Scalable Parallel Numerical Constraint Solver using Global Load Balancing

June 14 @X10’15

Daisuke Ishii Tokyo Institute of Technology

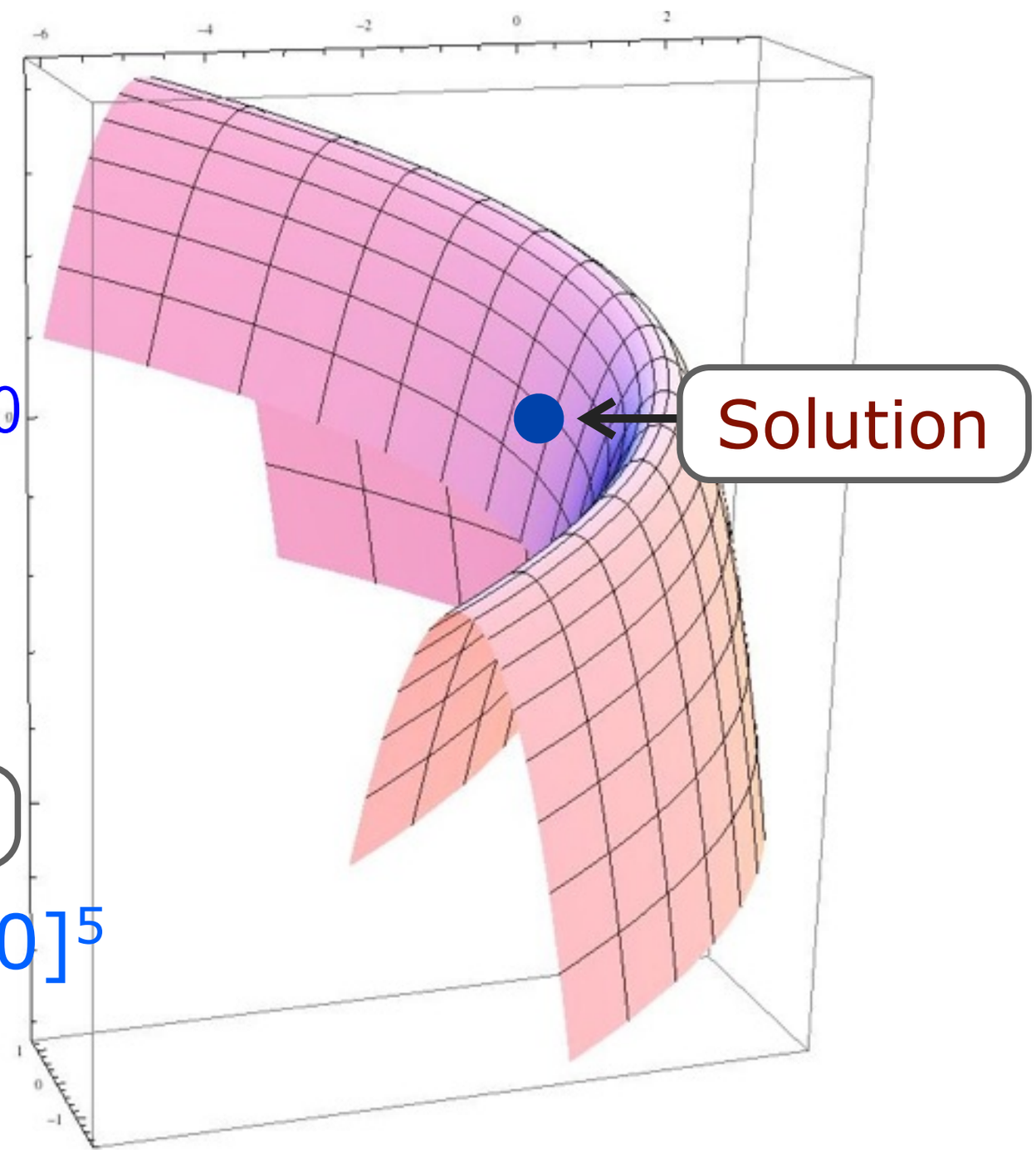
Kazuki Yoshizoe Tokyo University

Toyotaro Suzumura IBM Research, UCD, JST

Example of Numerical Constraint Satisfaction Problem

Constraint

$$\begin{pmatrix} x_1 \\ x_4 \\ (3-2x_2)x_2 - x_1 - 2x_3 + 1 \\ (3-2x_3)x_3 - x_2 - 2x_4 + 1 \\ (3-2x_4)x_4 - x_3 - 2x_5 + 1 \end{pmatrix} = 0$$



Variables

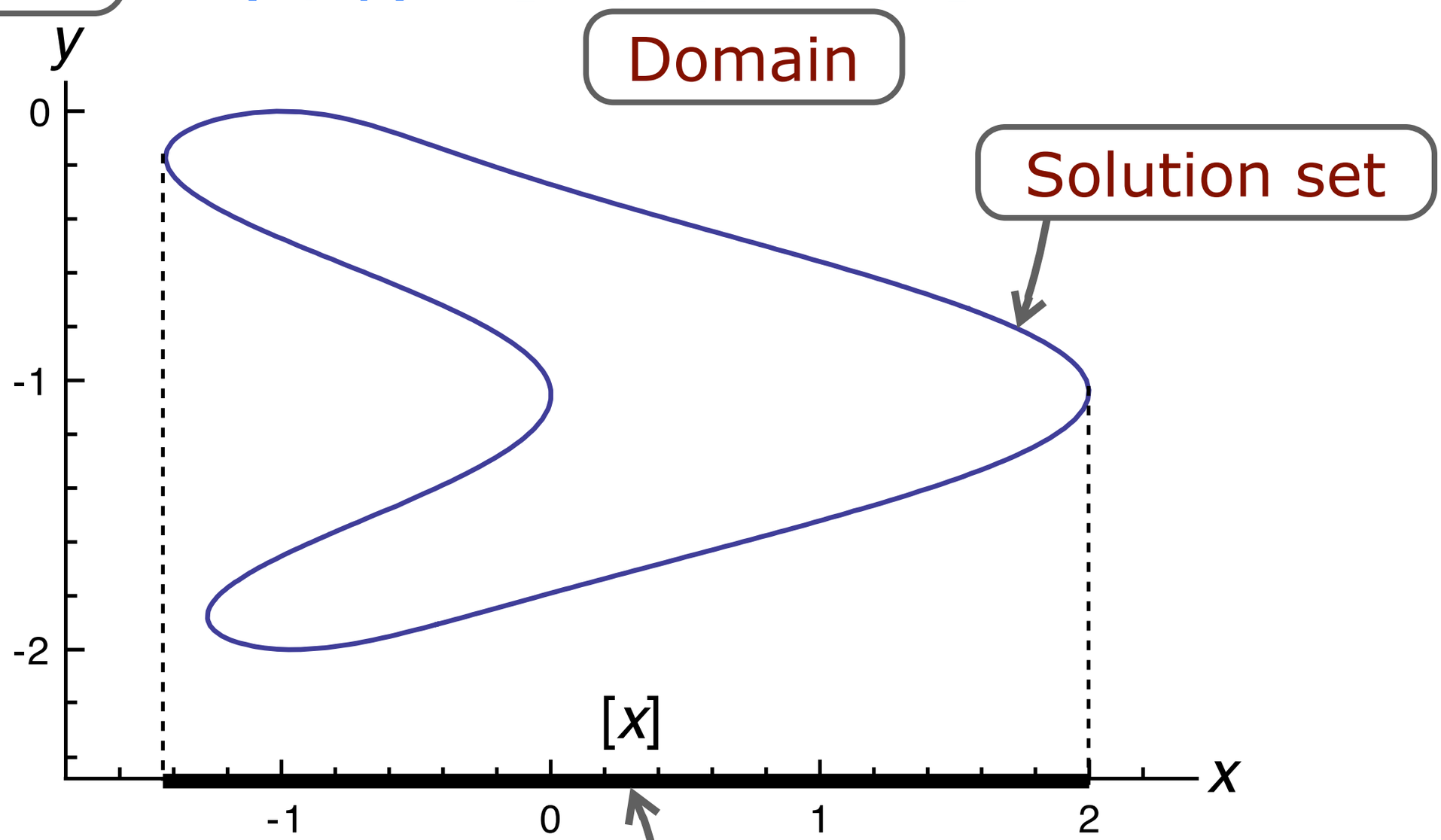
Domain

$(x_1,x_2,x_3,x_4,x_5) \in [-100,100]^5$

Example of Numerical Constraint Satisfaction Problem

Constraint $\exists y\ (x + \cos 3y)^2 + (y+1)^2 - 1 = 0$

Variables $(x, y) \in [-2,3] \times [-3,1]$



Projected solution set onto x

Example of Numerical Constraint Satisfaction Problem

Constraint

$$\exists y_1, y_2 \begin{pmatrix} x_1 - y_1 - 2\cos y_2 \\ x_2 - y_1 - 2\sin y_2 \end{pmatrix} = 0$$

