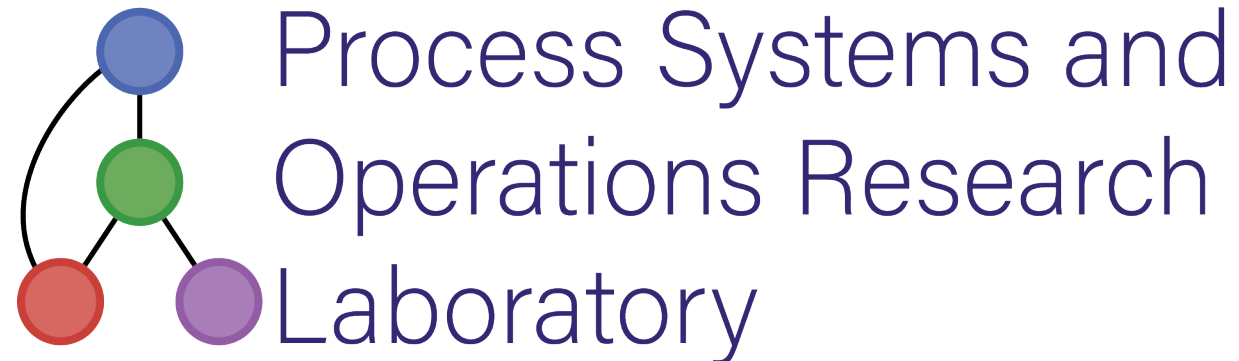


# Toward Extending EAGO.jl for Exploiting GPU Parallelism

Dimitri Alston, Robert Gottlieb

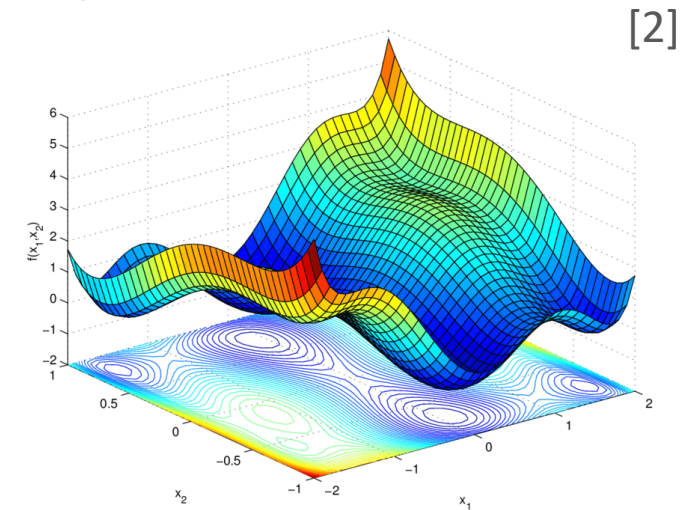
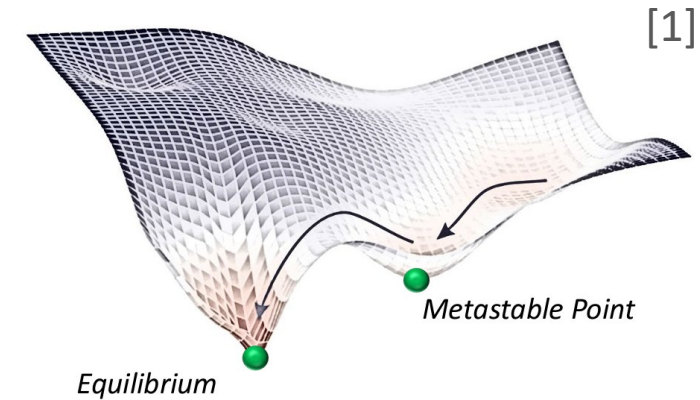
Matthew Stuber, P&W Associate Professor in  
Advanced Systems Engineering

June 1<sup>st</sup>, 2026



# Nonconvex Optimization

- Nonconvex problems are widespread in practice
- Convex solution methods may miss the global optimum
  - Multiple feasible solutions may exist
- **Deterministic global optimization** guarantees optimality or infeasibility
  - Techniques for nonconvex problems are more expensive



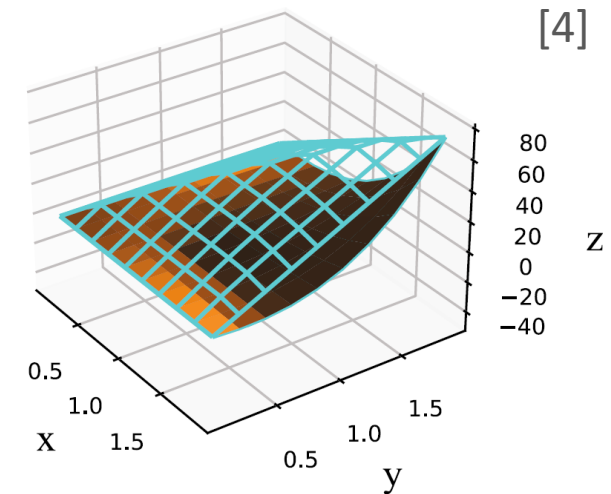
[1] Grajcarova, L. Simulations of structural phase transitions in crystals using ab initio metadynamics. INIS-IAEA. (2013).

[2] Henrion, D. and Lasserre, J.-B. GloptiPoly: global optimization over polynomials with MATLAB and SeDuMi. Proceedings of the 41<sup>st</sup> IEEE Conference on Decision and Control. 1(2): 747-752 (2002).

# EAGO.jl

## Easy Advanced Global Optimization

- **Deterministic global solver** for nonconvex MINLPs
  - Semi-infinite programs
  - Dynamic optimization
  - User-defined functions
- Easy to formulate complicated problems
- Applies McCormick-based relaxations for convex lower-bounding problems
- Open-source and extensible



[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

[4] Wilhelm, M.E., and Stuber, M.D. Improved Convex and Concave Relaxations of Composite Bilinear Forms. Journal of Optimization Theory and Applications. 197, 174-204 (2023).

# EAGO.jl

## A Research Platform

- Focus on unsolved problems
- Designed for custom routines
  - Anyone can implement and test new ideas

```
using JuMP, EAGO

import EAGO: lower_problem!
> function EAGO.lower_problem!(t::ExtendGPU, m::EAGO.GlobalOptimizer) ...
end
```

# High-Performance Computing

Global Solver	Uses GPU
BARON <sup>[5]</sup>	No
ANTIGONE <sup>[6]</sup>	No
SCIP <sup>[7]</sup>	No
MAiNGO <sup>[8]</sup>	Yes (2025)
EAGO <sup>[3]</sup>	Yes (2022)
[...]	[...]

[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

[5] Sahinidis, N.V. BARON: A general purpose global optimization software package. Journal of Global Optimization. 8(2): 201-205 (1996).

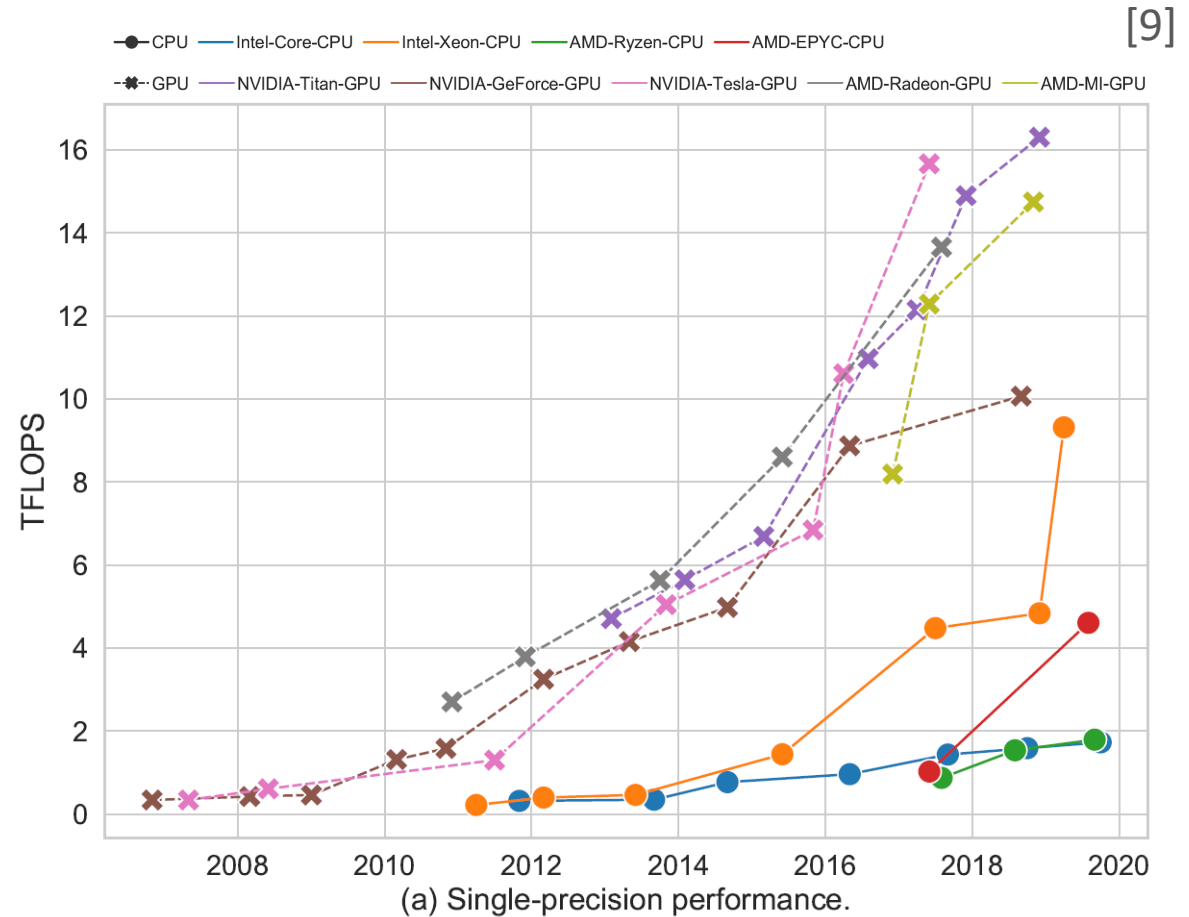
[6] Misener, R. and Floudas, C.A. ANTIGONE: Algorithms for coNtinuous / Integer Global Optimization of Nonlinear Equations. Journal of Global Optimization. 59(2-3): 503-526 (2014).

[7] Vigerske, S. and Gleixner, A.. SCIP: global optimization of mixed-integer non-linear programs in a branch-and-cut framework. Optimization Methods and Software. 33(3): 563-593 (2017).

[8] Bongartz, D., et al. MAiNGO - McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization. Technical report, RWTH-Aachen (2018).

# GPU Architecture

- Single-instruction, multiple threads (SIMT) execution mode performs well on specialized tasks
  - Machine learning model training<sup>[10]</sup>
  - Data analysis<sup>[11]</sup>
  - Large-scale simulations<sup>[12]</sup>
  - Etc.



