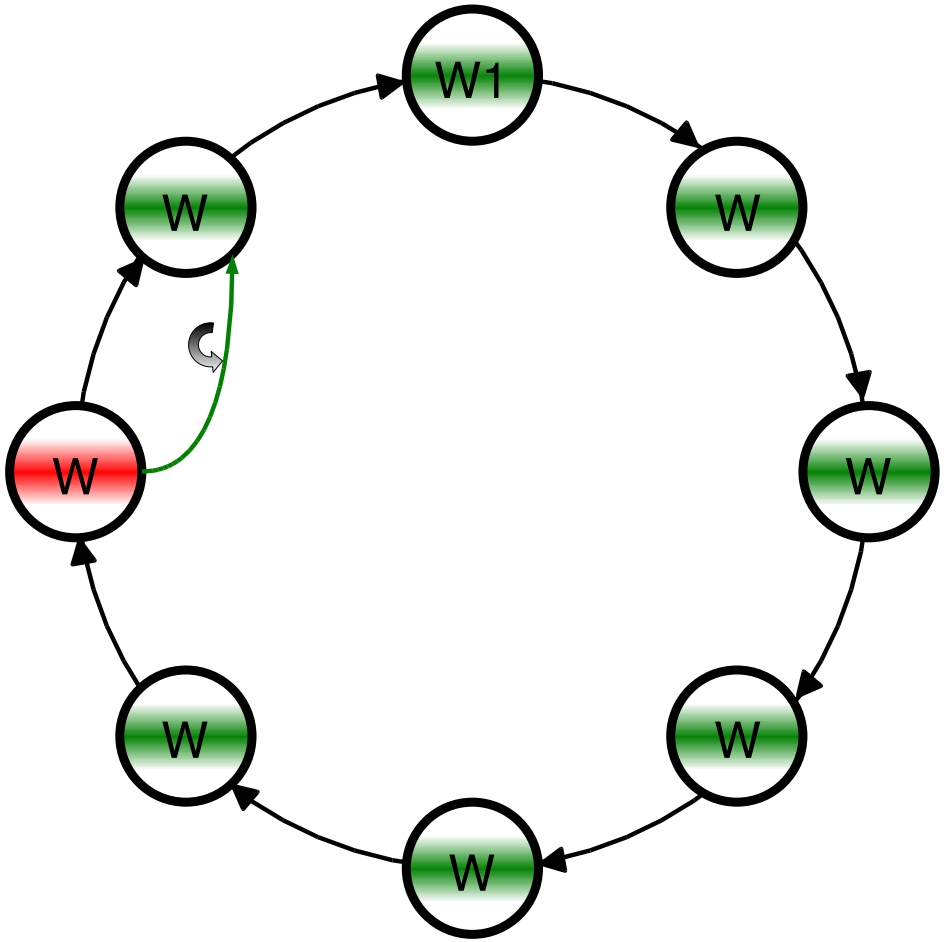
Global Functionality



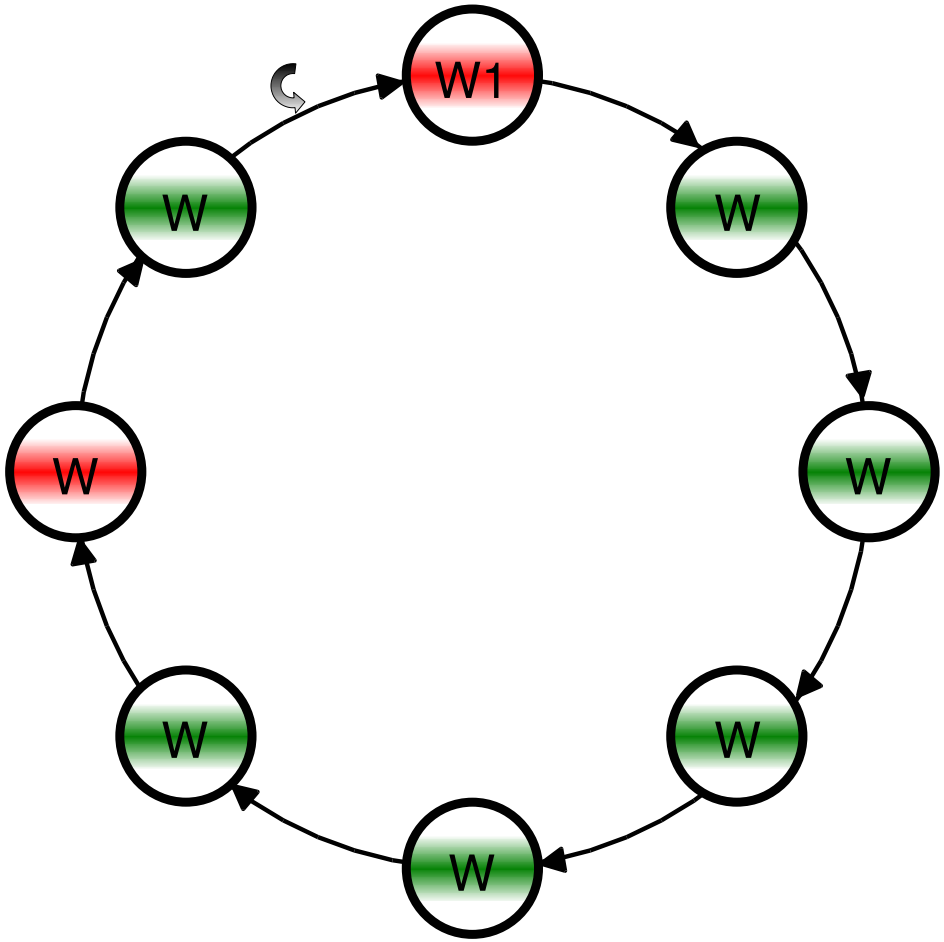
Global Functionality



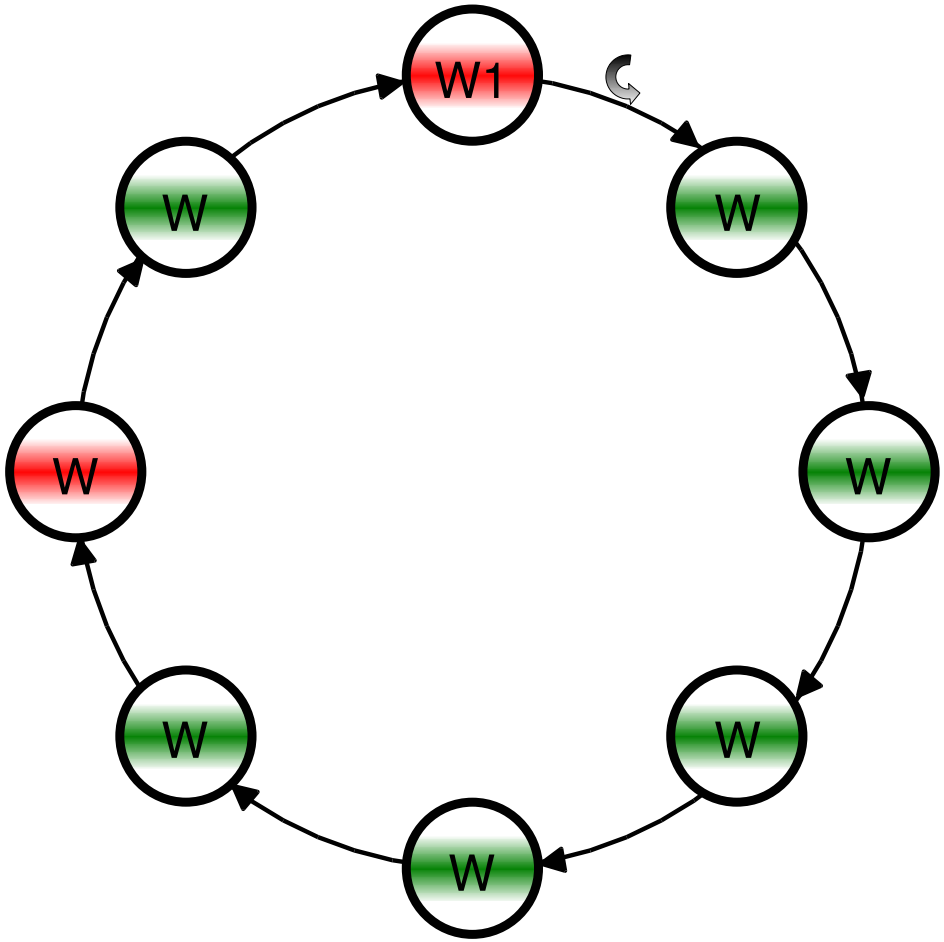
Global Functionality



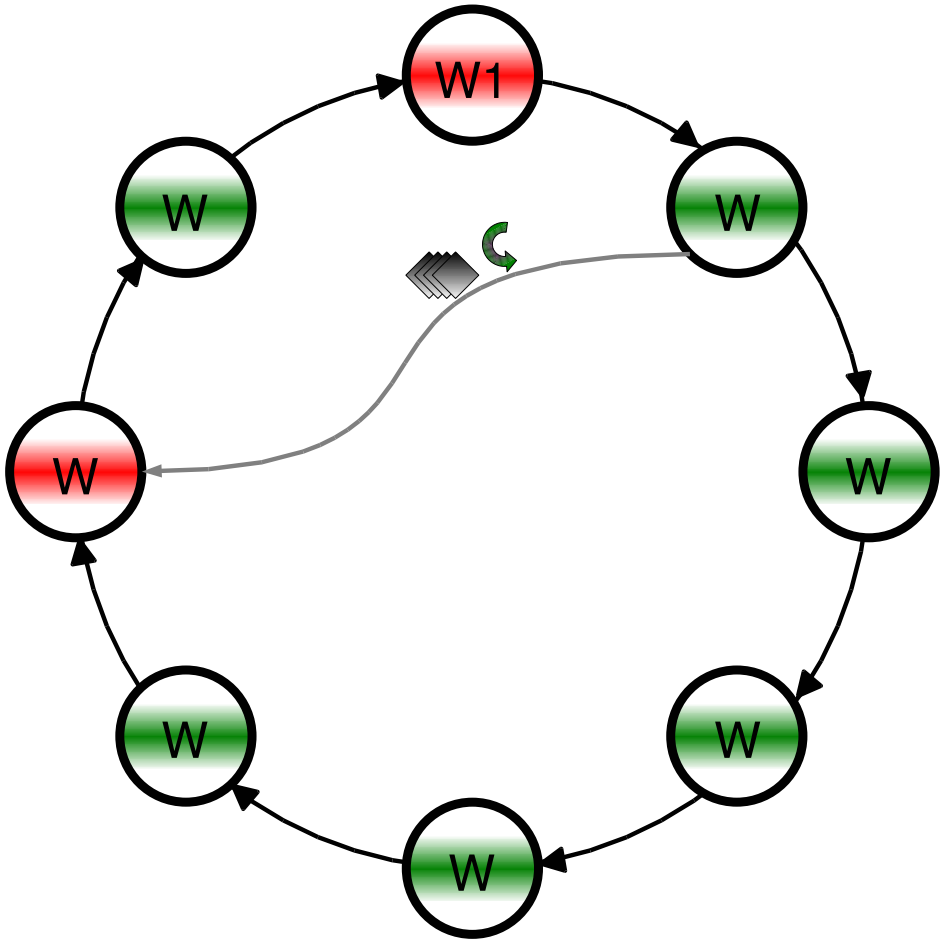
Global Functionality



Global Functionality



Global Functionality



Algorithm Classes

Algorithm class	**task pool size**	**post-processing**	**state**	**task pool structure**
parallel transformation (`map`)	fixed at start	no	no	queue
transformation and reduction (`map-reduce`)	fixed at start	yes	no	queue
backtracking (tree search)	dynamic	maybe	no	queue or stack