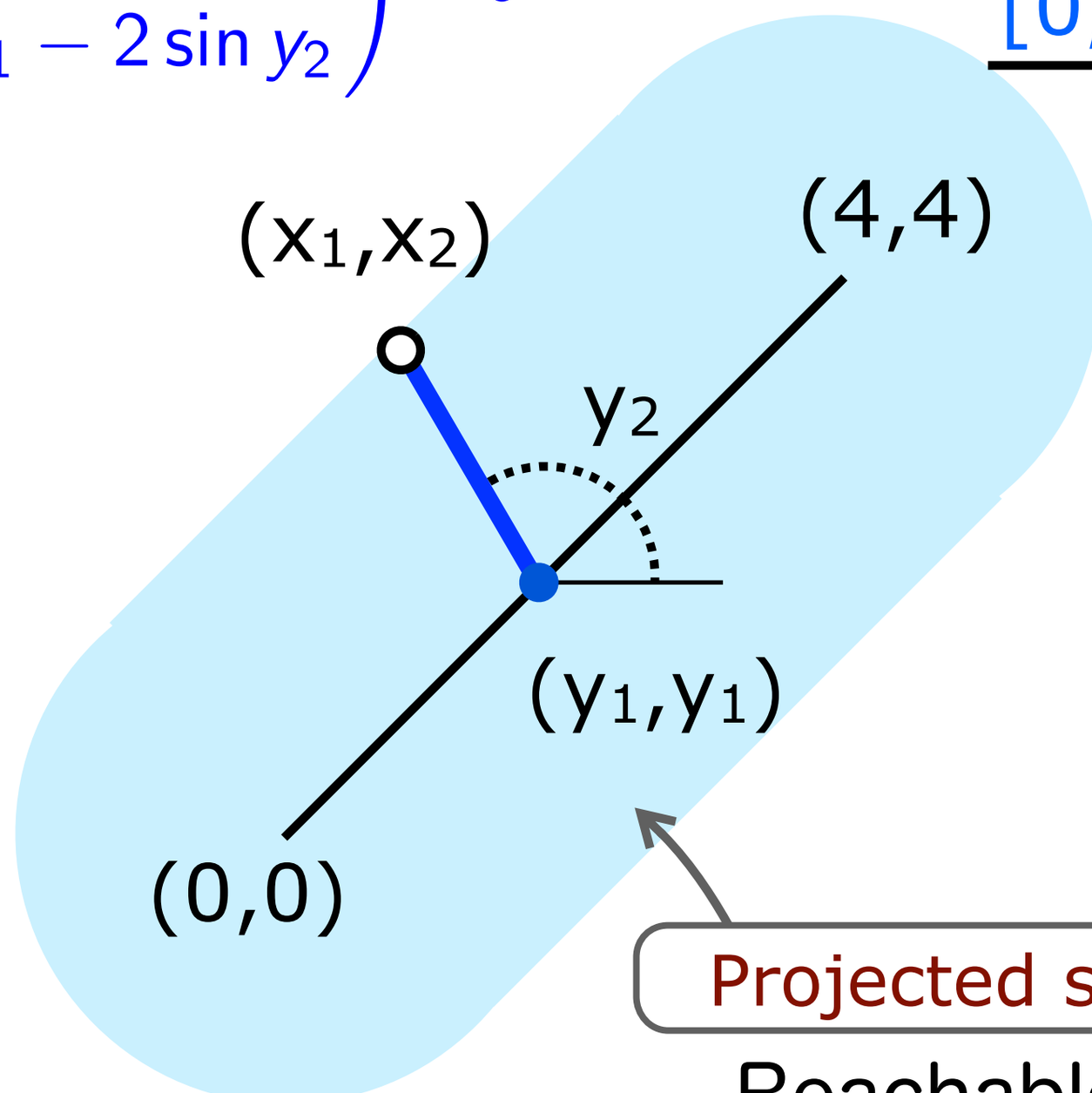
Variables

(x_1, x_2, y_1, y_2)

Domain

$\in [-10,10]^2 \times [0,4] \times [-\pi, \pi]$

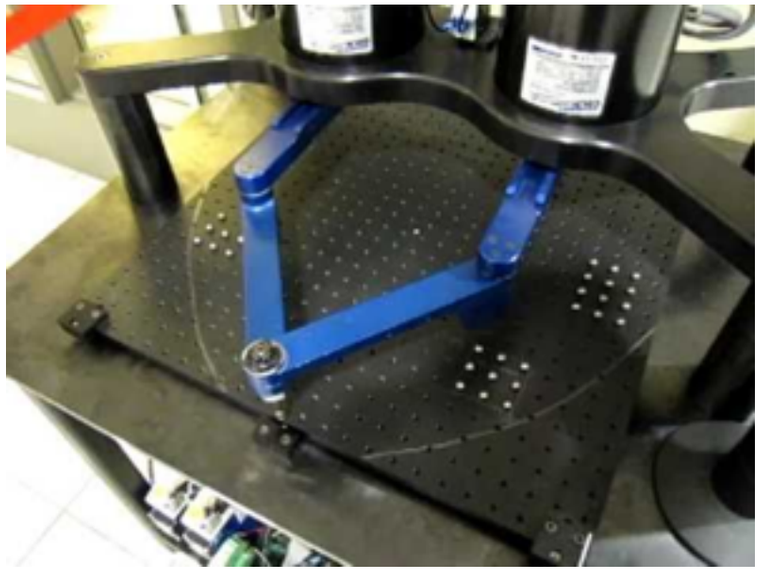
Possible set of inputs



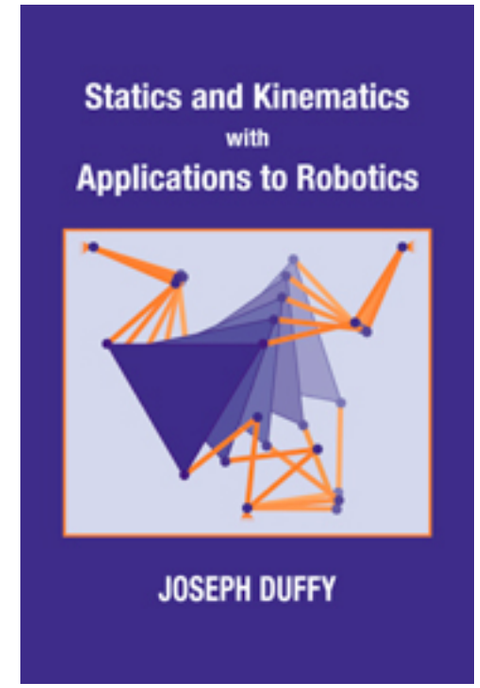
Projected solution set

Reachable region

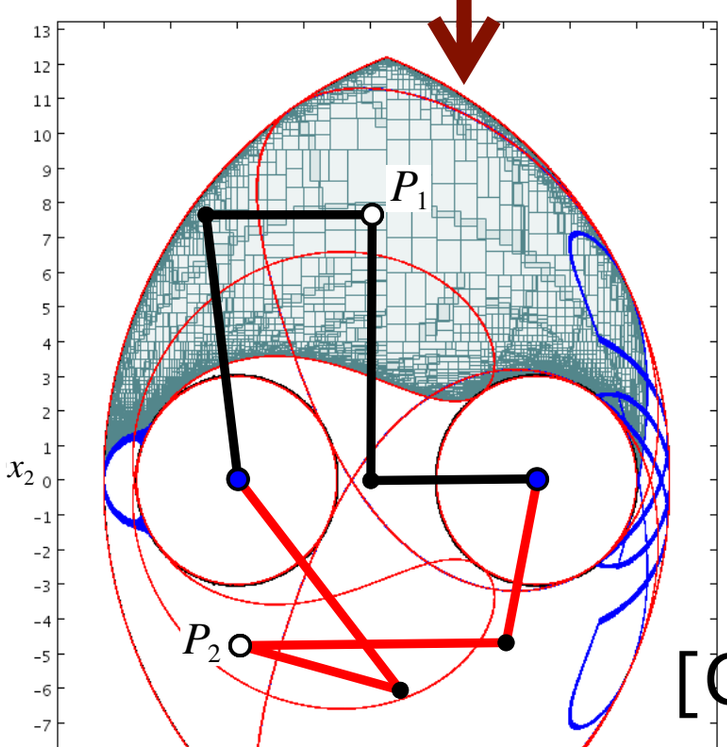
Application: Workspace Analysis of Parallel Robots



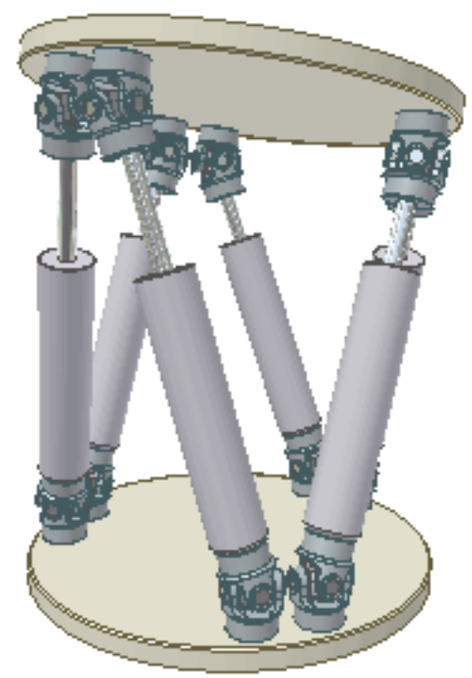
DexTAR [L. Campos+ 2010]



NCSP



[Caro+ 14]



http://en.wikipedia.org/wiki/Stewart_platform

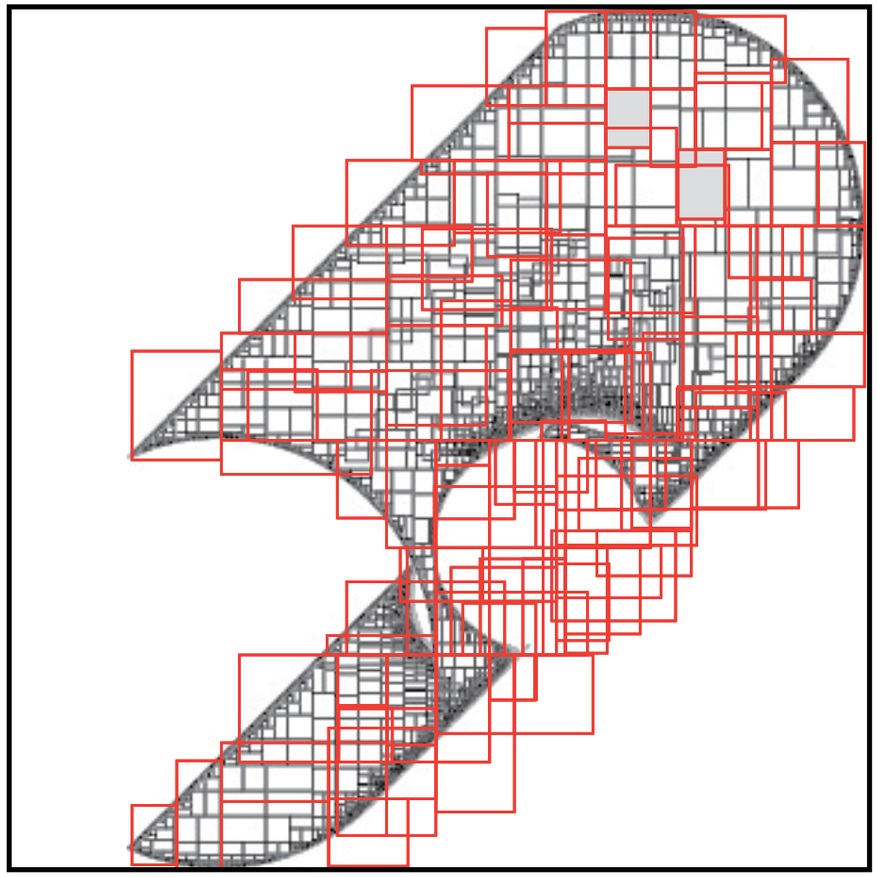
Numerical Constraint Satisfaction Problems (NCSPs)

- Framework for describing/analyzing problems in the domain of **reals**
  - Application: systems control, robotics, economics, biology, etc.