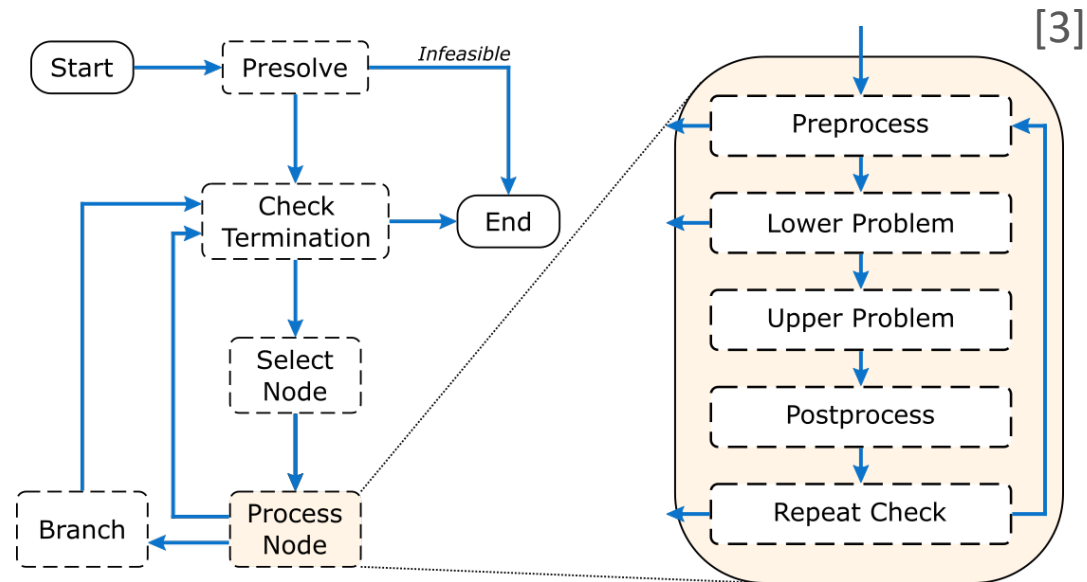
[9] Sun, Y., et al. Summarizing CPU and GPU design trends with product data. (2019).

[10] Steinkraus, D., et al. Using GPUs for machine learning algorithms. Eighth International Conference on Document Analysis and Recognition. 2: 1115-1120 (2005).

[11] Singh, H., et al. GPU and CUDA in Hard Computing Approaches: Analytical Review. Proceedings of ICRIC 2019. 177-196 (2020).

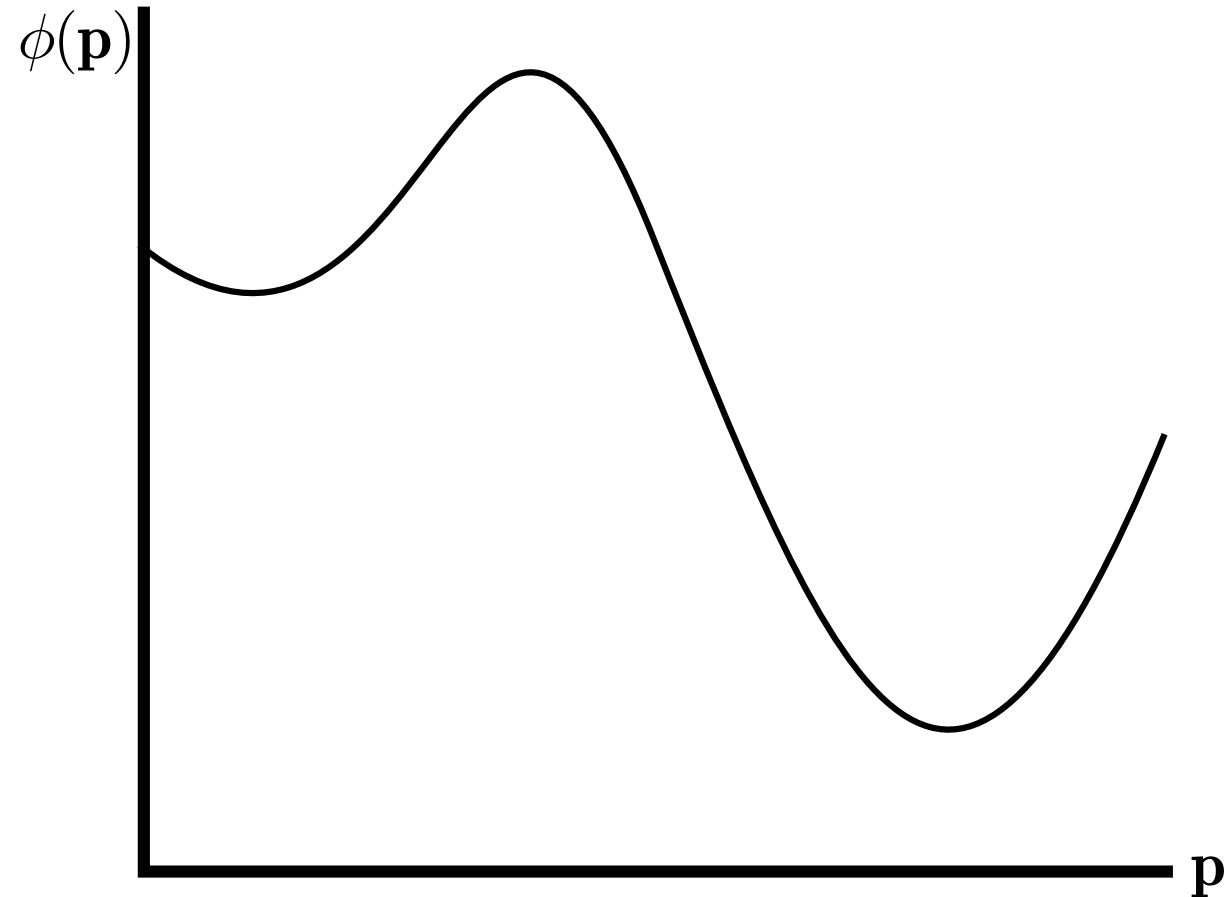
[12] Stone, J.E., et al. GPU-accelerated molecular modeling coming of age. Journal of Molecular Graphics and Modelling. 29(2): 116-125 (2010).