branch-and-bound (optimum search)	dynamic	yes	yes	priority queue

Skeleton Interface

```
mwRing :: (Trans t, Trans r, Trans s, NFData r') ⇒
  Int    →                                    -- no of processes
  -- task processing and result post processing
  ([(t,s)] → [(Maybe (r',s),[t])]) →  -- worker function
  ([Maybe (r',s)] → s → [r])       →  -- result transformation
  ([[r]] → [r])                    →  -- result merge function
  -- work pool transformation
  ([t] → [t] → s → [t])            →  -- attach function
  ([t] → s → ([t],[t]))            →  -- split  function
  ([t] → s → ([t],Maybe (t,s)))    →  -- detach function
  -- state comparison function
  (s → s → Bool)                   →  -- compare function
  -- initialisation
  s   → [t]                        →  -- initial state / tasks
  [r]                                 -- results
```

Simple specialized Interfaces

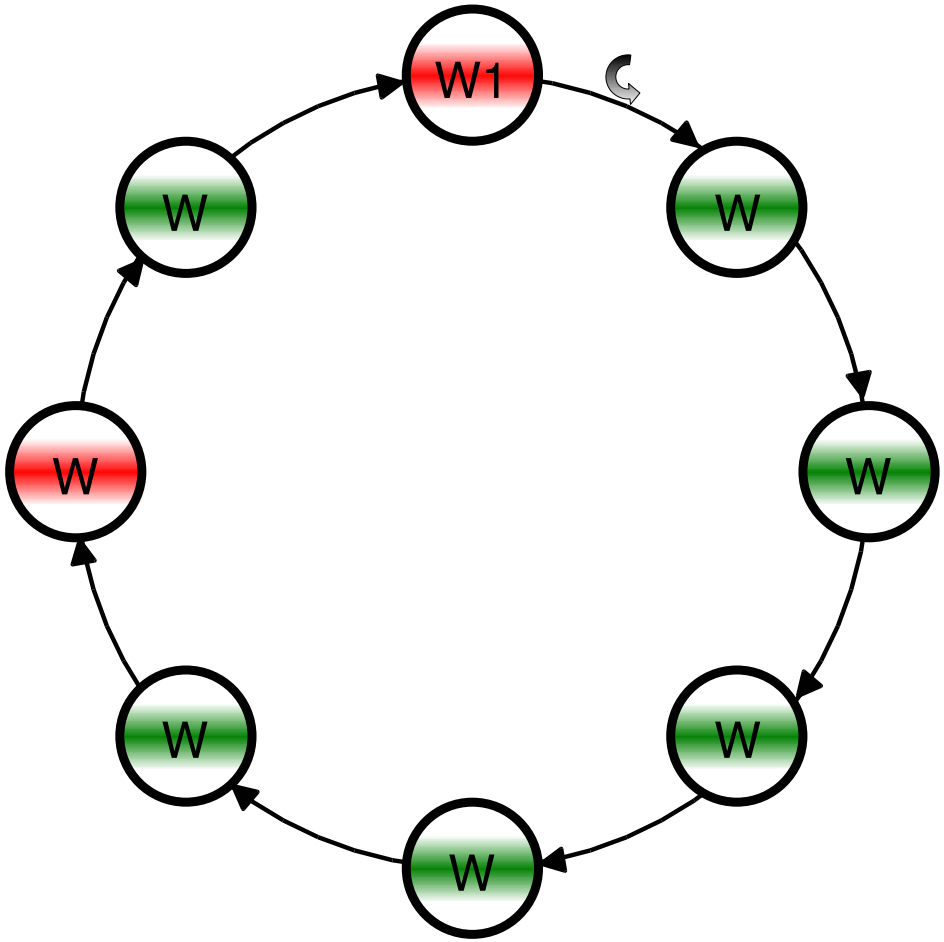
```
-- map-reduce skeleton
mwRingMapReduce :: (Trans t, Trans r) ⇒
  (t→r)       →                          -- worker function
  ([r]→[r])   →                          -- reduce function
  [t]  →  [r]                            -- tasks / results
```