- **NCSP solvers** based on **interval analysis** have been developed
  - Computation of **boxes** (i.e. interval vectors) that enclose the solution set using the **Branch-and-Prune algorithm**
  - E.g. Realpaver [Granvilliers+ 06]

Parallelization of NCSP Solving Process

- **Exponential computational complexity limits the number of tractable instances**



- We propose a **scalable parallel NCSP solver** using **search-space splitting** and **global load balancing**

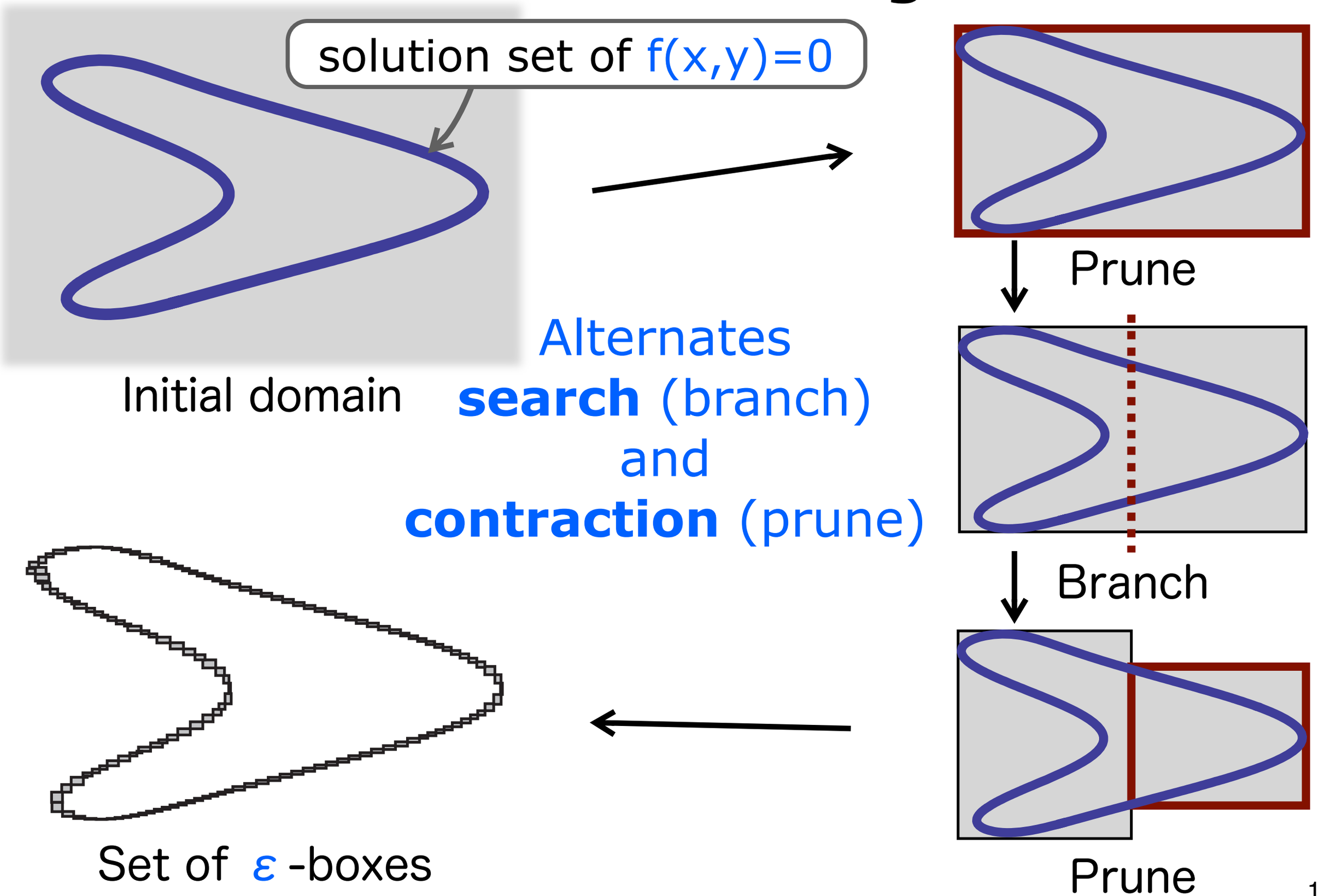
Parallelization of (Numerical) CSP Solvers

- **Difficulty of parallelizing CSP solvers lies in the balanced search-space splitting**
  - Search tree easily becomes unbalanced, and it is difficult to predict the appropriate splitting
  - Dynamic load balancing (work stealing) scheme
  - Parallel CSP solvers (w. central master process) are limited to scaling up to a few hundred cores
    [Jaffar+ 04], [Bourdeaux+ 06], [Xie 10], [Bergman+ 14], etc.
- **Global load balancing** [Saraswat+ 11], [Zhang+ 14]
  - Scalable scheme with decentralized work stealing and termination detection
  - For generic irregular parallel computation
- **Scalable parallelization of the branch-and-prune algorithm that handles the continuous domain** (Cf. branch-and-bound algorithm)

Talk Outline

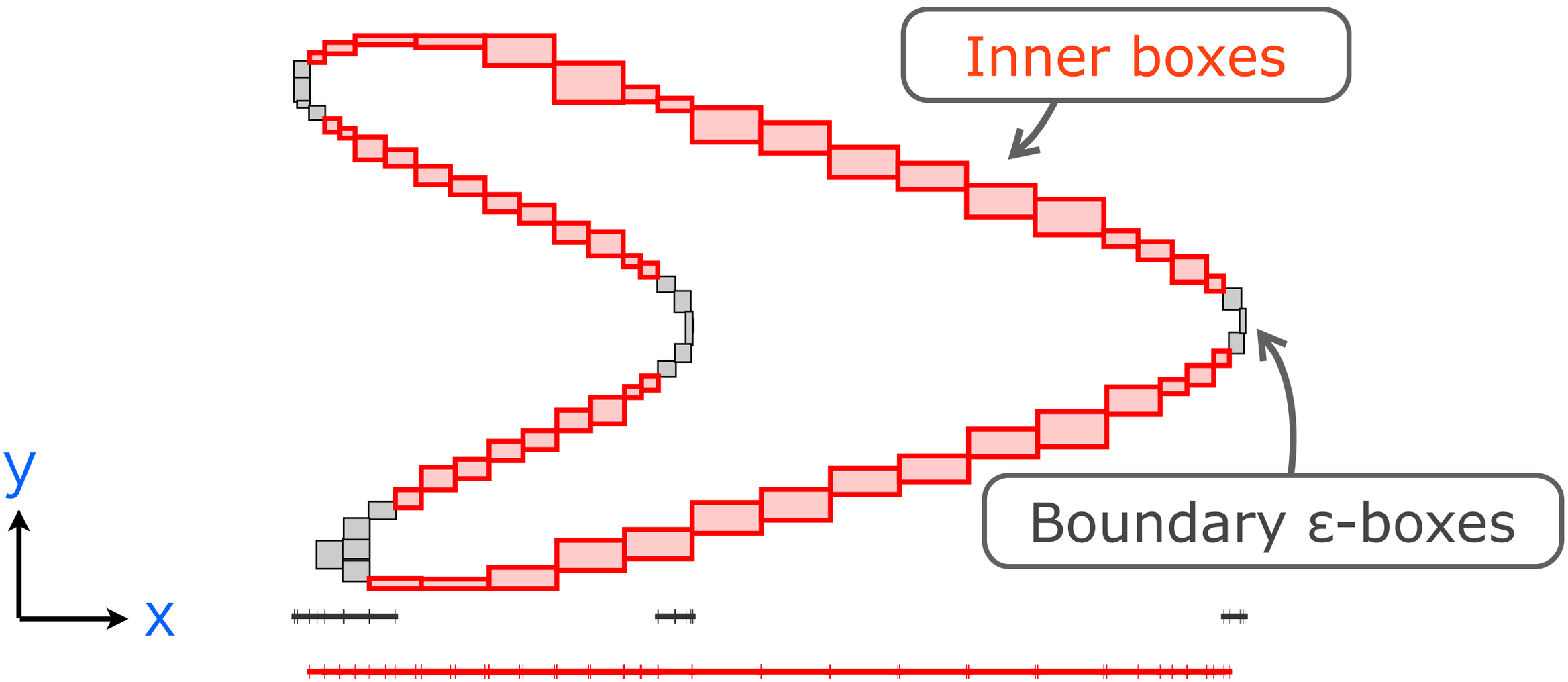
1. Introduction
2. **Branch-and-Prune algorithm and parallel method**
3. Experimental results

Branch-and-Prune Algorithm



Detection of Inner Boxes [Ishii+ 12]

- Prune procedure can verify that a box is inside the x-projection of the solution set
- Inner boxes need not to be searched