# Branch-and-Bound Algorithm

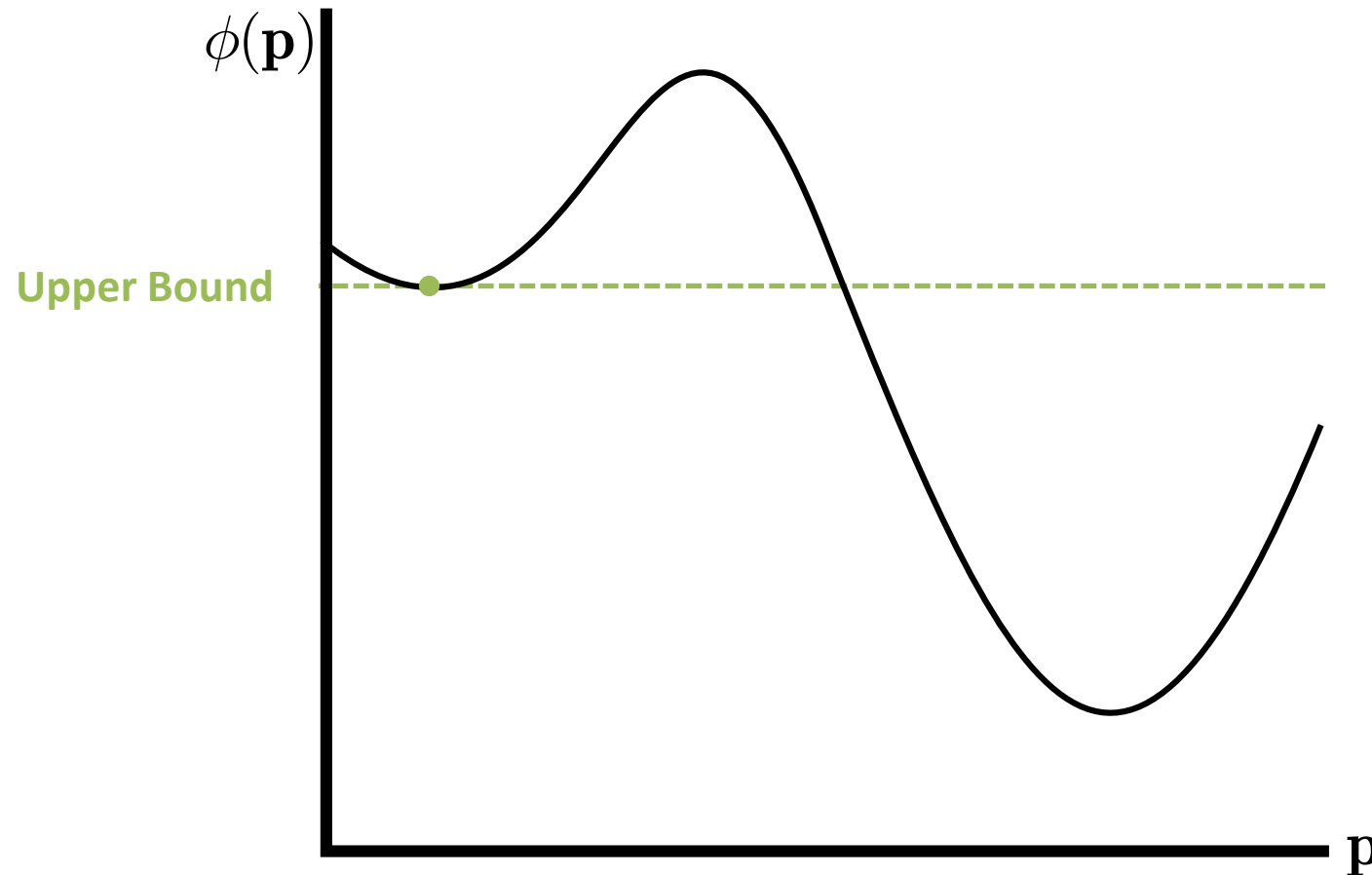


[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

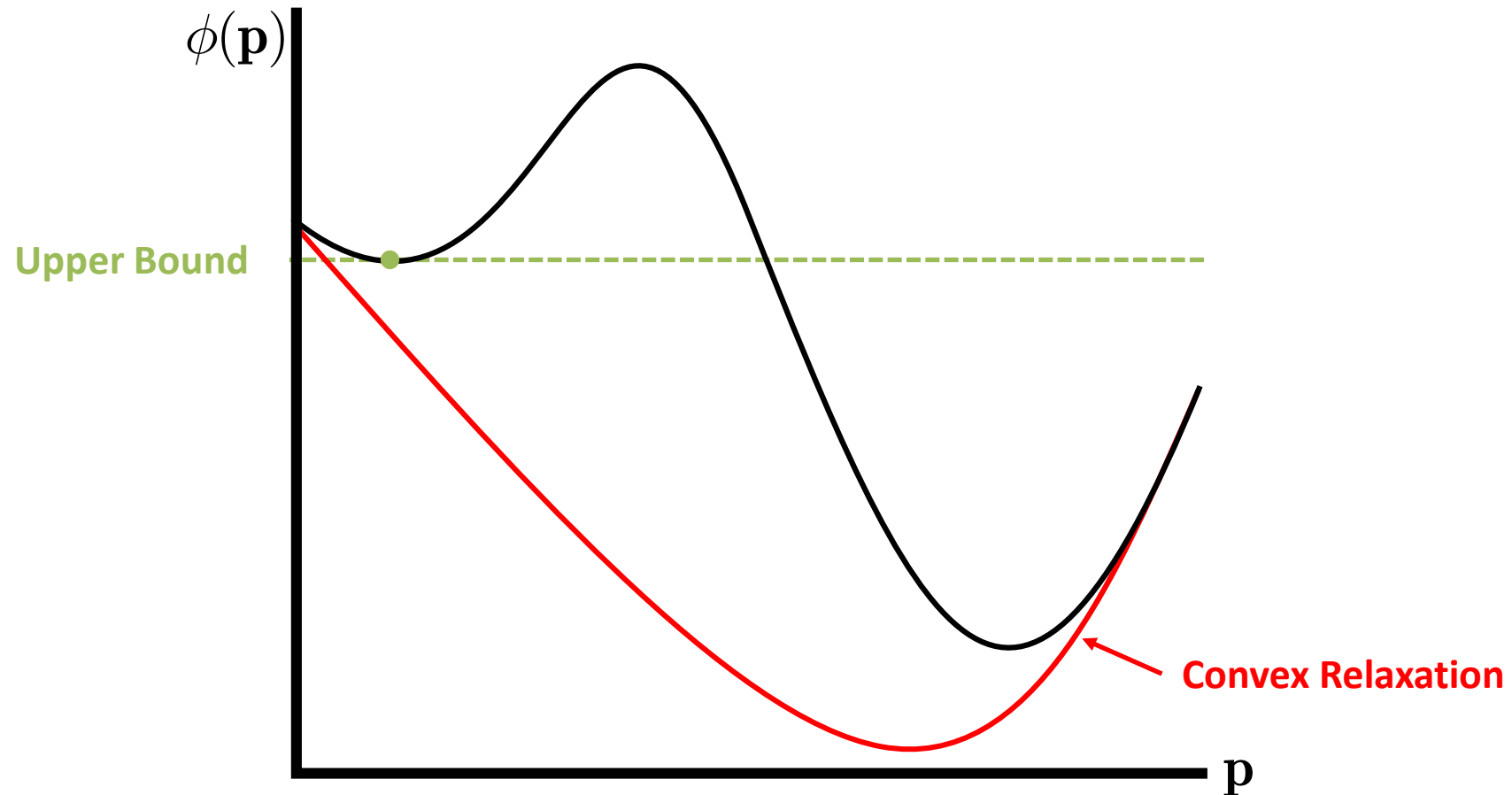
# Branch-and-Bound Algorithm



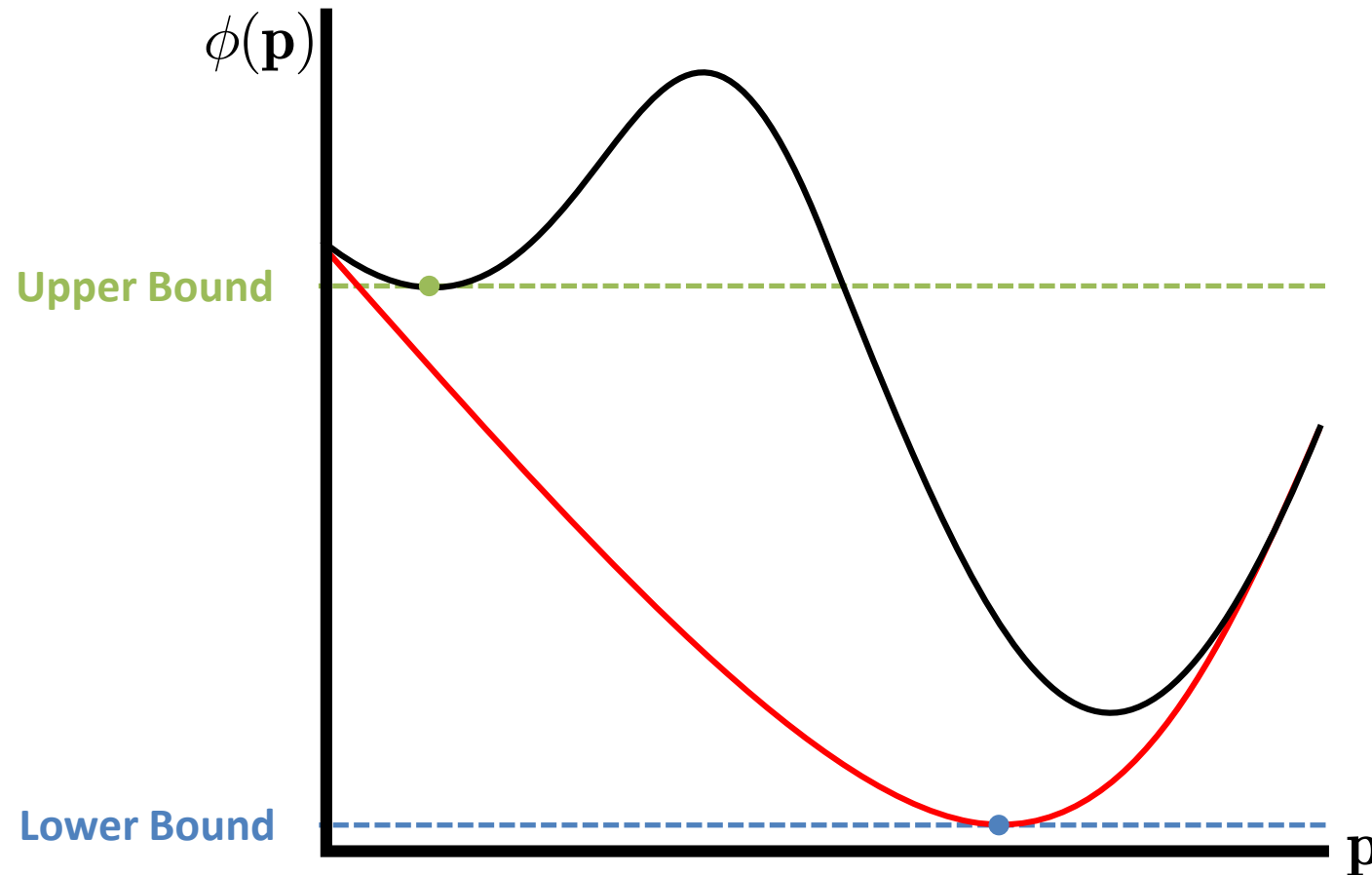