Branch and Bound Interface

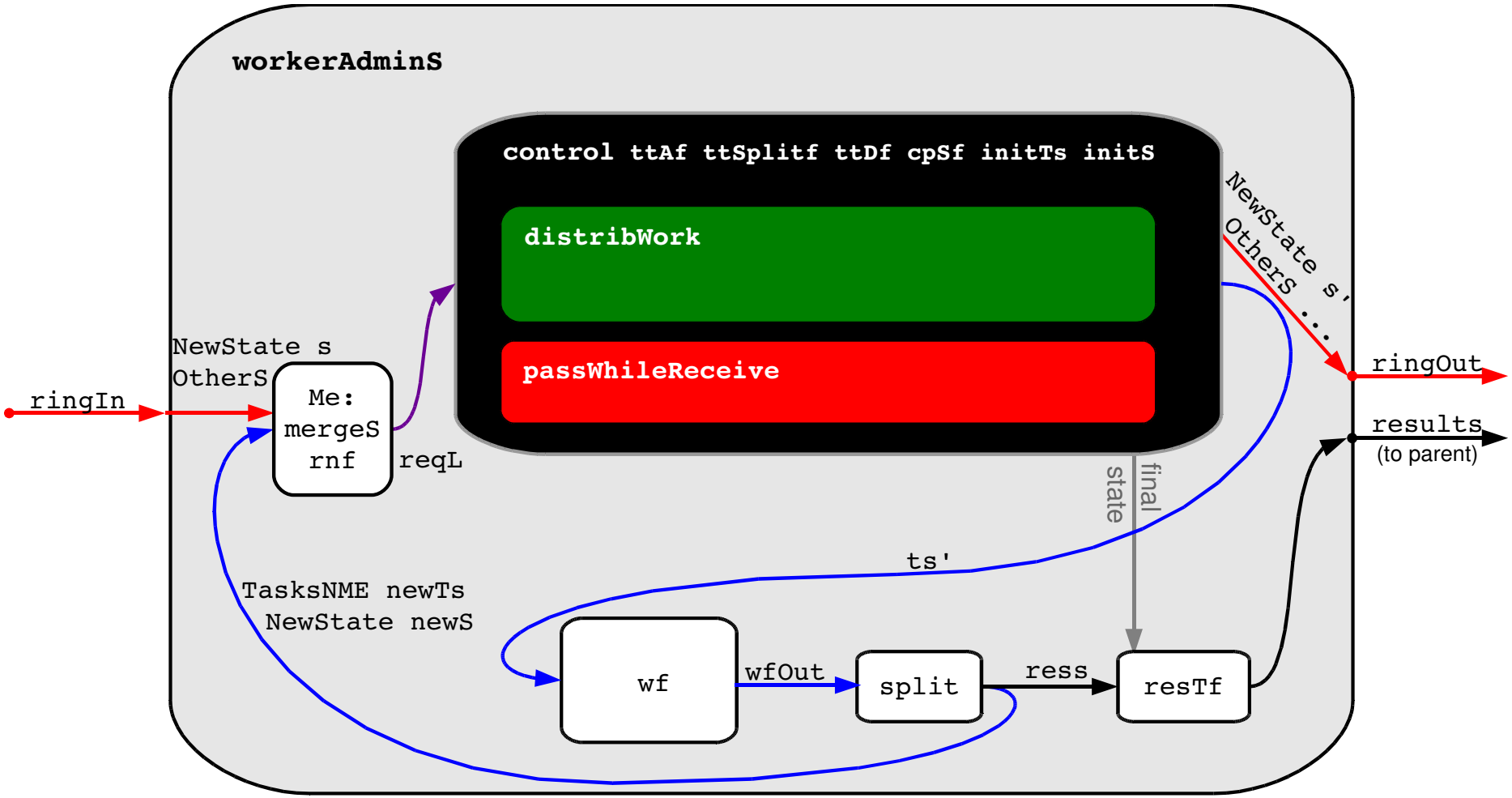
```
-- best first search skeleton
mwRingBNBBestFirst :: (Trans t, Trans b, Trans r) ⇒
  ([((t,b),b)] → [((Maybe (r,b)),[(t,b)])]) -- worker function
  → (b → b → Bool)                          -- state comparison
  → b → [(t,b)]                             -- initial state/tasks
  → [(r,b)]                                 -- result

-- interface for branch and bound worker functions
bNB_WF :: (NFData h, Trans b, Trans t) ⇒
  (b → b → Bool)                            -- state comparison
  → (t→(t,b))                               -- bound functon
  → (t→(Maybe (r,b),[t]))                   -- branch function
  → [((t,b),b)]                             -- tasklist
  → [((Maybe (r,b)),[(t,b)])]               -- results/new tasks
```