Outer and inner approximation of the projection

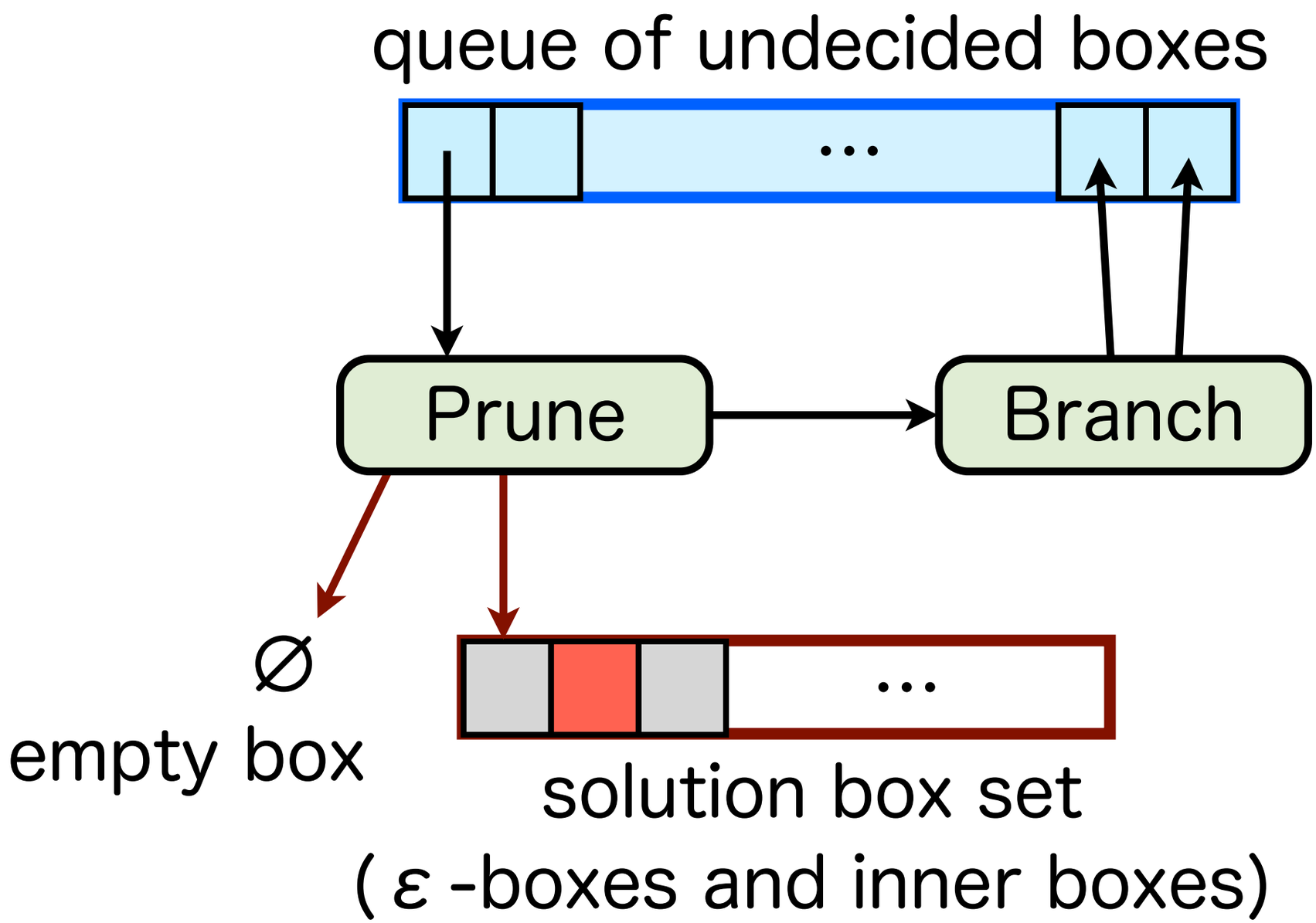
Parallel Branch-and-Prune

- **We apply a Global Load Balancing scheme that runs a solver worker on each of the available CPU cores**

- Each worker homogeneously interleaves the following procedures
  1. Branch and prune search
  2. Distribution/load balancing of search space
  3. Termination detection