# Branch-and-Bound Algorithm



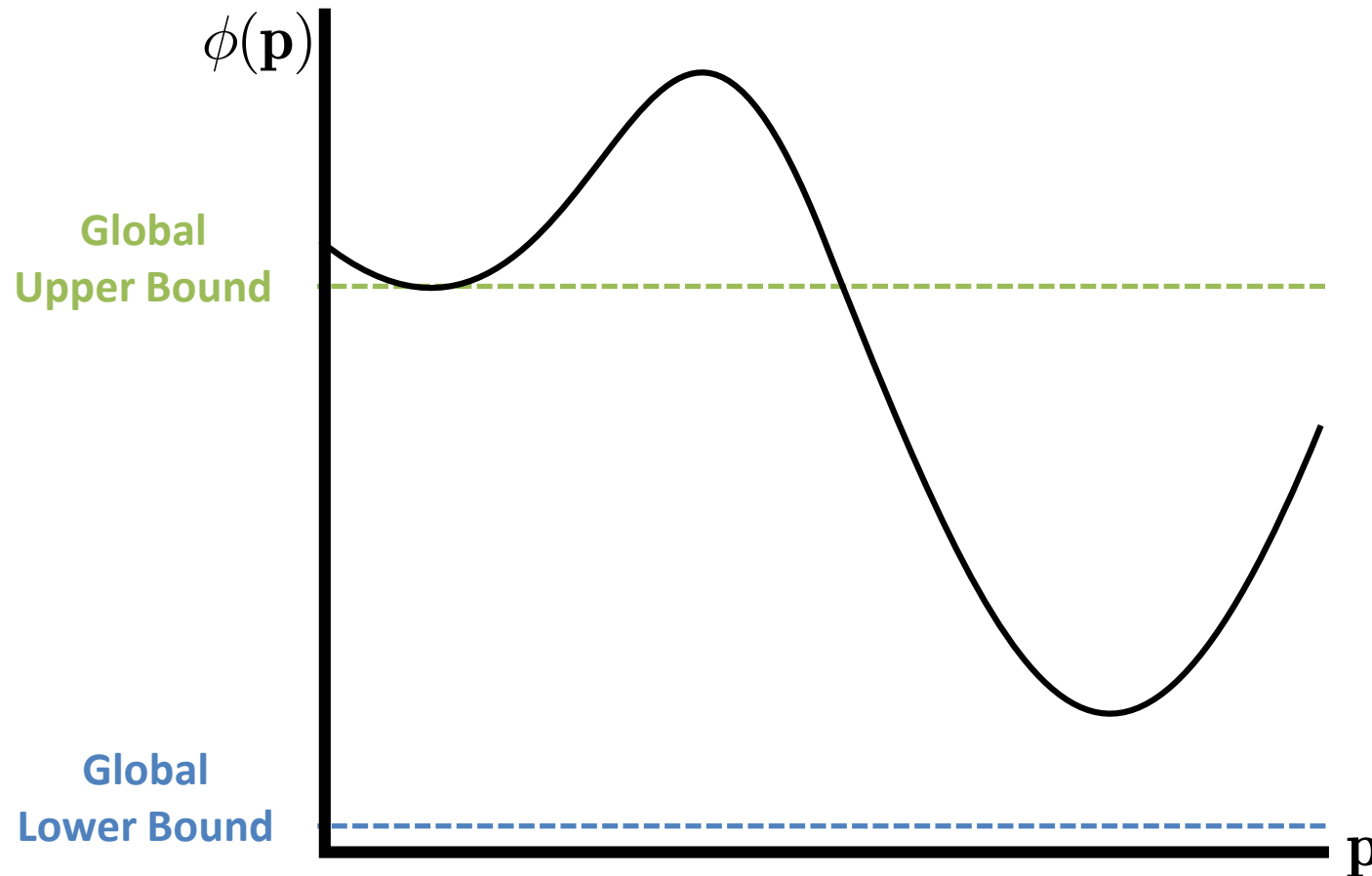
# Branch-and-Bound Algorithm



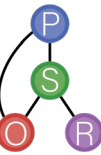
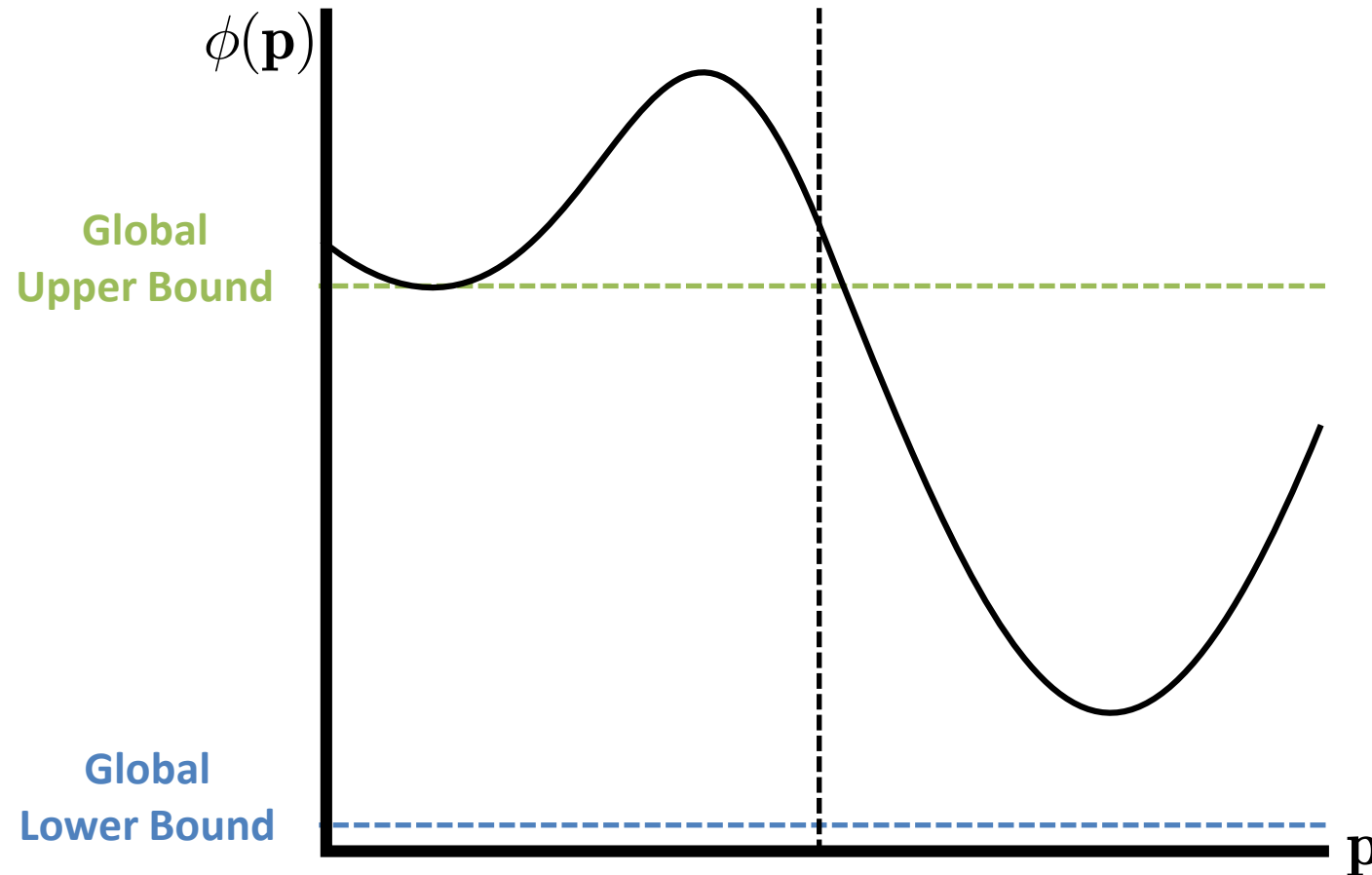
# Branch-and-Bound Algorithm