Global Functionality



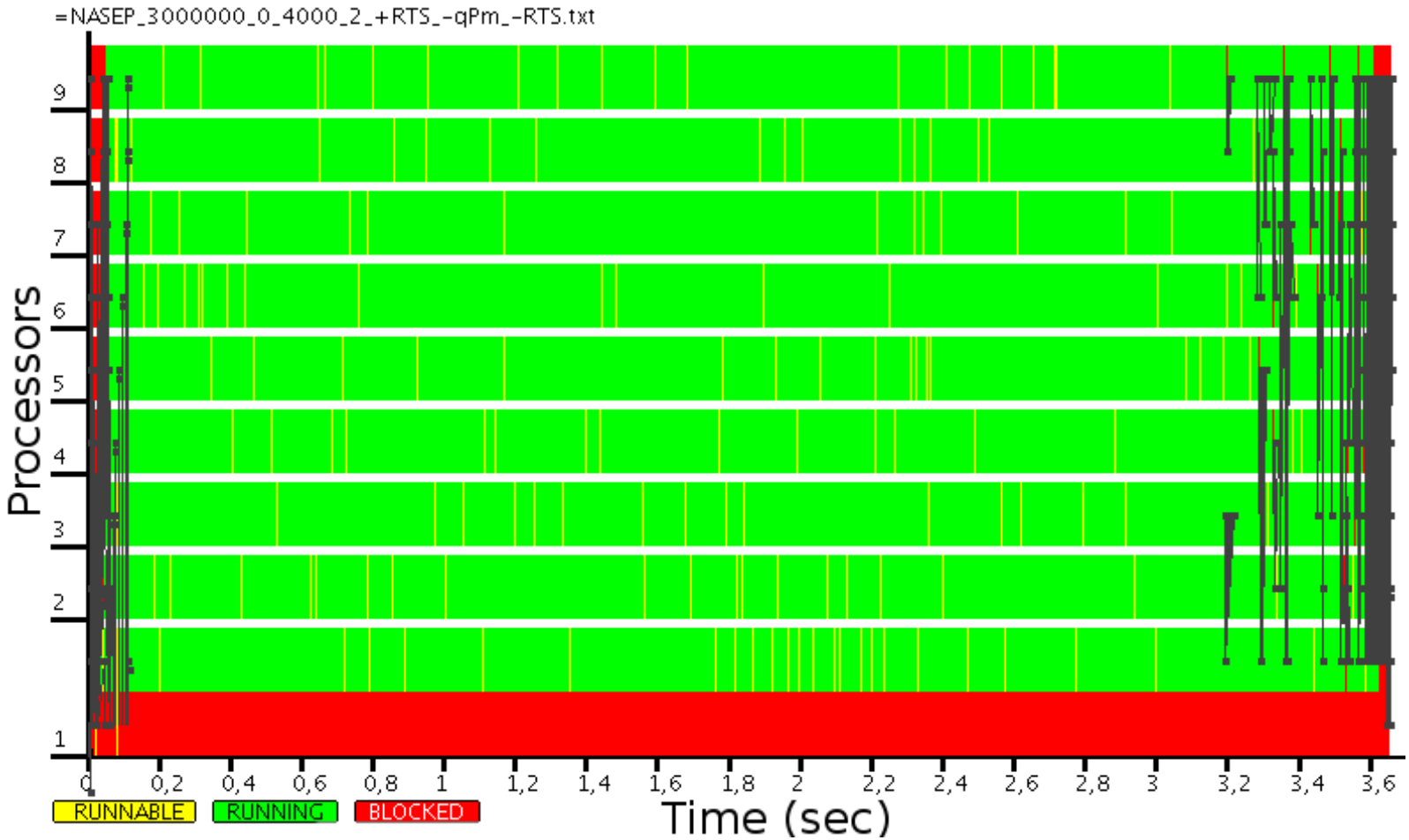
Worker Architecture



NAS EP Benchmark

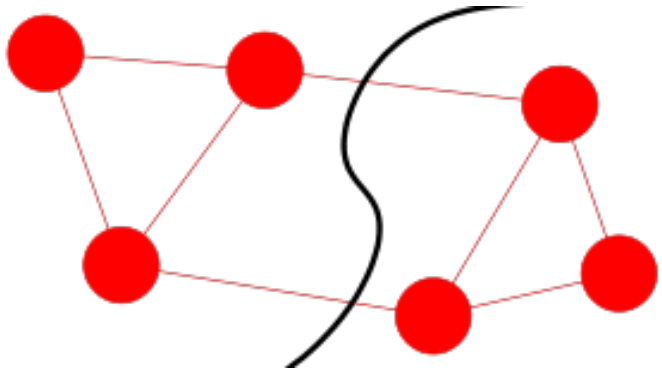
The "embarassingly parallel" NAS benchmark
$\longrightarrow$ typical **map-reduce** problem

Runtime trace (9 machines, fast Ethernet):