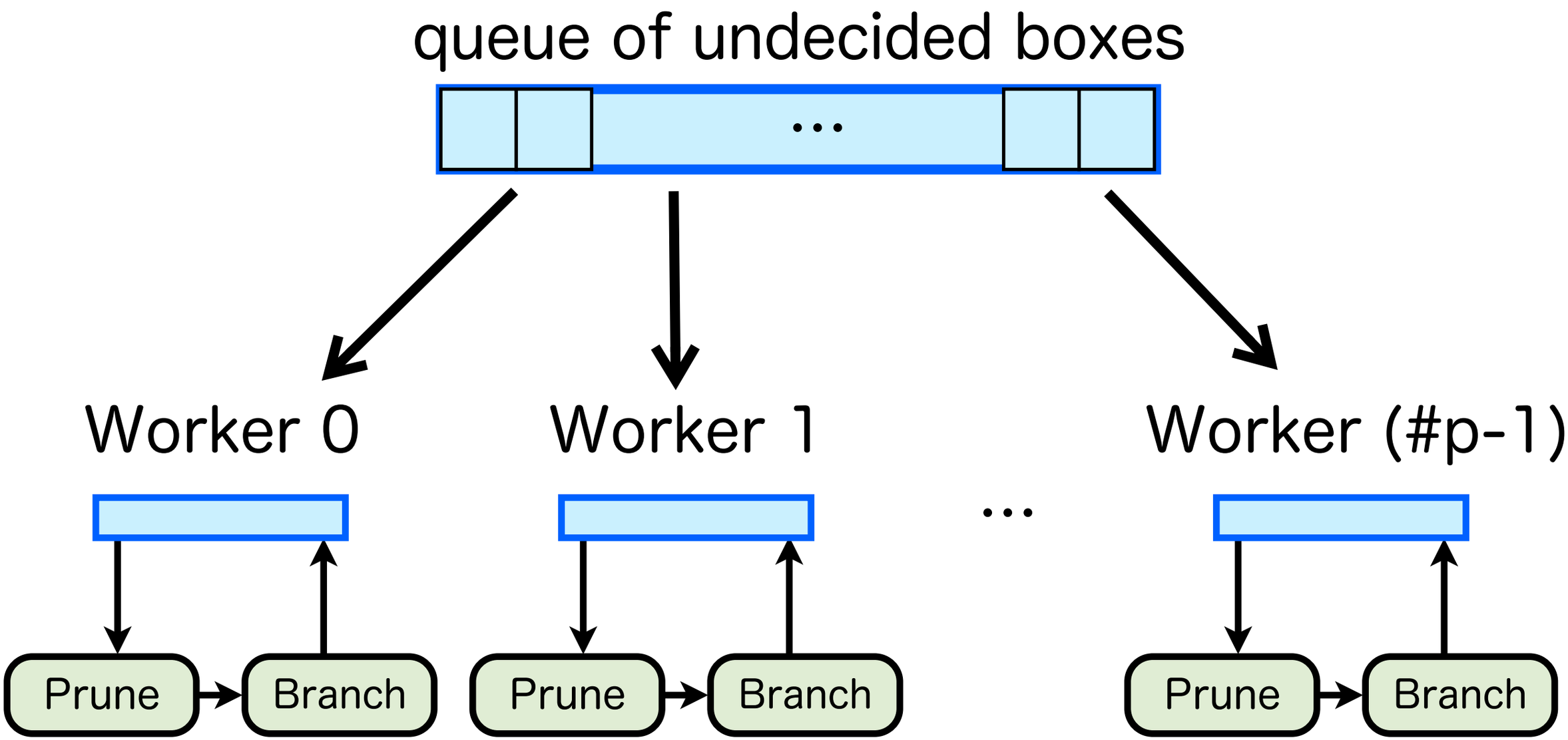
Branch-and-Prune Algorithm

- A step computation



Parallel Branch-and-Prune

- Search-space splitting

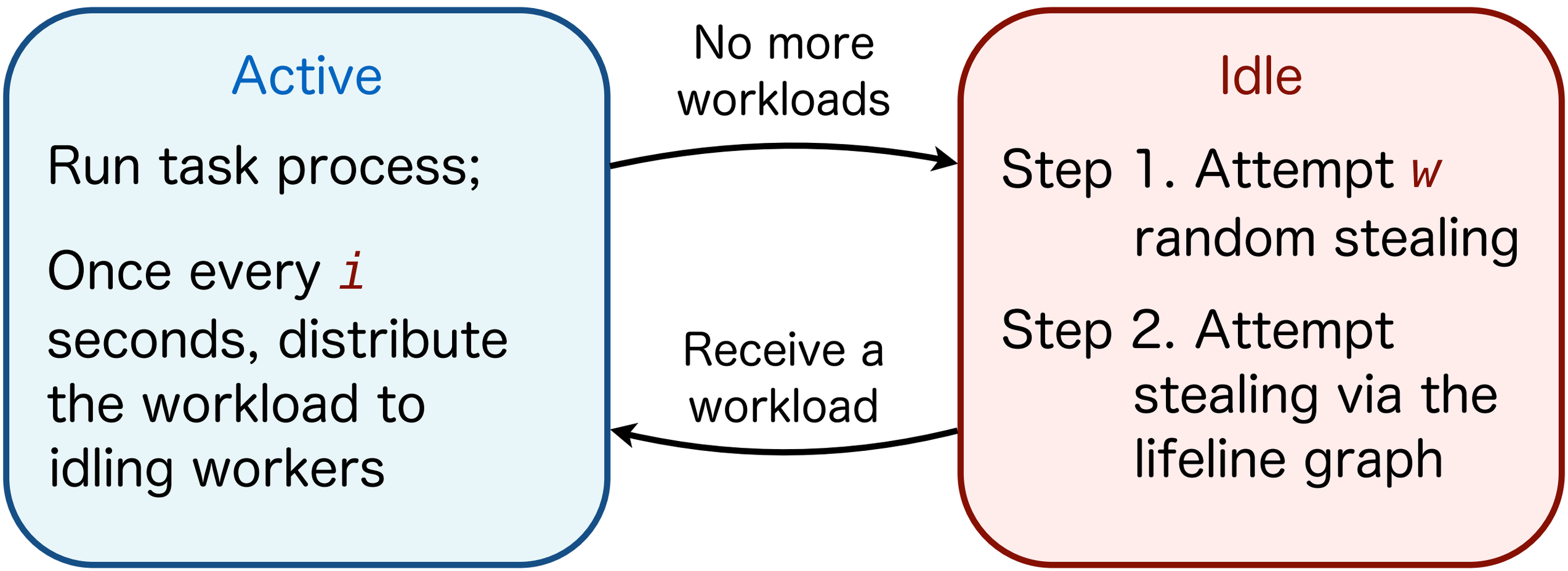


Global Load Balancing (GLB) Scheme

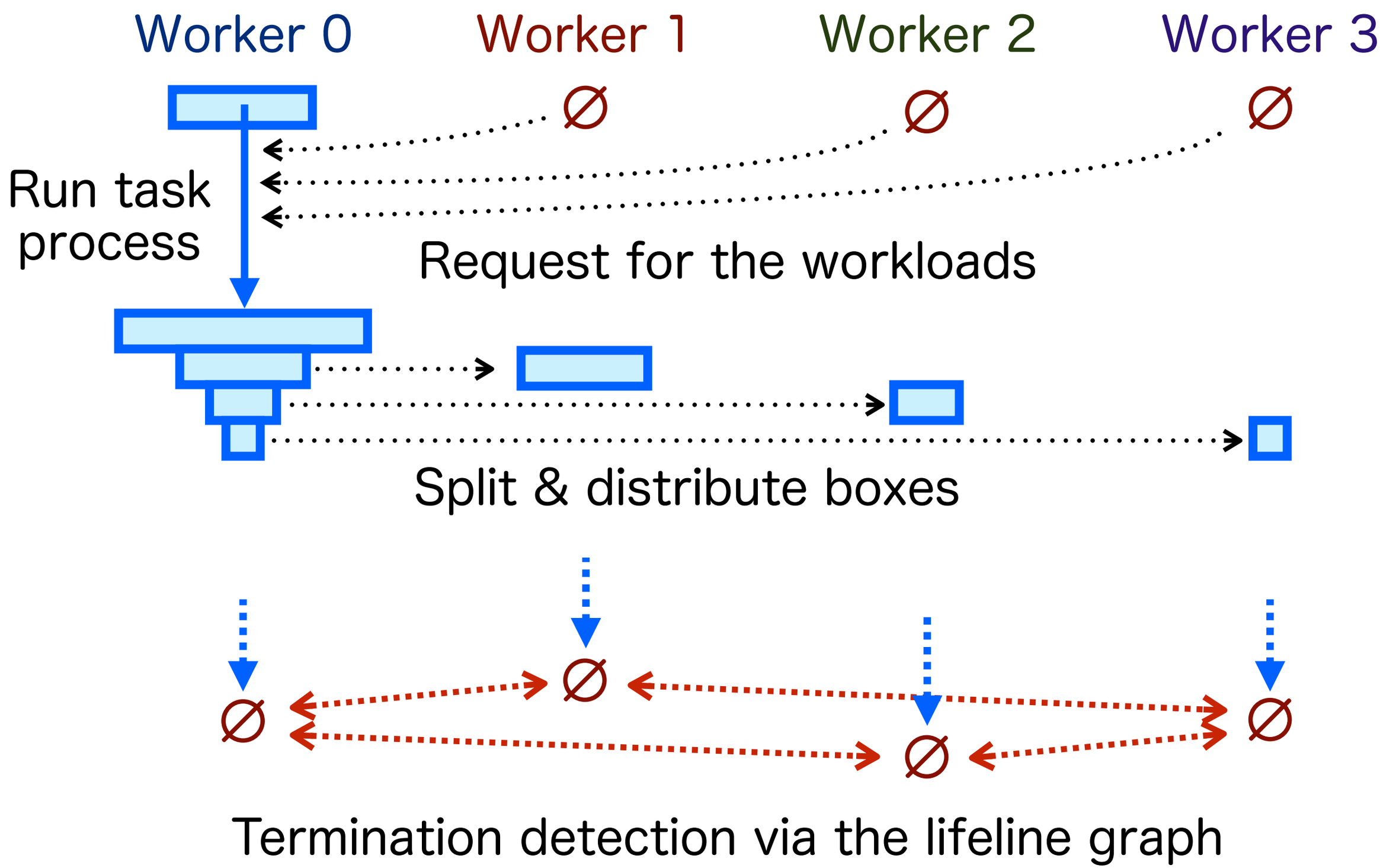
- [Saraswat+ PPoPP'11], [Zhang+ PPAA'14]
- **Generic scheme for parallelizing irregular tasks**
  - Workload for each subtask is not predictable
- **Based on a *Lifeline graph* work-stealing algorithm**
  - Provides a workload distribution/termination mechanism among a number of homogeneous workers
- **Implemented in X10 with configurable parameters**
  - User should provide `glb.TaskQueue` and `glb.TaskBag` implementations

Parallel Computation using GLB

- Each worker becomes either active or idle



Parallel Computation using GLB



TaskQueue Implementation

```
NCSPTaskQueue implements glb.TaskQueue
```

```
// Queue of boxes.
val queue : List[IntervalVec];

def process(i:Double) : Boolean {
  // Run Branch&Prune during i seconds.
}
def split() : TaskBag {
  // Split the queue into halves and
  // return a portion as a TaskBag.
}
def merge(bag:TaskBag) {
  // Append the bag into the queue.
}
def getResult() : Long {
  // Return # boxes in the queue.
}
```

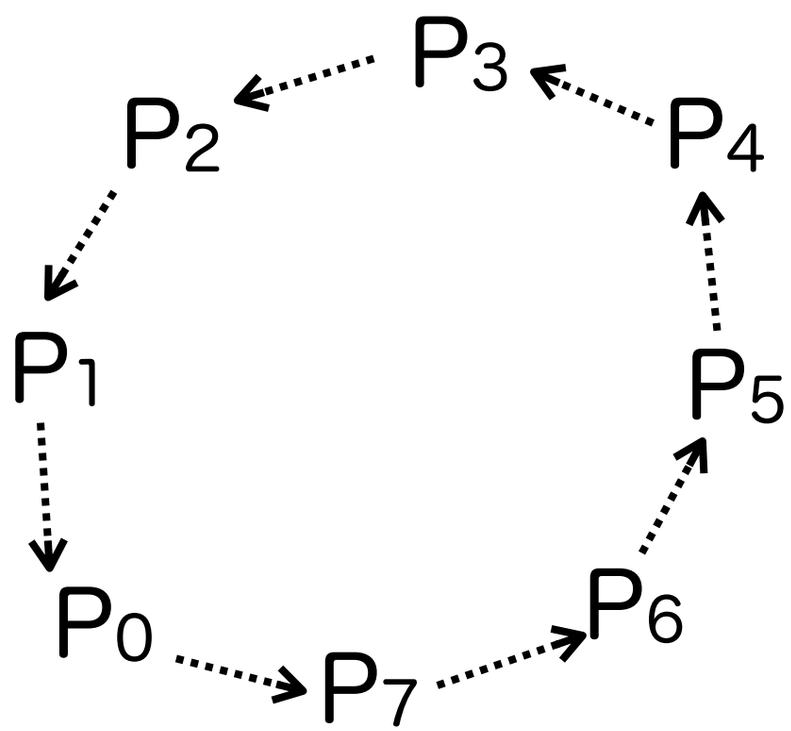
TaskBag Implementation

```
NCSPTaskBag implements glb.TaskBag
```

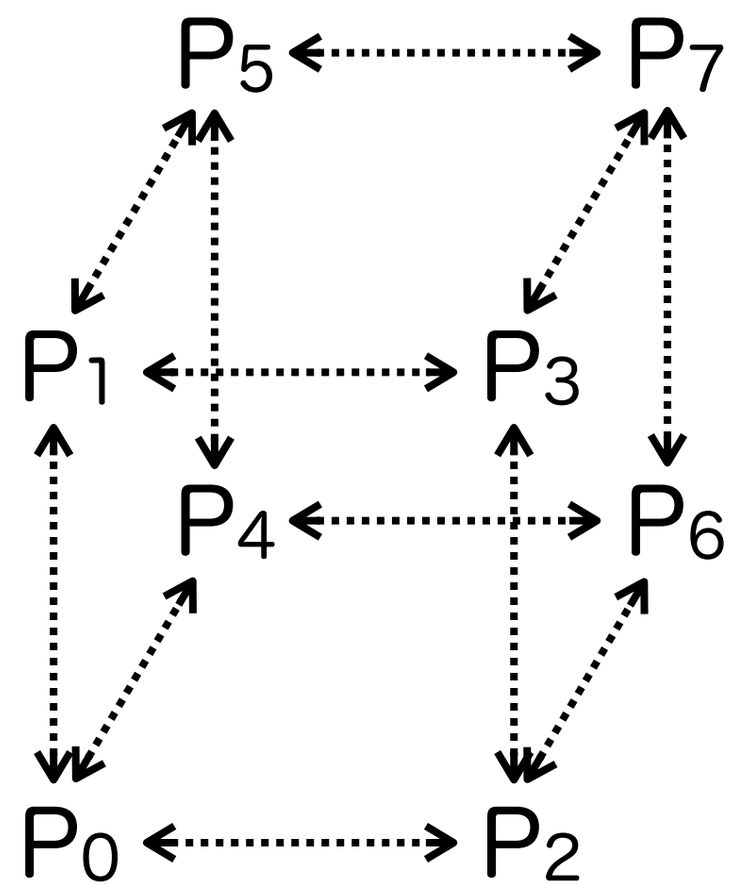
```
// Array of (undecided) boxes.
val data : Rail[IntervalVec];

def size() : Long {
  // # of boxes
  return data.size();
}
```

Lifeline

- Hyper-cube-shaped graph between the workers/ places
- Parameters (assume $l^z \geqq P$)
  - l: diameter
  - z: # branches



l=8, z=1



l=2, z=3

Talk Outline

1. Introduction
2. Branch-and-Prune algorithm and parallel method
3. **Experimental results**

Implementation

- **Implemented the proposed method with C++ and X10 (ver. 2.4.3.2, MPI backend)**
  - Deploys a worker on a X10 place (i.e. a CPU core)
- Used libraries
  - Realpaver: a sequential C++ implementation of branch and prune
- Code available at https://github.com/dsksh/icpx10

TSUBAME Supercomputer @Tokyo Tech.