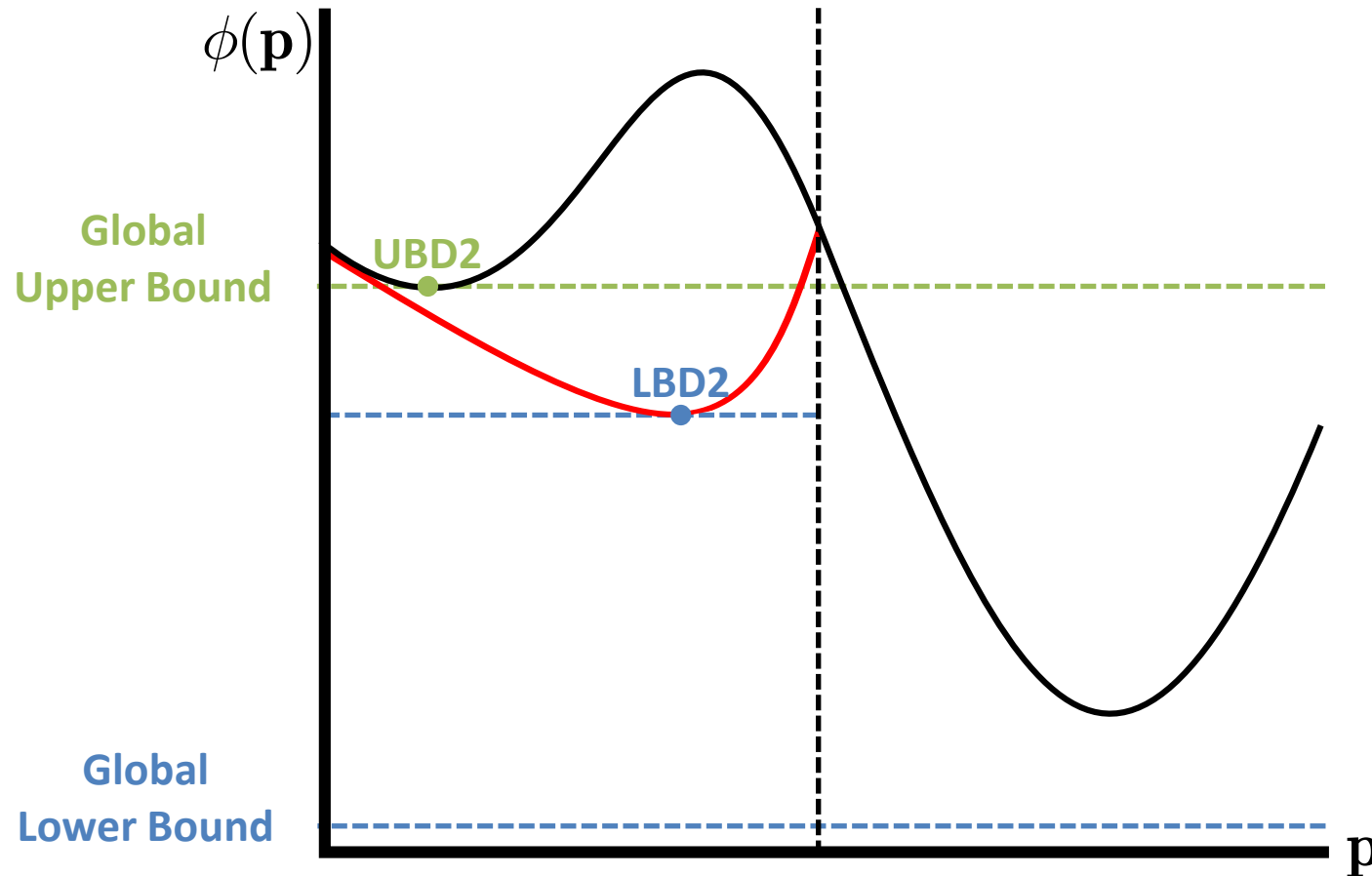
# Branch-and-Bound Algorithm



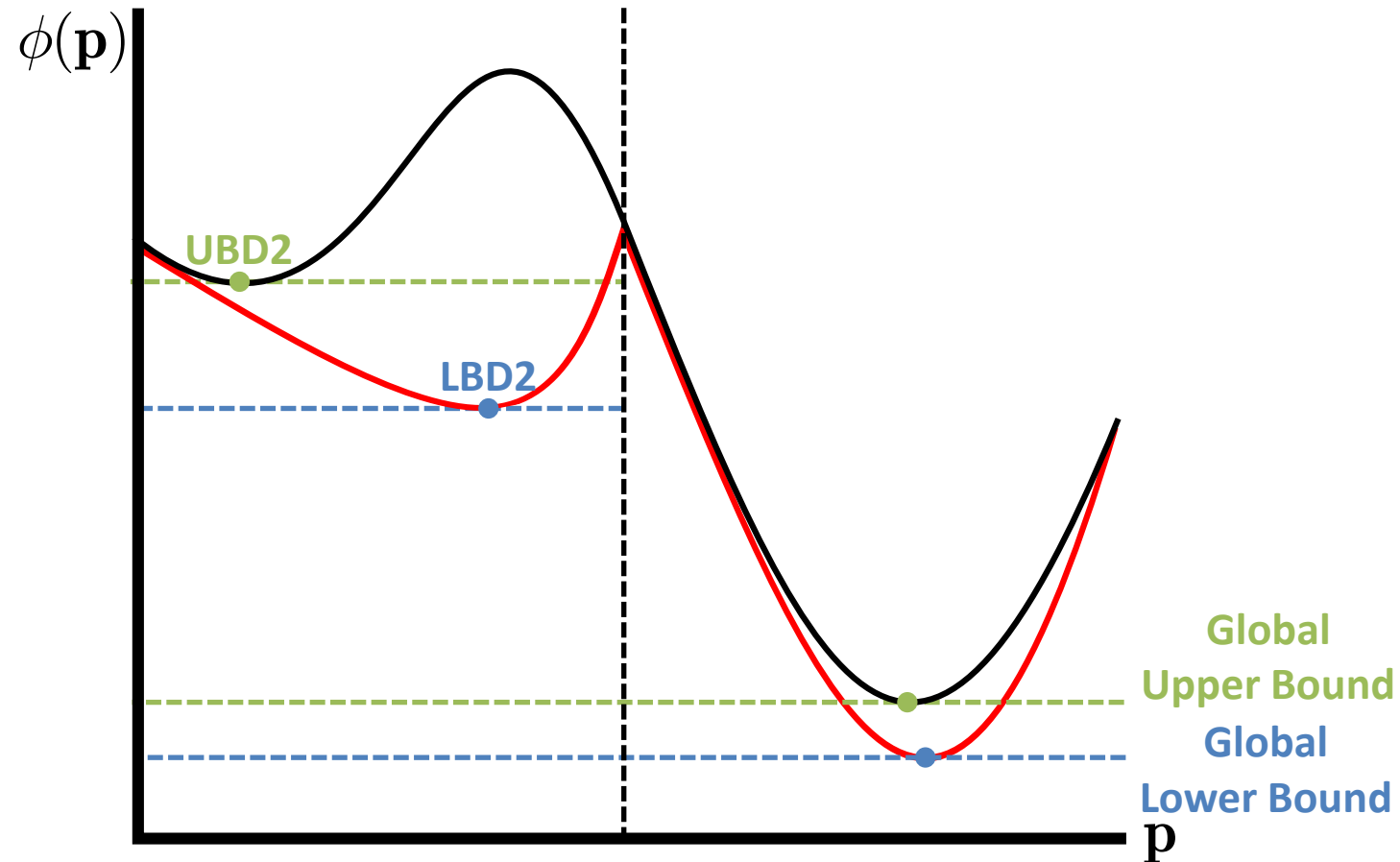
# Branch-and-Bound Algorithm



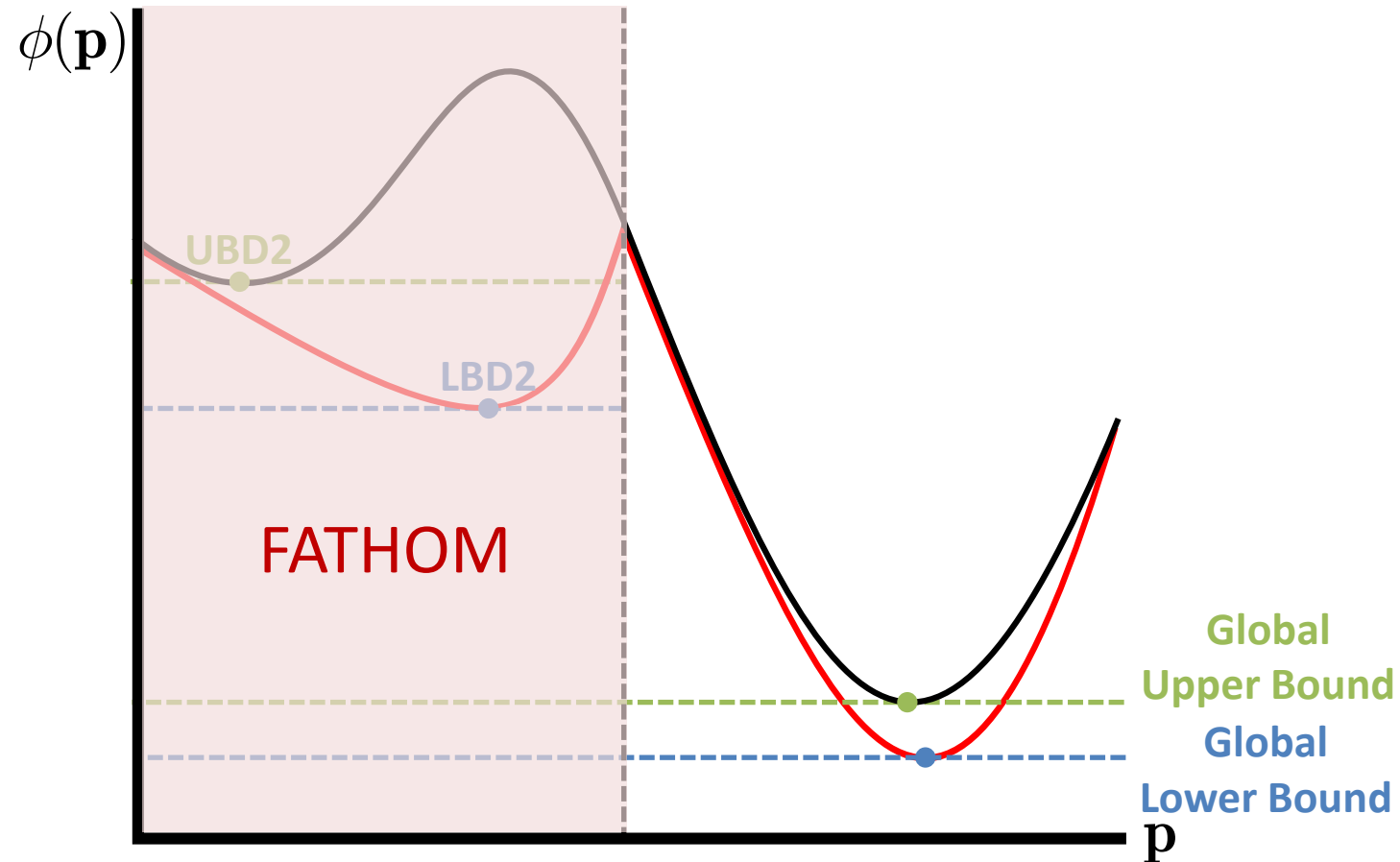
# Branch-and-Bound Algorithm



# Branch-and-Bound Algorithm

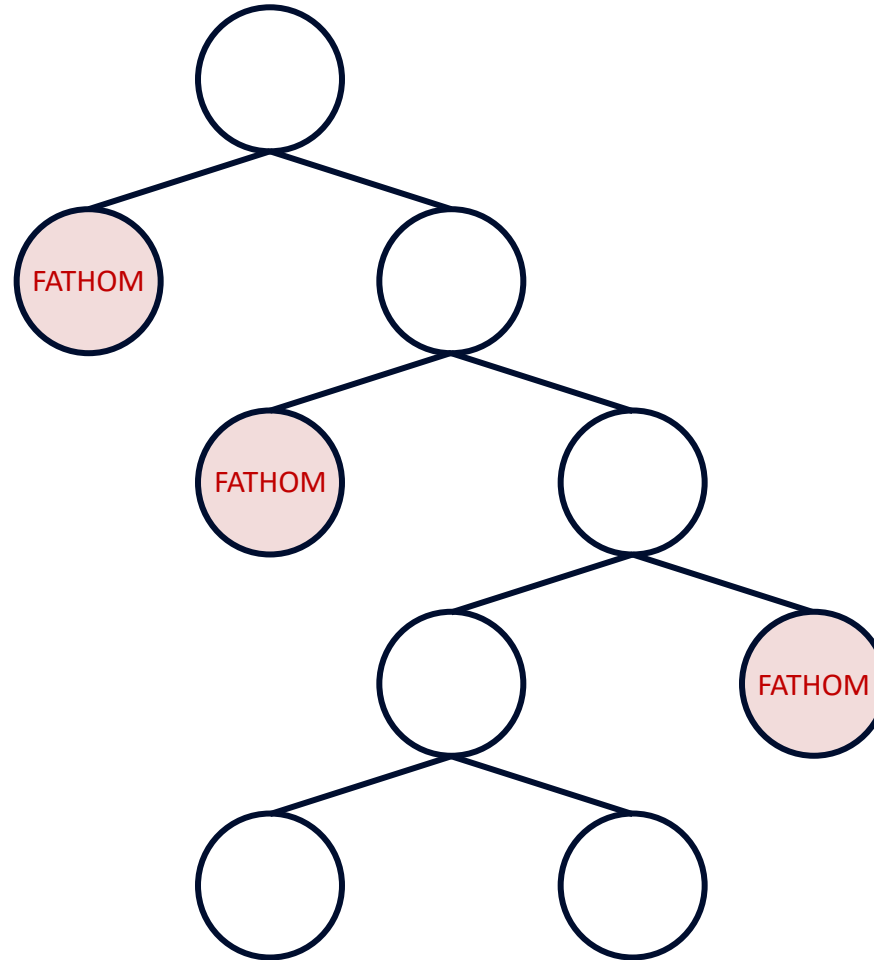


# Branch-and-Bound Algorithm



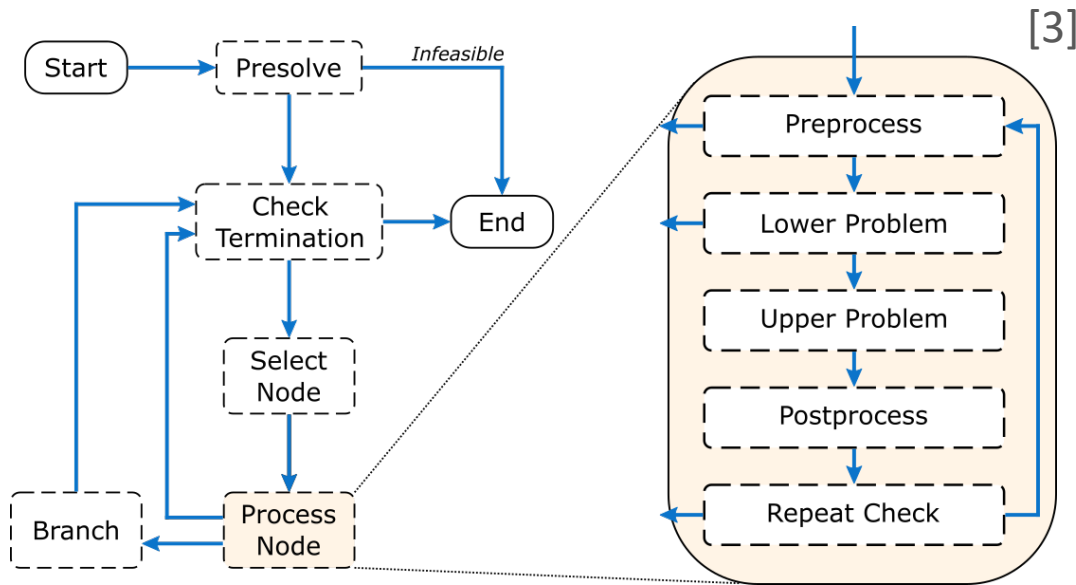
# Branch-and-Bound Algorithm

**Worst-Case  
Exponential Runtime**  
 $O(2^n)$



# Aligning Branch-and-Bound with GPUs

## Branch-and-Bound Algorithm



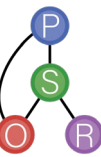
## GPU Architecture



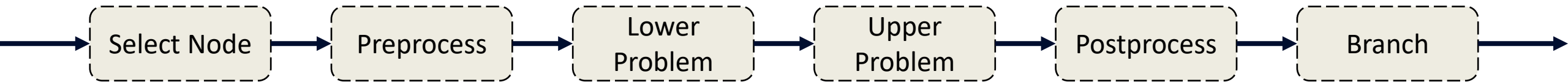
[14]

[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

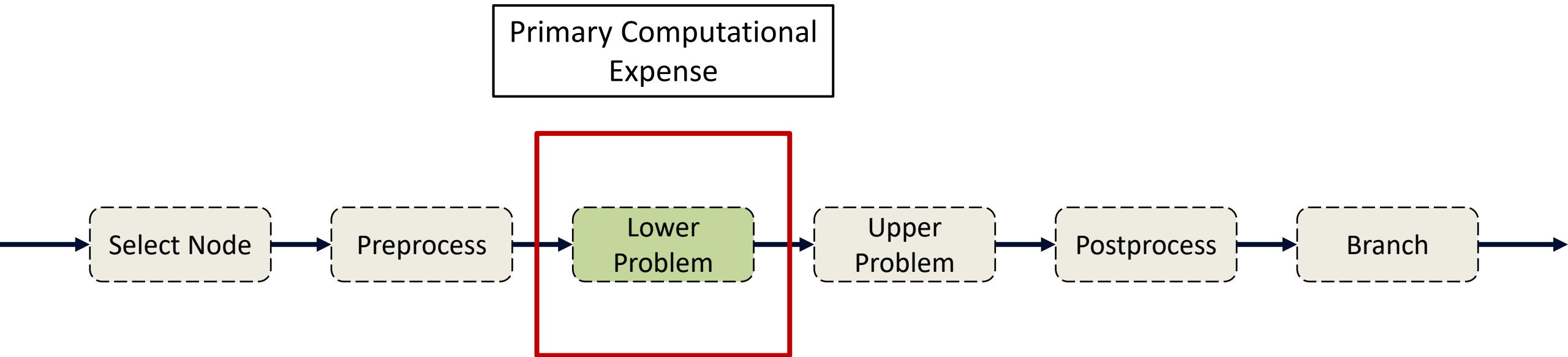
[14] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).