Graph-Partitioning-Problem

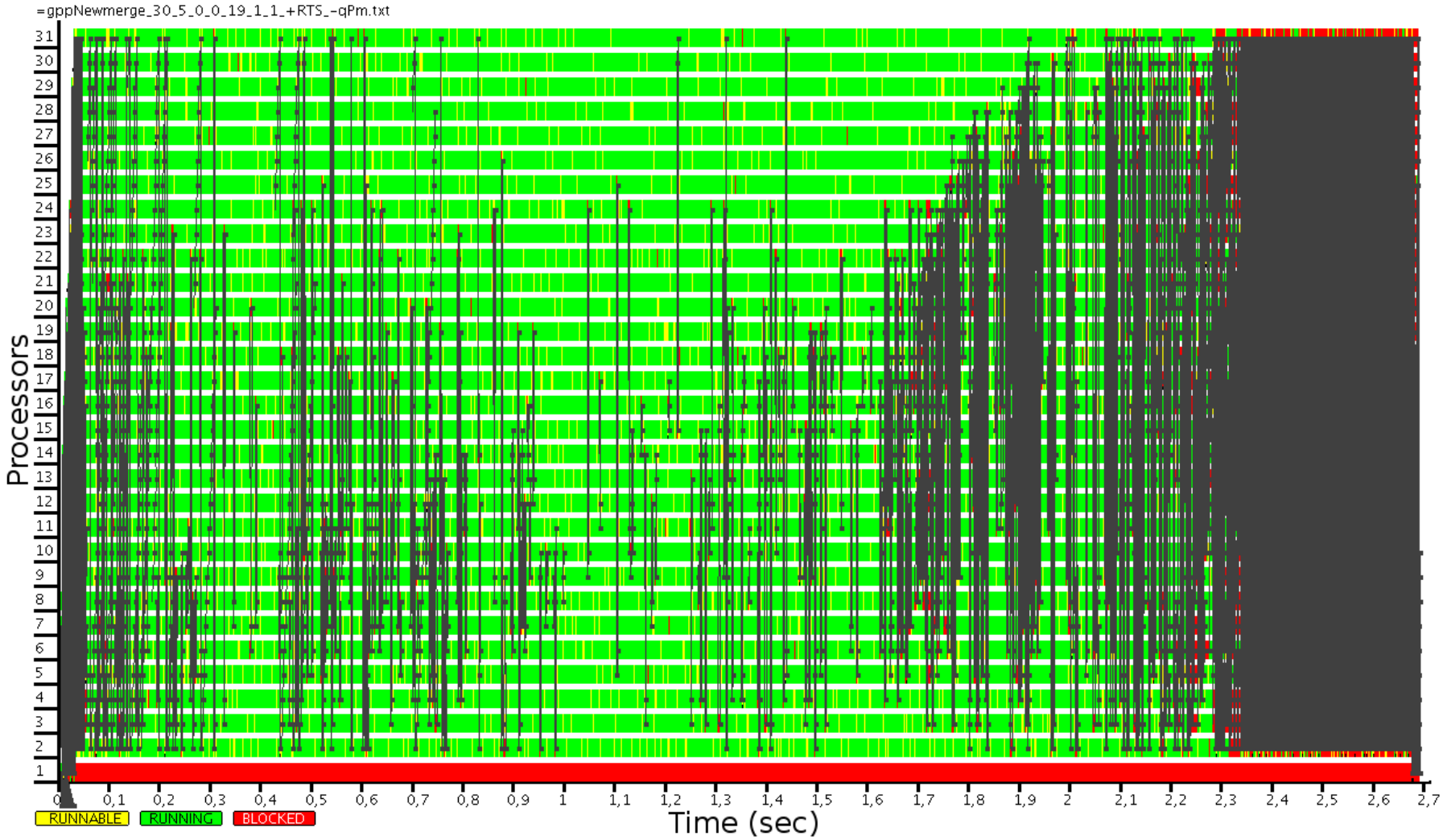
Goal:
Partition G into two subgraphs of equal size with min truncation cost