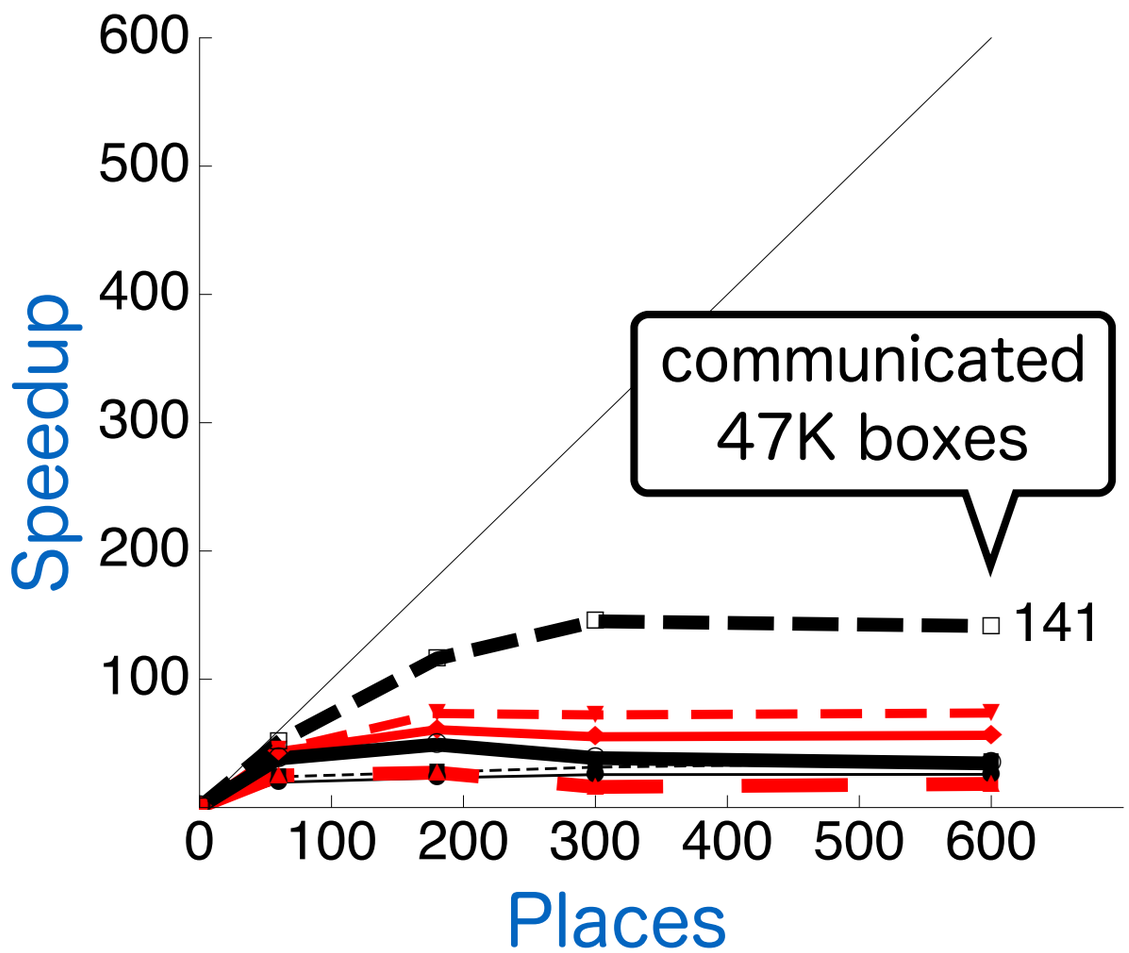
- **CPU**: Intel Westmere EP (Xeon X5670 2.93GHz, 12 CPU cores per node)
- **RAM**: 54GB
- **# of nodes**: 1466 (we used 50 nodes)
- **Network**: Dual-rail QDR Infiniband (80Gbps)

Experiments

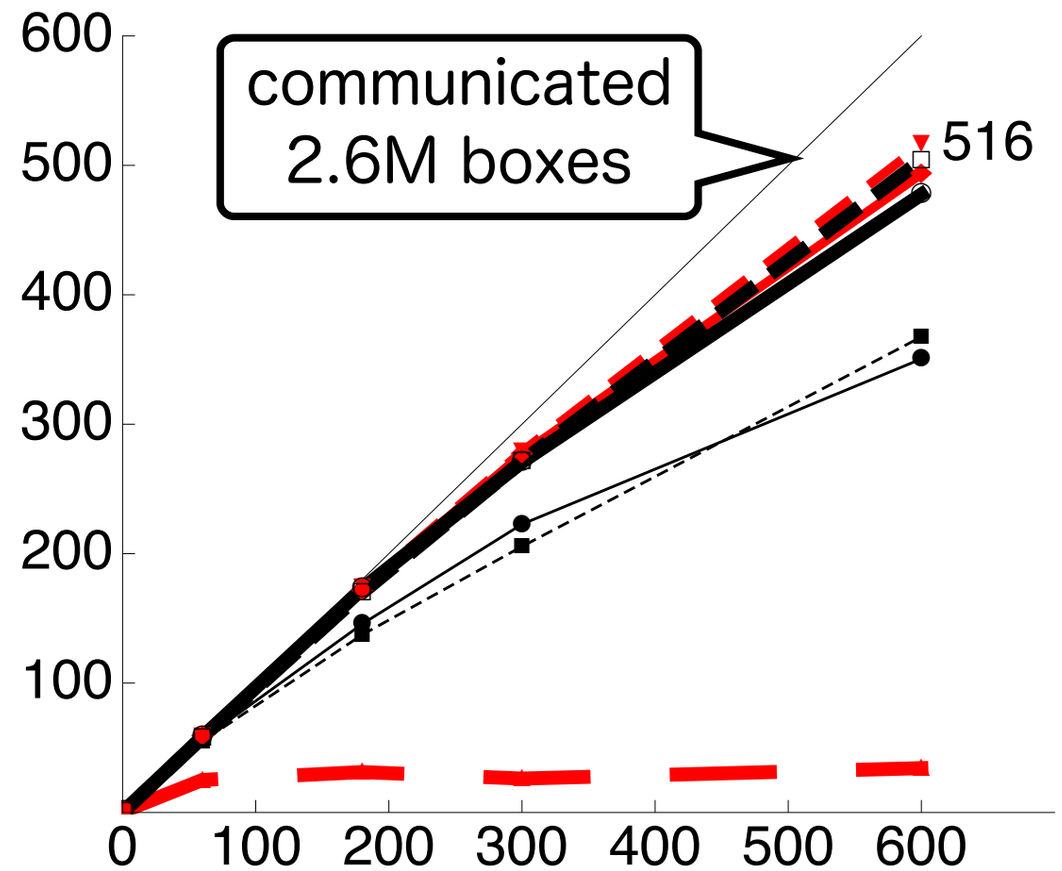
- **Solved 10 instances of 4 problems**
  - Instances are generated by varying
    - # of variables
    - # of constraints
    - box precision (ε)
- **Solving parameters**
  - i: time interval between load balancing
  - l: diameter of the lifeline; z: $\mathrm{ceil}(\log_l P)$
  - w: # of branches in the lifeline
- **Solved with 7 parameter configurations**

(1)	i=0.001s,	l=2,	w=0
(2)	i=0.001s,	l=2,	w=1
(3)	i=0.001s,	l=2,	w=z
(4)	i=0.001s,	l=P,	w=0
(5)	i=0.001s,	l=P,	w=z
(6)	i=0.1s,	l=2,	w=0
(7)	i=0.1s,	l=2,	w=z

Speedup (Economics Model)



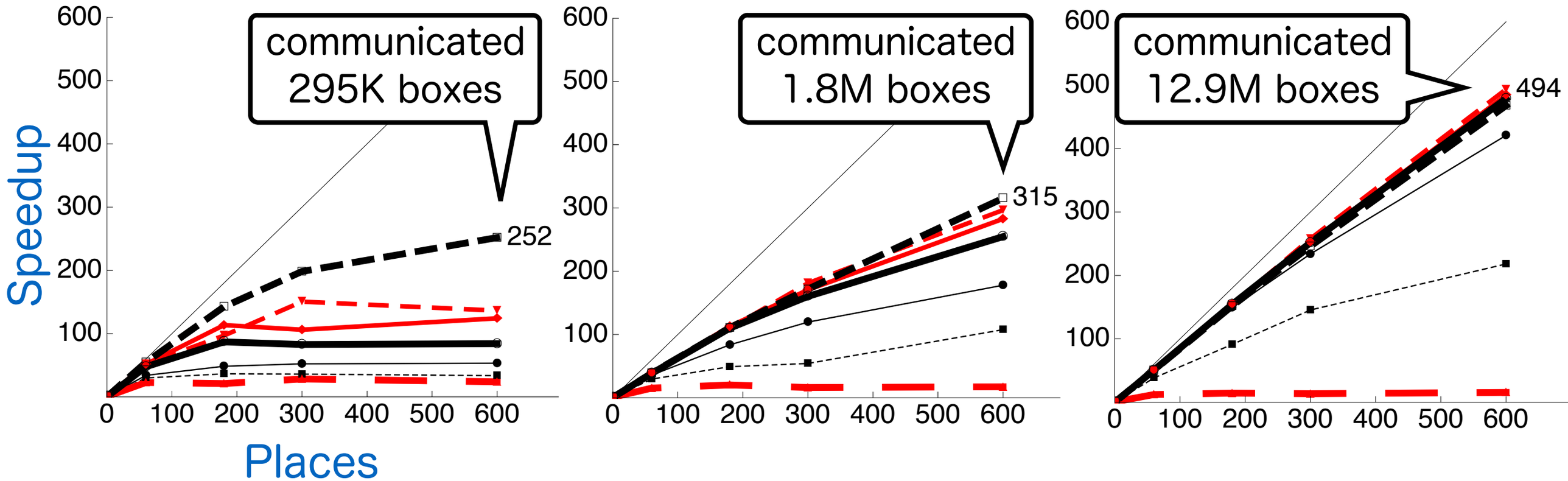
eco-8-10^{-8}



eco-10-10^{-8}

(1) i=0.001s, l=2, w=0
(2) i=0.001s, l=2, w=1
(3) i=0.001s, l=2, w=z
(4) i=0.001s, l=P, w=0
(5) i=0.001s, l=P, w=z
(6) i=0.1s, l=2, w=0
(7) i=0.1s, l=2, w=z