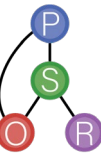
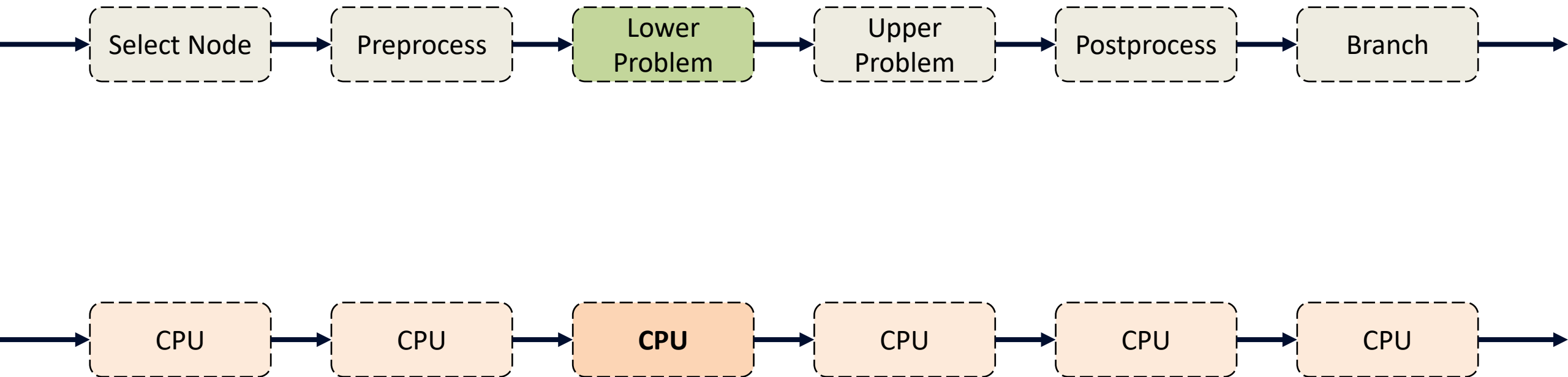
# Serial CPU Branch-and-Bound



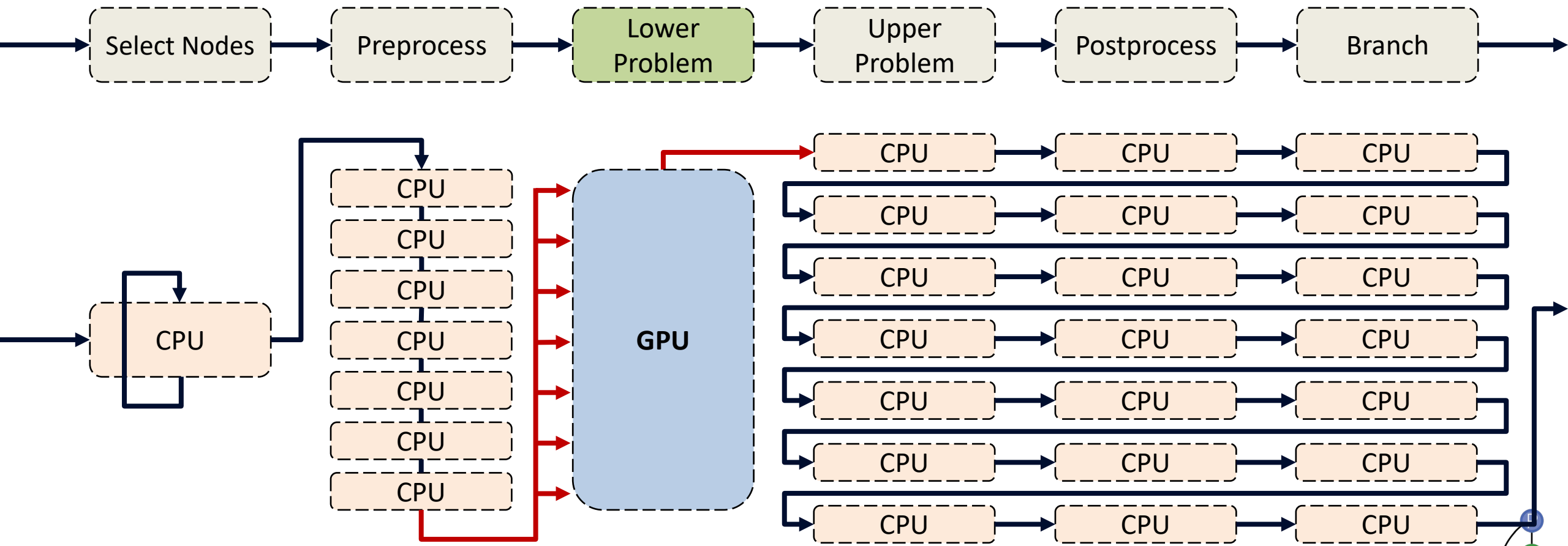
# Serial CPU Branch-and-Bound



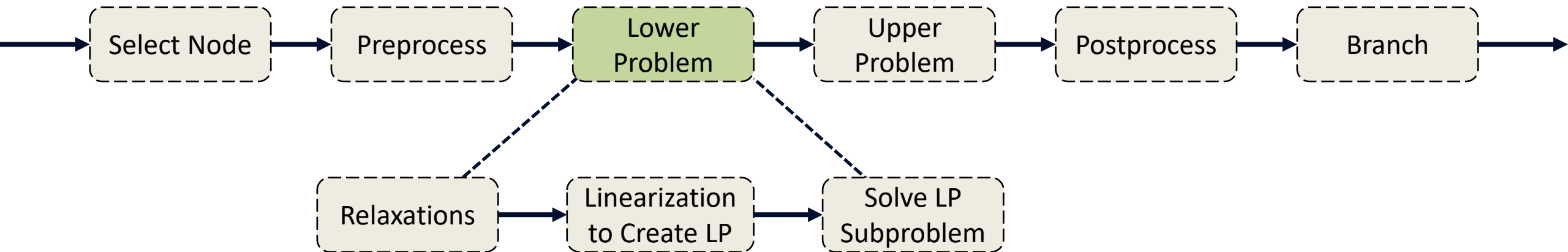
# Serial CPU Branch-and-Bound



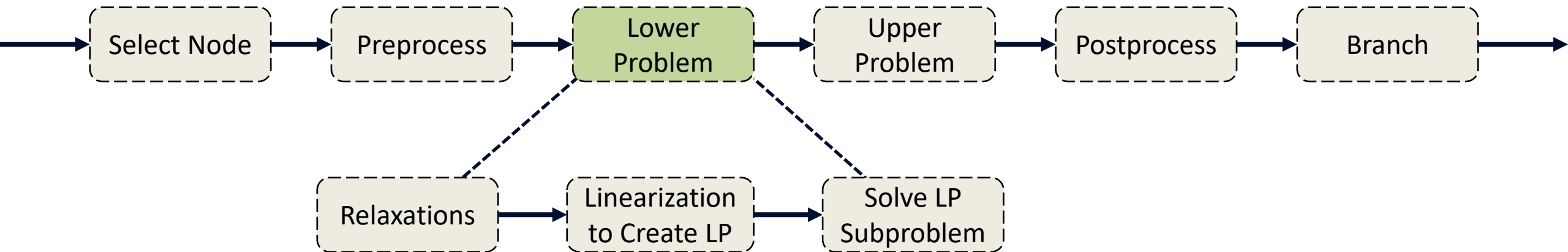
# Serial CPU Branch-and-Bound



# Serial CPU Branch-and-Bound



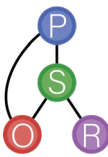
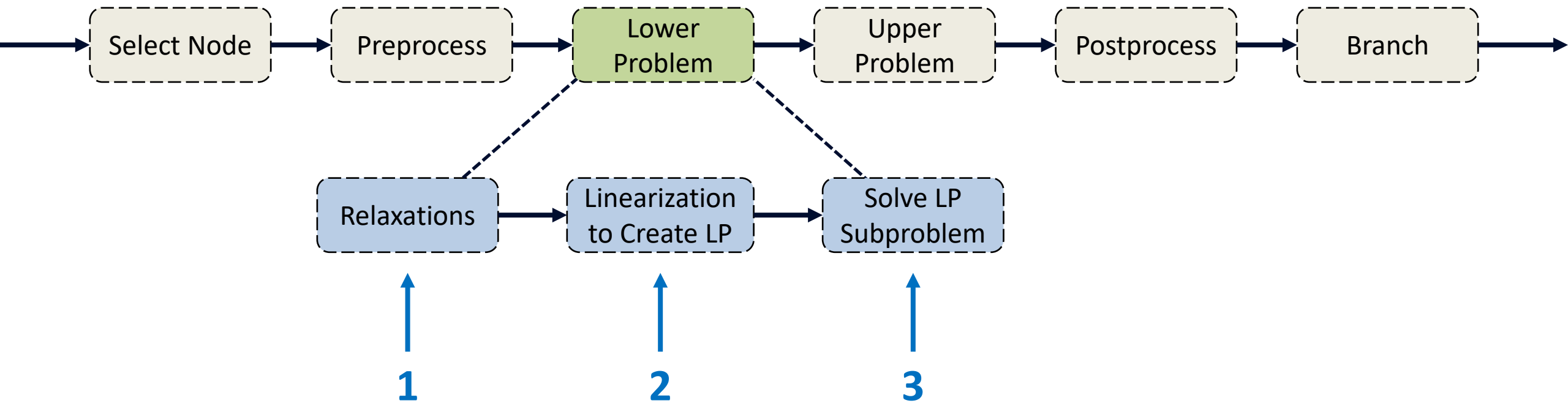
# Serial CPU Branch-and-Bound



using JuMP, EAGO, HiGHS

```
model = JuMP.Model(EAGO.Optimizer(SubSolvers(; r=HiGHS.Optimizer())))
```