... a branch and bound problem

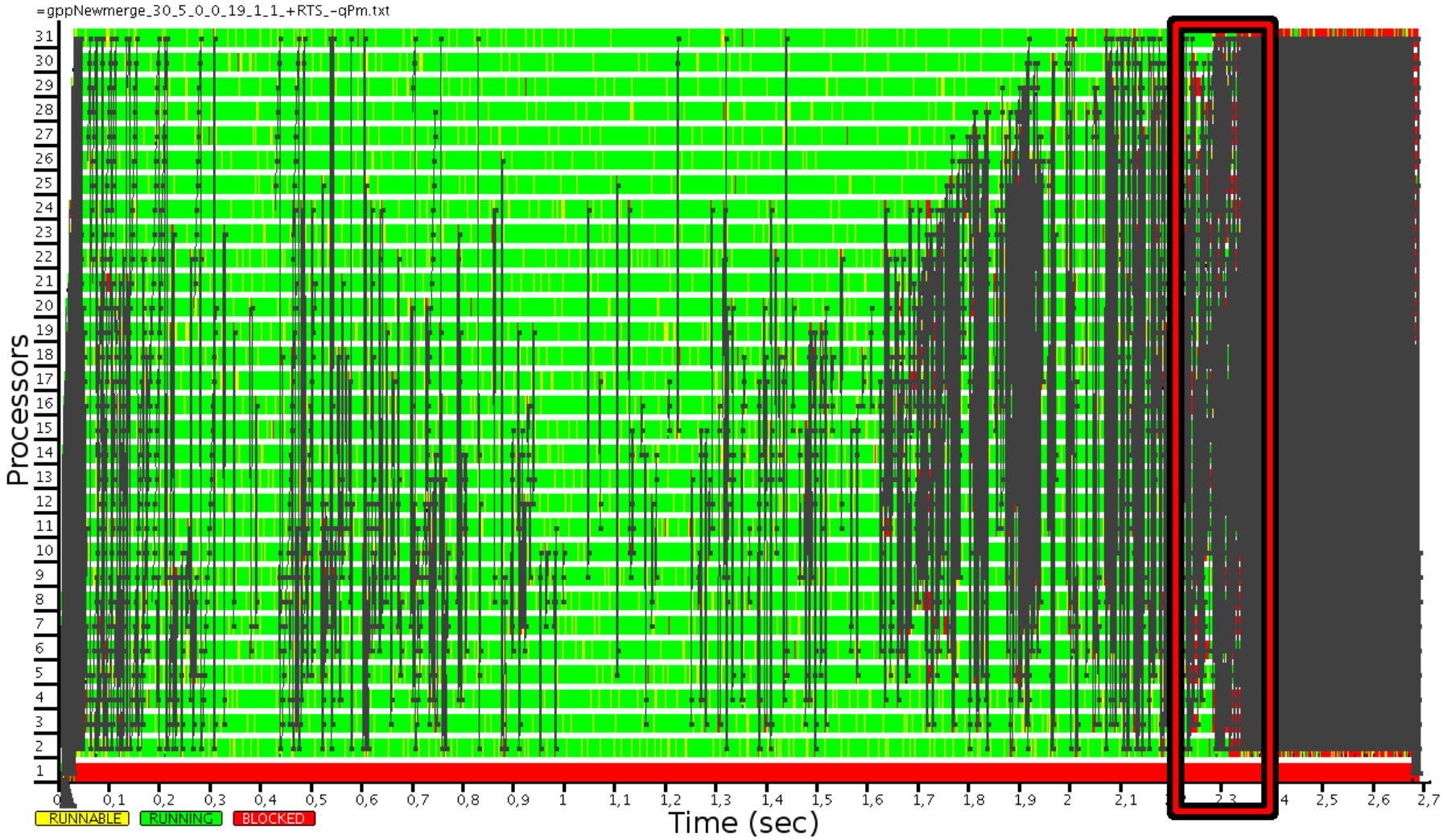
Graph-Partitioning-Problem Trace

Problem size: 30 nodes on 31 PEs (P4@3GHz), beowulf cluster

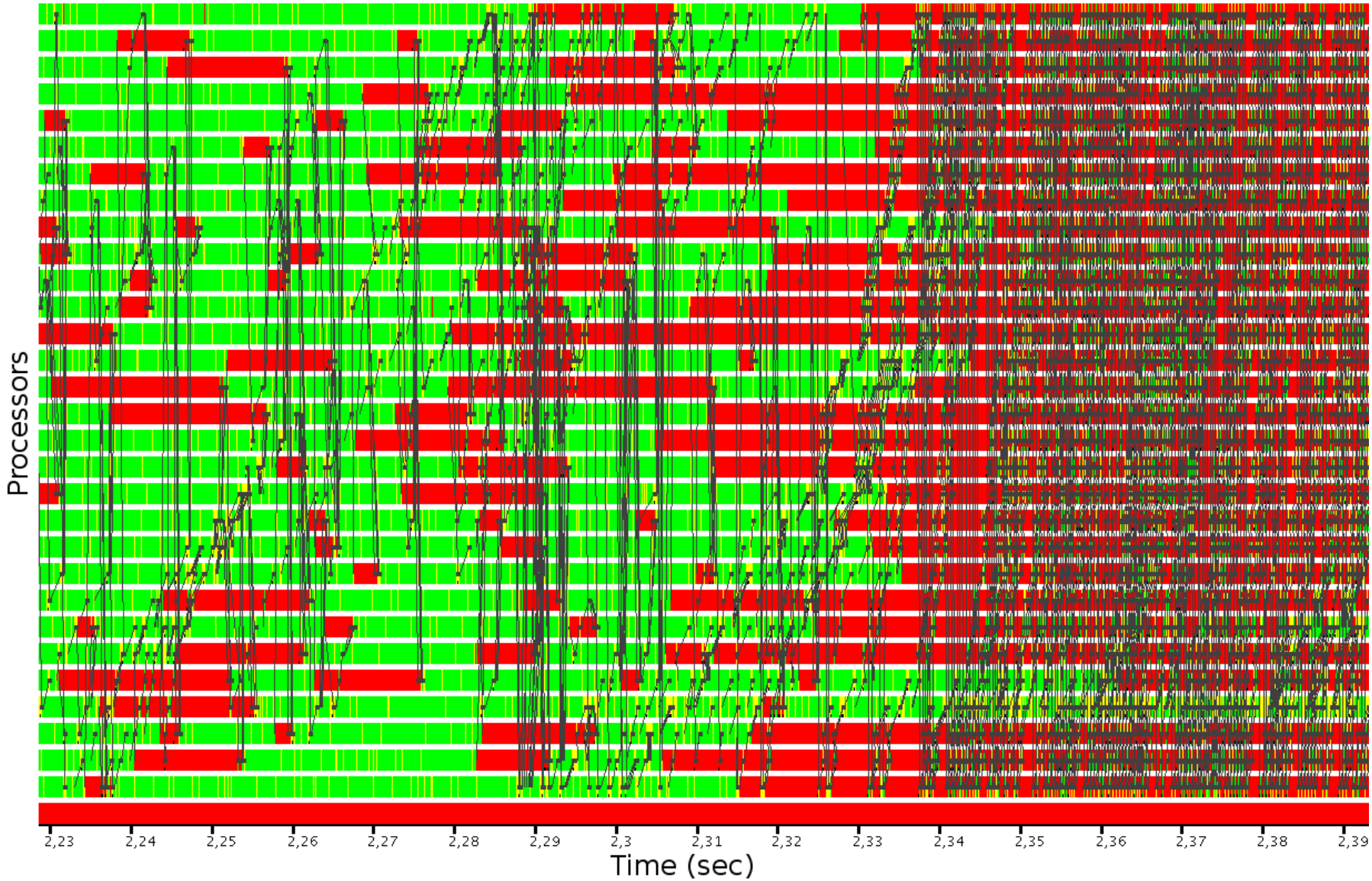


Graph-Partitioning-Problem Trace

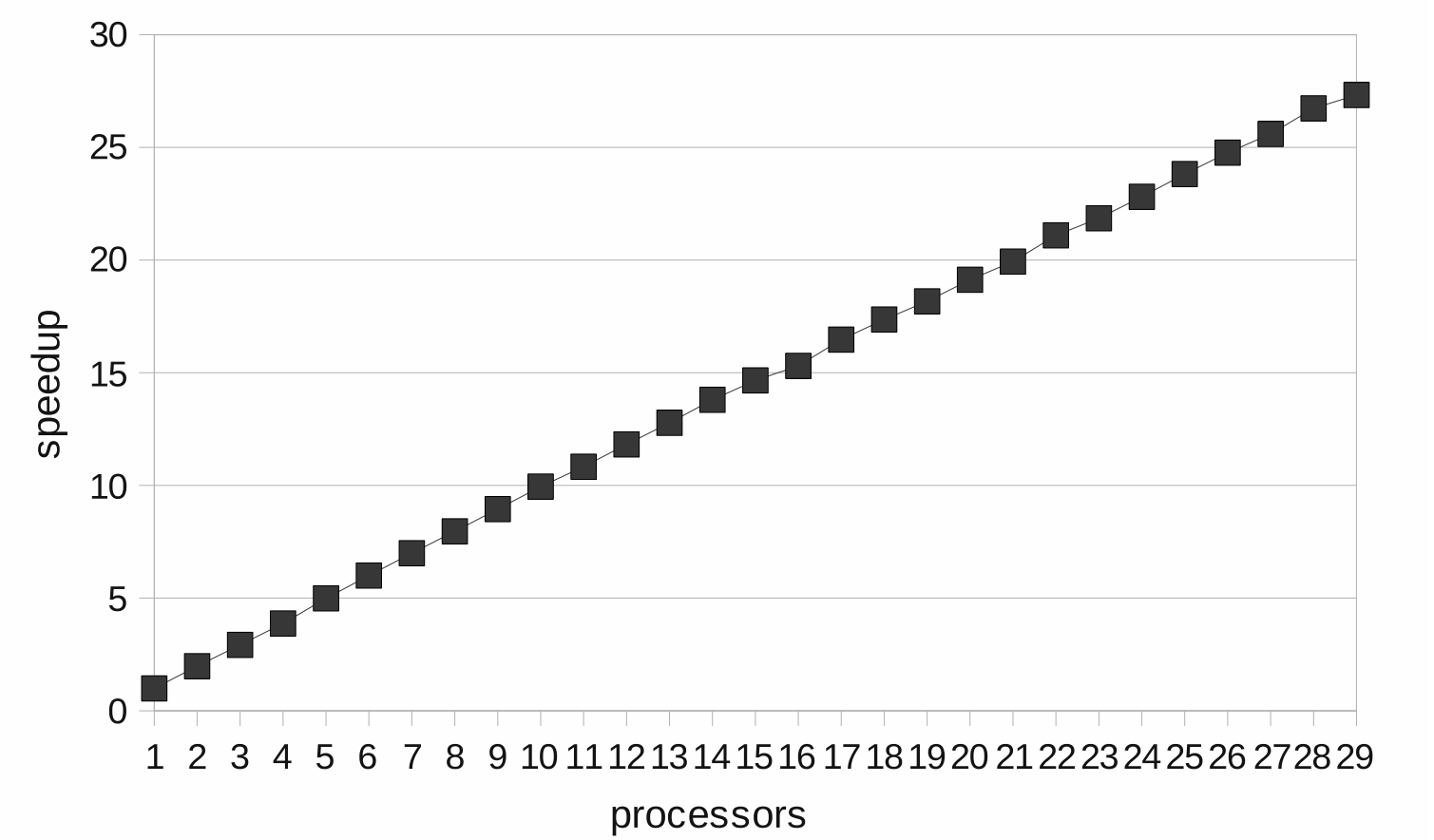
Problem size: 30 nodes on 31 PEs (P4@3GHz), beowulf cluster



Graph-Partitioning-Problem Trace ZOOM



Graph-Partitioning-Problem Speedup



Conclusions

- Highly flexible skeleton
- Simple interfaces
- Declarative implementation
- Good load balancing
- Well scaling

Conclusions

- Highly flexible skeleton
- Simple interfaces
- Declarative implementation
- Good load balancing
- Well scaling

Conclusions

- Highly flexible skeleton
- Simple interfaces
- Declarative implementation
- Good load balancing
- Well scaling