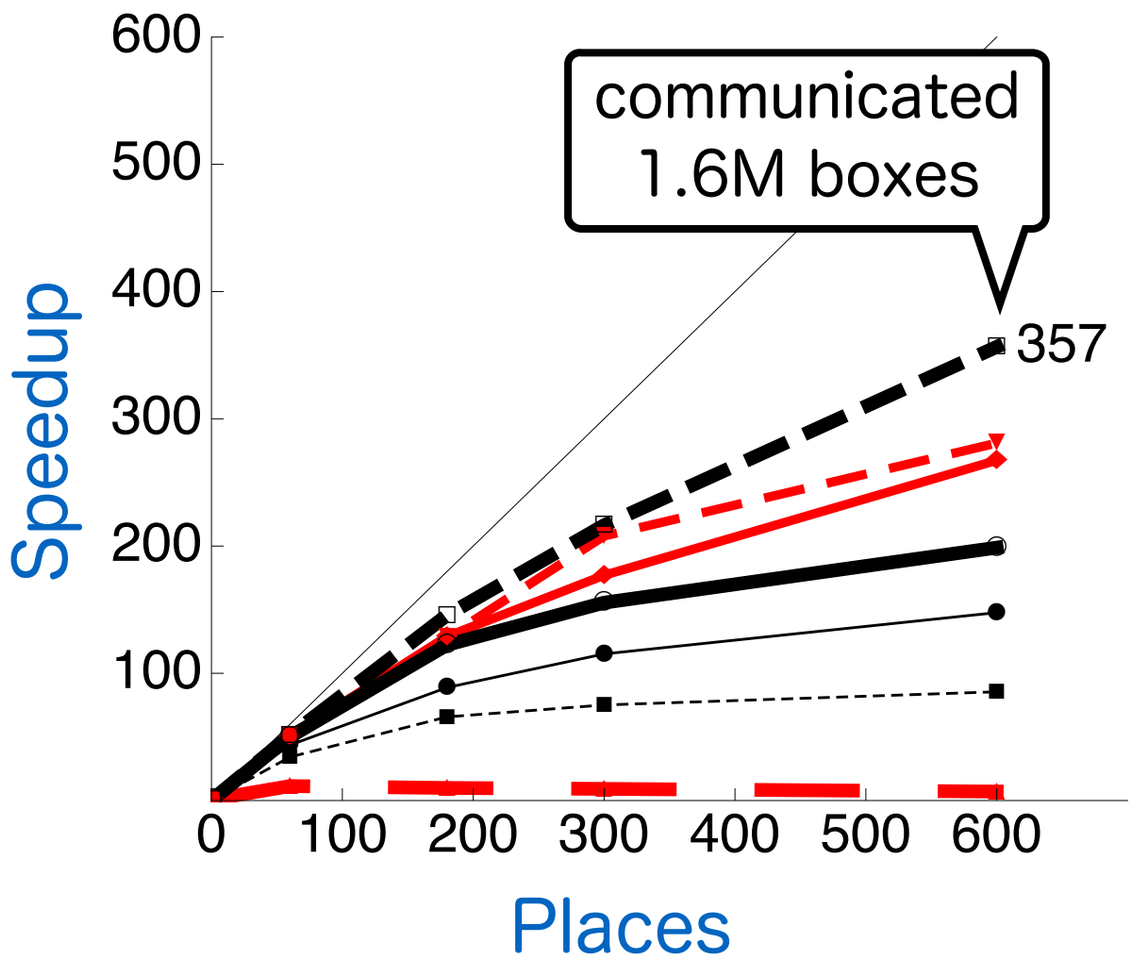
Speedup (Intersection of Sphere and Planes)



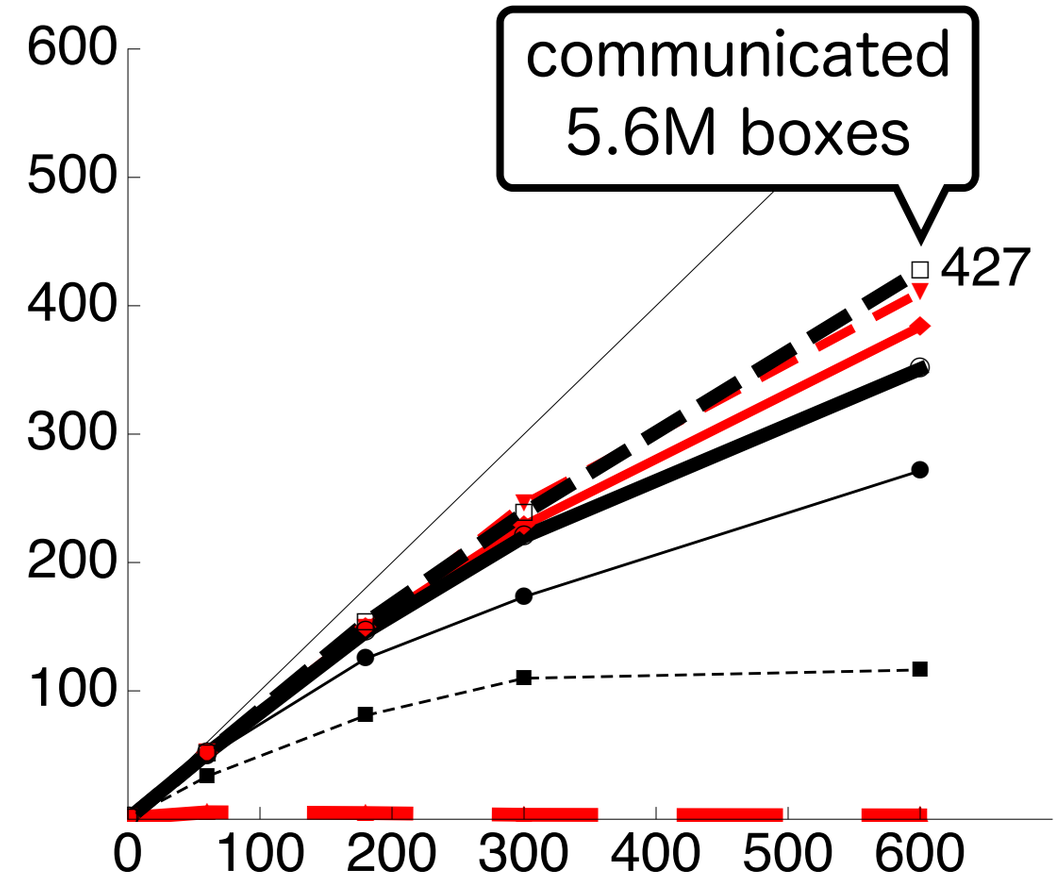
sp-2-2-0.004 sp-2-4-0.004 sp-2-4-0.001

(1) i=0.001s, l=2, w=0
(2) i=0.001s, l=2, w=1
(3) i=0.001s, l=2, w=z
(4) i=0.001s, l=P, w=0
(5) i=0.001s, l=P, w=z
(6) i=0.1s, l=2, w=0
(7) i=0.1s, l=2, w=z

Speedup (3-RPR Robot)



3rpr-3-3-0.2



3rpr-3-3-0.1

(1) *i*=0.001s, *l*=2, *w*=0
(2) *i*=0.001s, *l*=2, *w*=1
(3) *i*=0.001s, *l*=2, *w*=*z*
(4) *i*=0.001s, *l*=*P*, *w*=0
(5) *i*=0.001s, *l*=*P*, *w*=*z*
(6) *i*=0.1s, *l*=2, *w*=0
(7) *i*=0.1s, *l*=2, *w*=*z*

Discussion

- **Our parallel solver scaled up to 600 places/ cores**
  - Achieved up to 516-fold speedup (efficiency of 0.84)

Discussion

- **Configuration (1) accomplished the best speedups for most of the problems**
  - Workload distribution and termination via lifeline were quick
  - …despite its large communication overhead

- **For large instances, configurations (2), (3), and (5) outperformed configuration (1)**
  - W. frequent random stealing
  - Active ratio in the CPU time was high
  - Configuration (2) often performed better because of its quick termination process

Discussion

- **Computation w. the smallest time interval *i* performed well**
  - Workload distribution once after almost every Prune process