# Tasks for GPU Parallelization



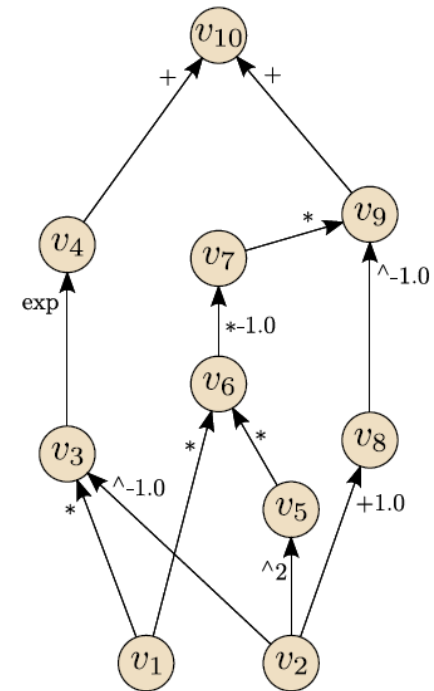
# McCormick Relaxations

- Provide convex underestimators of nonlinear functions over a specified domain
- Built recursively for factorable functions

$$\exp\left(x / y\right) - xy^2 / (y + 1)$$



$$\begin{aligned} v_1 &= x \\ v_2 &= y \\ v_3 &= v_1 / v_2 \\ v_4 &= \exp(v_3) \\ v_5 &= v_2^2 \\ v_6 &= v_1 v_5 \\ v_7 &= -v_6 \\ v_8 &= v_2 + 1.0 \\ v_9 &= v_7 / v_8 \\ v_{10} &= v_4 + v_9 \end{aligned}$$



[15] McCormick, G. Computability of global solutions to factorable nonconvex programs: Part I - Convex underestimating problems. *Mathematical Programming*. 10 (1): 147-175 (1976).

[16] Mitsos, A., et al. McCormick-based relaxations of algorithms. *SIAM Journal on Optimization*, SIAM. 20, 73-601 (2009).

[17] Scott, J. K., et al. Generalized McCormick relaxations. *Journal of Global Optimization*. 51(4): 569-606 (2011).

# McCormick.jl

```
using McCormick
```

```
x = 2.0
```

```
xIntv = interval(1.0, 4.0)
```

```
xMC = MC{1,NS}(x, xIntv, 1)
```

# McCormick.jl

```
using McCormick
```

```
x = 2.0
```

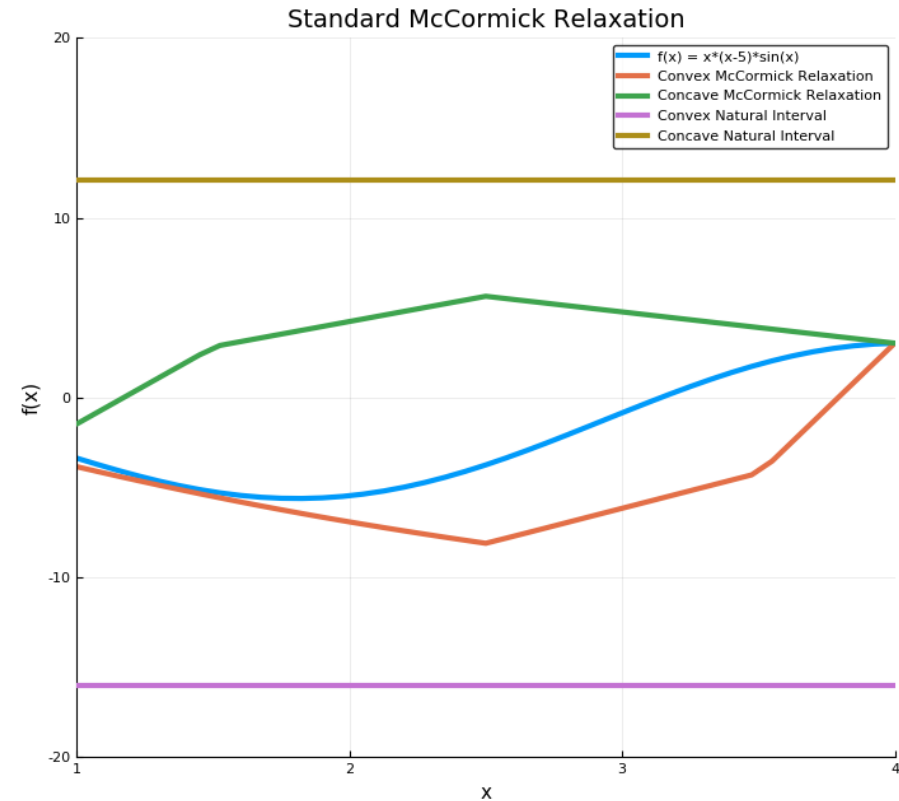
```
xIntv = interval(1.0, 4.0)
```

```
xMC = MC{1,NS}(x, xIntv, 1)
```

```
f(x) = x*(x - 5.0)*sin(x)
```

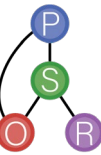
```
f(xIntv)
```

```
f(xMC)
```



# McCormick.jl

```
struct MC{N, T <: RelaxTag} <: Real
    "Convex relaxation"
    cv::Float64
    "Concave relaxation"
    cc::Float64
    "Interval bounds"
    Intv::IntervalArithmetic.Interval{Float64}
    "(Sub)gradient of convex relaxation"
    cv_grad::StaticArrays.SVector{N,Float64}
    "(Sub)gradient of concave relaxation"
    cc_grad::StaticArrays.SVector{N,Float64}
end
```



# McCormick.jl

$$x \in [x^L, x^U], y \in [y^L, y^U]$$

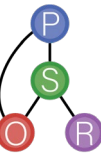
## Interval Arithmetic

$$x \cdot y = [\min\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}, \max\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}]$$

## McCormick

$$(x \cdot y)^{cv} = \max\{x^L y + xy^L - x^L y^L, x^U y + xy^U - x^U y^U\}$$

$$(x \cdot y)^{cc} = \min\{x^U y + xy^L - x^U y^L, x^L y + xy^U - x^L y^U\}$$



# McCormick.jl

$$x \in [x^L, x^U], y \in [y^L, y^U]$$

## Interval Arithmetic

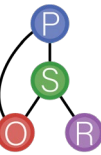
$$x \cdot y = [\min\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}, \max\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}]$$

## McCormick

$$(x \cdot y)^{cv} = \max\{x^L y + x y^L - x^L y^L, x^U y + x y^U - x^U y^U\}$$

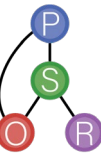
$$(x \cdot y)^{cc} = \min\{x^U y + x y^L - x^U y^L, x^L y + x y^U - x^L y^U\}$$

```
@inline function *(x1::MC{N,T}, x2::MC{N,T}) where {N, T<:Union{NS,MV}}  
    mult_kernel(x1, x2, x1.Intv*x2.Intv)  
end
```



# EAGO's Directed Acyclic Graph

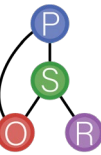
```
using JuMP, EAGO
model = JuMP.Model(EAGO.Optimizer)
@variables(model, begin
    0 <= x <= 1
    0 <= y <= 1
end)
@constraint(model, exp(x/y) - x*y^2/(y+1) <= 1)
```



# EAGO's Directed Acyclic Graph

```
using JuMP, EAGO
model = JuMP.Model(EAGO.Optimizer)
@variables(model, begin
    0 <= x <= 1
    0 <= y <= 1
end)
@constraint(model, exp(x/y) - x*y^2/(y+1) <= 1)
```

Node	Expression	Interval
1	-(2, 16)	[-1.0, ∞)
2	-(3, 7)	[0.0, ∞)
3	exp(4,)	[1.0, ∞)
4	/(5, 6)	[0.0, ∞)
5	x[1]	[0.0, 1.0]
6	x[2]	[0.0, 1.0]
7	/(8, 13)	[0.0, 1.0]
8	*(9, 10)	[0.0, 1.0]
9	x[1]	[0.0, 1.0]
10	^(11, 12)	[0.0, 1.0]
11	x[2]	[0.0, 1.0]
12	c[1]	2.0
13	+(14, 15)	[1.0, 2.0]
14	x[2]	[0.0, 1.0]
15	c[2]	1.0
16	c[3]	1.0



# EAGO's Directed Acyclic Graph

Node	Expression	Interval	Node	Expression	Interval	Node	Expression	Interval	Node	Expression	Interval
1	-(2, 16)	[-0.25, ∞)	1	-(2, 16)	[-0.215975, ∞)	1	-(2, 16)	[-0.101279, ∞)	1	-(2, 16)	[0.117, ∞)
2	-(3, 7)	[0.75, ∞)	2	-(3, 7)	[0.784025, ∞)	2	-(3, 7)	[0.898721, ∞)	2	-(3, 7)	[1.117, ∞)
3	exp(4,)	[1.0, ∞)	3	exp(4,)	[1.28402, ∞)	3	exp(4,)	[1.64872, ∞)	3	exp(4,)	[2.117, ∞)
4	/(5, 6)	[0.0, ∞)	4	/(5, 6)	[0.25, ∞)	4	/(5, 6)	[0.5, ∞)	4	/(5, 6)	[0.75, ∞)
5	x[1]	[0.0, 0.25]	5	x[1]	[0.25, 0.5]	5	x[1]	[0.5, 0.75]	5	x[1]	[0.75, 1.0]
6	x[2]	[0.0, 1.0]	6	x[2]	[0.0, 1.0]	6	x[2]	[0.0, 1.0]	6	x[2]	[0.0, 1.0]
7	/(8, 13)	[0.0, 0.25]	7	/(8, 13)	[0.0, 0.5]	7	/(8, 13)	[0.0, 0.75]	7	/(8, 13)	[0.0, 1.0]
8	*(9, 10)	[0.0, 0.25]	8	*(9, 10)	[0.0, 0.5]	8	*(9, 10)	[0.0, 0.75]	8	*(9, 10)	[0.0, 1.0]
9	x[1]	[0.0, 0.25]	9	x[1]	[0.25, 0.5]	9	x[1]	[0.5, 0.75]	9	x[1]	[0.75, 1.0]
10	^(11, 12)	[0.0, 1.0]	10	^(11, 12)	[0.0, 1.0]	10	^(11, 12)	[0.0, 1.0]	10	^(11, 12)	[0.0, 1.0]
11	x[2]	[0.0, 1.0]	11	x[2]	[0.0, 1.0]	11	x[2]	[0.0, 1.0]	11	x[2]	[0.0, 1.0]
12	c[1]	2.0	12	c[1]	2.0	12	c[1]	2.0	12	c[1]	2.0
13	+(14, 15)	[1.0, 2.0]	13	+(14, 15)	[1.0, 2.0]	13	+(14, 15)	[1.0, 2.0]	13	+(14, 15)	[1.0, 2.0]
14	x[2]	[0.0, 1.0]	14	x[2]	[0.0, 1.0]	14	x[2]	[0.0, 1.0]	14	x[2]	[0.0, 1.0]
15	c[2]	1.0	15	c[2]	1.0	15	c[2]	1.0	15	c[2]	1.0
16	c[3]	1.0	16	c[3]	1.0	16	c[3]	1.0	16	c[3]	1.0

$$x \in [0, 0.25]$$

$$y \in [0, 1]$$

$$x \in [0.25, 0.5]$$

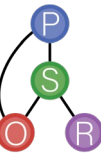
$$y \in [0, 1]$$

$$x \in [0.5, 0.75]$$

$$y \in [0, 1]$$

$$x \in [0.75, 1]$$

$$y \in [0, 1]$$



# SourceCodeMcCormick.jl

- Enables parallel evaluation of relaxations on GPUs
- Generates customized CUDA kernels

using SourceCodeMcCormick

```
Symbolics.@variables x, y
expr = exp(x/y) - x*y^2/(y+1)
func = kgen(expr)
```



```
# Kernel(s) generated for the expression: exp(x / y) + (-x*(y^2)) / (1 + y)
```