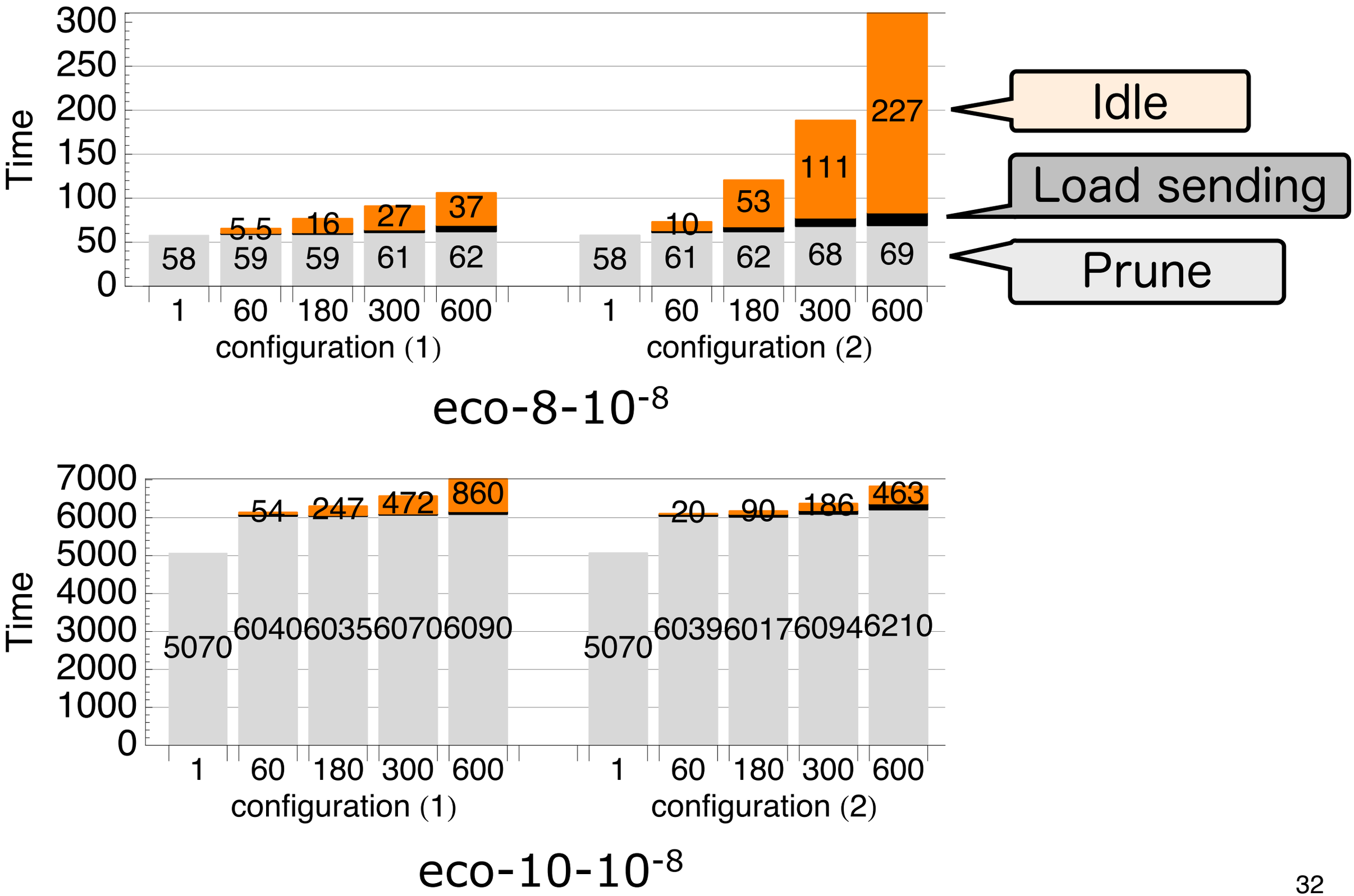
- **Configuration (4) performed poorly**
  - Lifeline was formed as a 1D hyper cube (ring)
  - Enabling the random stealing improved the computation (Configuration (5))

Discussion

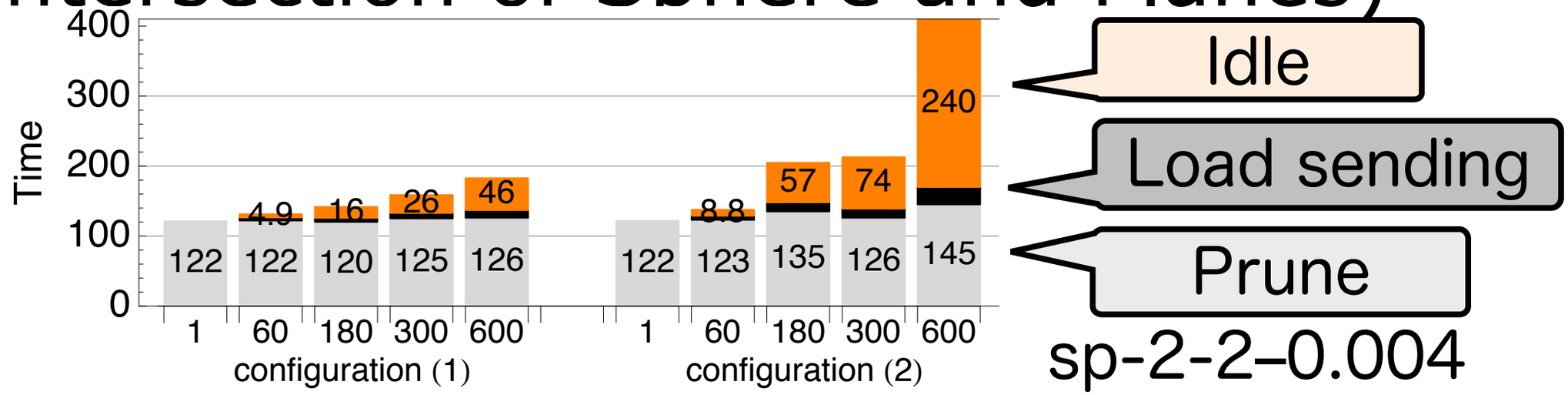
- **Configuration (1) often scaled better than other configurations when using 600 cores**
  - Random stealing w. many cores might suffered from large communication overhead

- **Certain speedups were achieved even w. quite short running time**

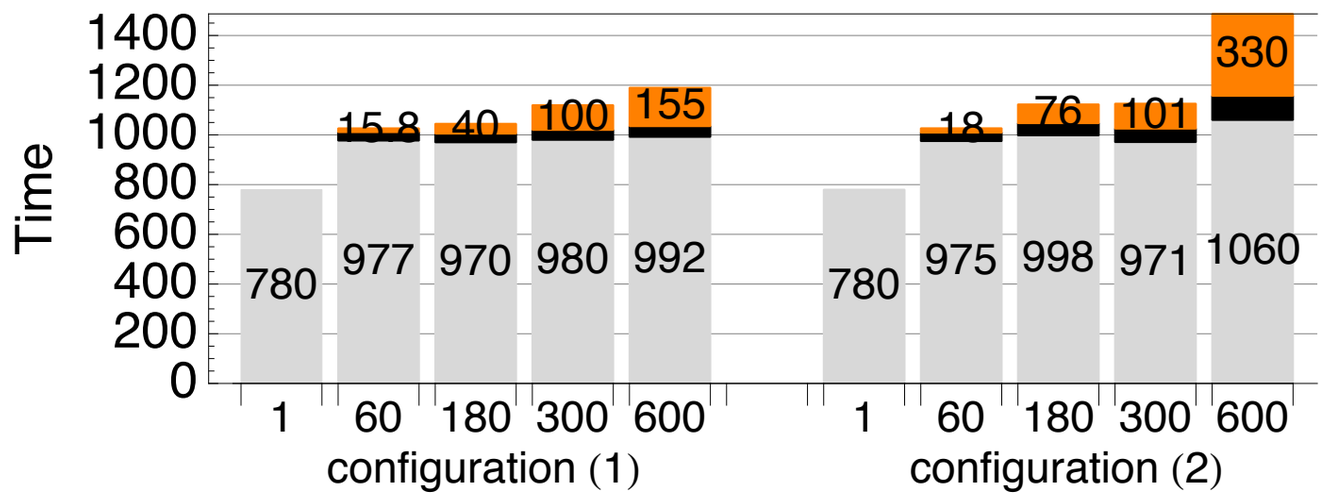
CPU Timing Breakdown (Economics Model)



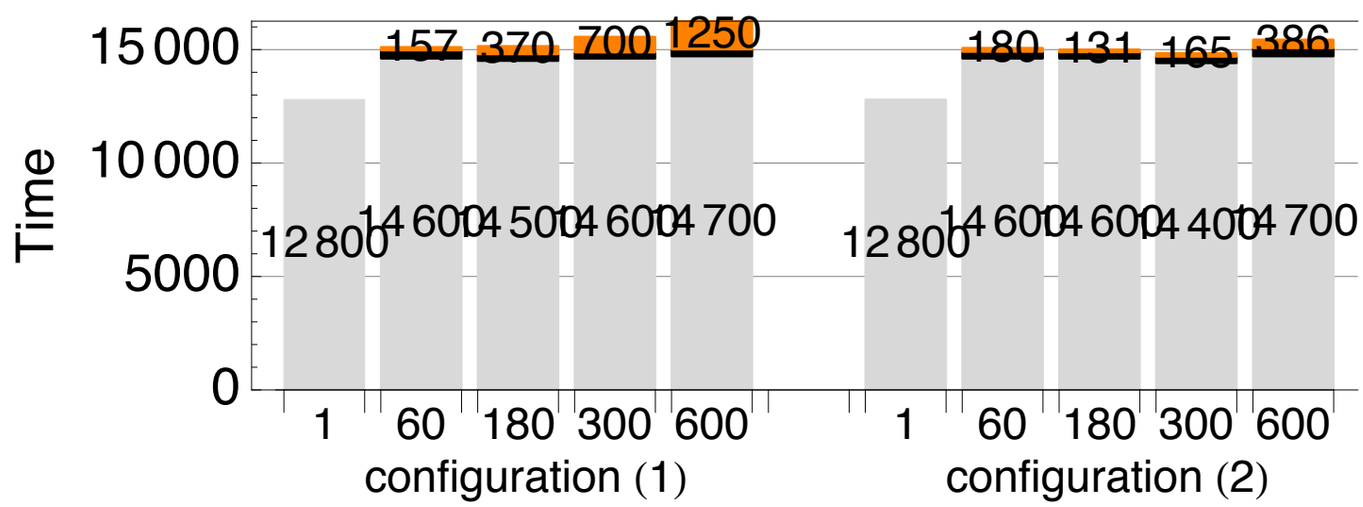
CPU Timing BreakdownSpeedup
(Intersection of Sphere and Planes)



sp-2-2–0.004



sp-2-4–0.004



sp-2-4–0.001

Conclusion

- **Parallel solving of NCSPs is a good application of GLB**
- Experimental results with 600 X10 places: Achieved almost linear speedup up to 516 fold
- Code available at https://github.com/dsksh/icpx10

- **Future work:**
  - Estimation of parameter configurations
  - Parallelize Prune
  - Application: e.g. large robotics problems