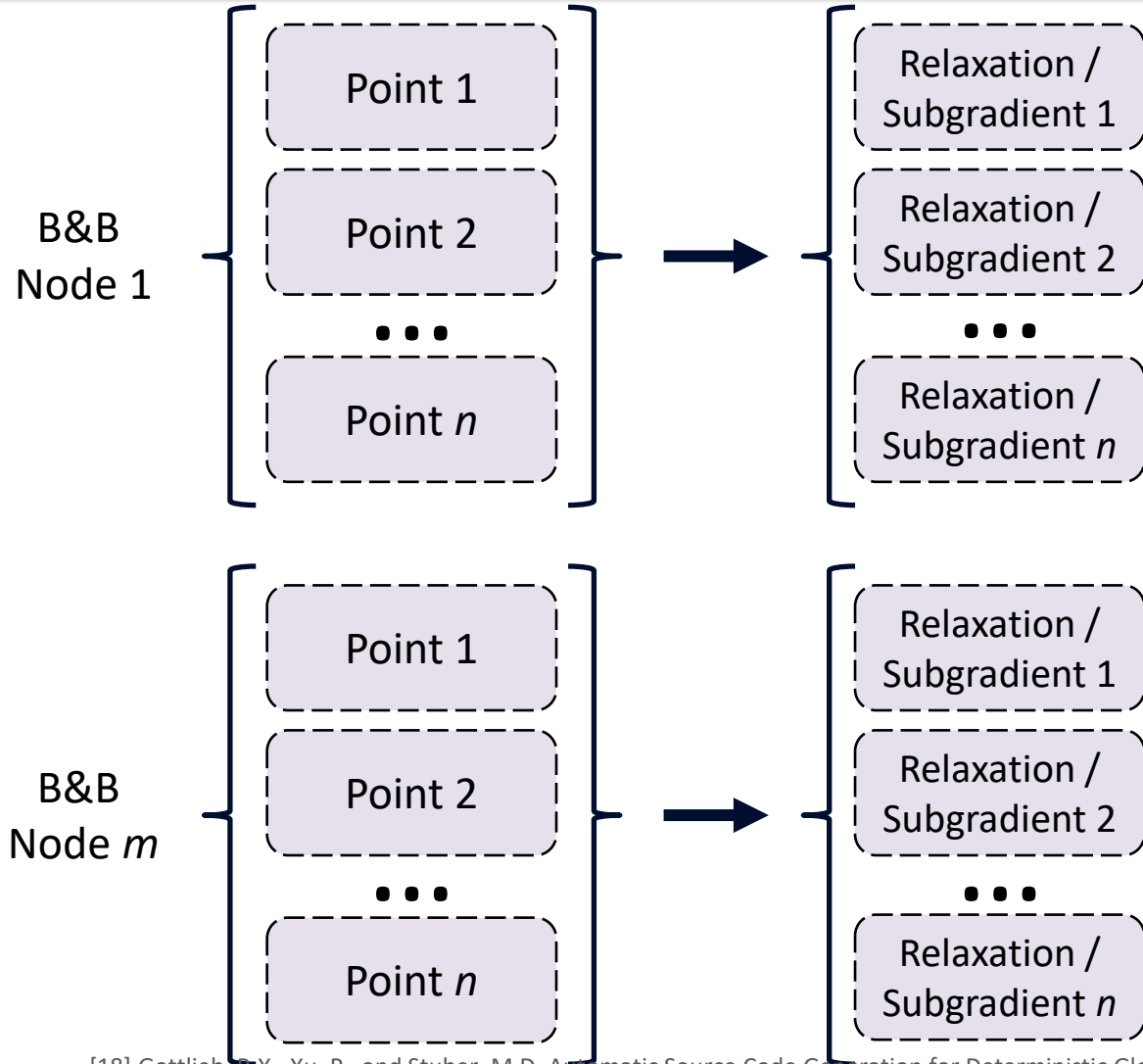
```
function f_5hzkcjneJVK_1(OUT, x, y)
    idx = threadIdx().x + (blockIdx().x - Int32(1)) * blockDim().x
    stride = blockDim().x * gridDim().x
    col = Int32(1)
    colmax = Int32(2)
    temp1_lo = 0.0
    temp1_hi = 0.0
    temp1_cv = 0.0
    temp1_cc = 0.0
    temp1_cvgrad = @MVector zeros(Float64, 2)
    temp1_ccgrad = @MVector zeros(Float64, 2)
    temp2_lo = 0.0
    temp2_hi = 0.0
    temp2_cv = 0.0
    temp2_cc = 0.0
    temp2_cvgrad = @MVector zeros(Float64, 2)
    temp2_ccgrad = @MVector zeros(Float64, 2)
    temp3_lo = 0.0
    temp3_hi = 0.0
    temp3_cv = 0.0
    temp3_cc = 0.0
    temp3_cvgrad = @MVector zeros(Float64, 2)
    temp3_ccgrad = @MVector zeros(Float64, 2)
    while idx <= Int32(size(OUT,1))
        #####
        ## Squared ##
        #####

        # Reset the column counter
        col = Int32(1)

        # Begin rule
        if y[idx,3] <= 0.0
            eps_min = y[idx,3]
```

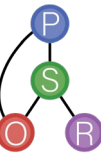


# SourceCodeMcCormick.jl

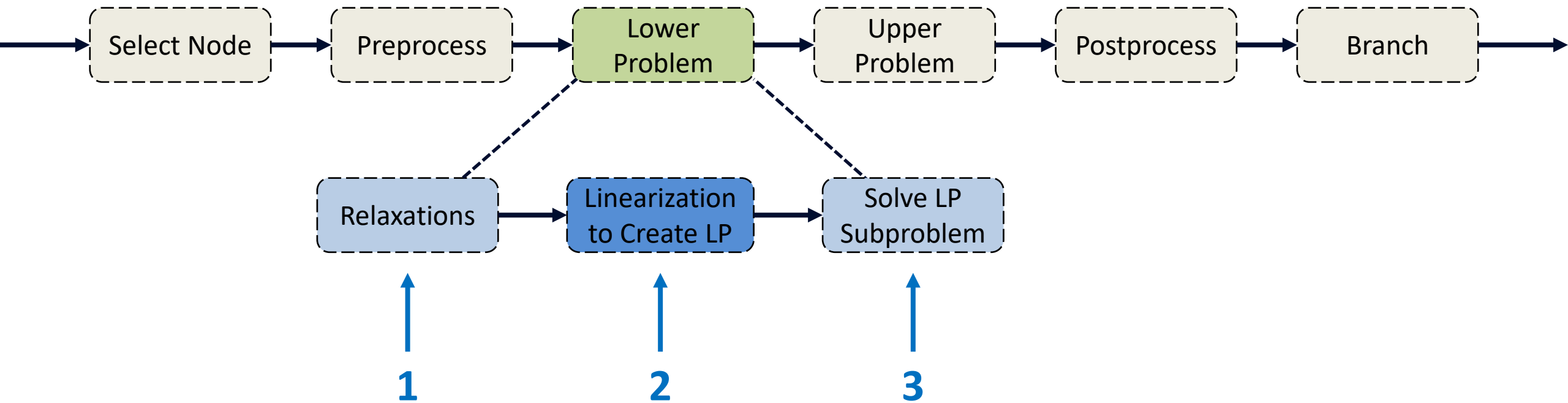


Expression Form	SCMC Avg. Evaluation Time (ns)	McCormick.jl Avg. Evaluation Time (ns)	Evaluation Time Speed-Up
$\sum_{i=1}^{10} x_i$	$1.9 \times 10^0$	$1.90 \times 10^2$	99.9x
$\exp(\sum_{i=1}^{10} x_i)$	$3.0 \times 10^0$	$2.72 \times 10^2$	90.6x
$(\sum_{i=1}^{10} x_i)^2$	$3.1 \times 10^0$	$2.36 \times 10^2$	76.2x
$\prod_{i=1}^{10} x_i$	$1.15 \times 10^1$	$5.79 \times 10^2$	50.3x
$\exp(\prod_{i=1}^{10} x_i)$	$1.19 \times 10^1$	$6.38 \times 10^2$	53.6x
$(\prod_{i=1}^{10} (x_i)^2)$	$1.02 \times 10^1$	$9.40 \times 10^2$	92.1x
$x_1 / (\prod_{i=2}^{10} x_i)$	$6.2 \times 10^0$	$5.11 \times 10^2$	82.4x
alkyl objective	$7. \times 10^{-1}$	$1.61 \times 10^2$	229.9x
ex6.2.10 objective	$9.04 \times 10^1$	$1.37 \times 10^4$	151.1x
arki0002 objective	$2.58 \times 10^3$	$1.62 \times 10^7$	6269.5x
Trained ANN	$1.90 \times 10^1$	$5.51 \times 10^3$	290.0x
Training ANN	$9.59 \times 10^1$	$1.28 \times 10^4$	133.4x

[18] Gottlieb, R.X., Xu, P., and Stuber, M.D. Automatic Source Code Generation for Deterministic Global Optimization with Parallel Architectures. Optimization Methods and Software. 1-39 (2024).

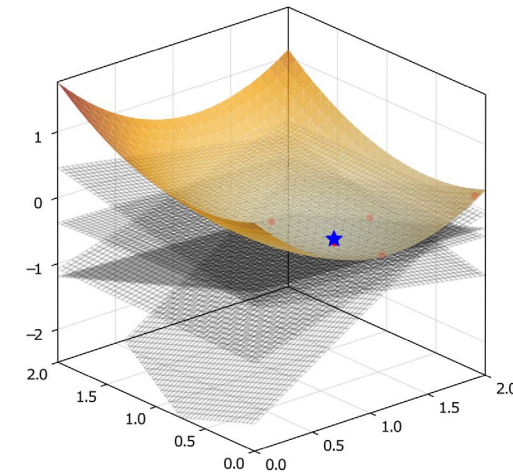
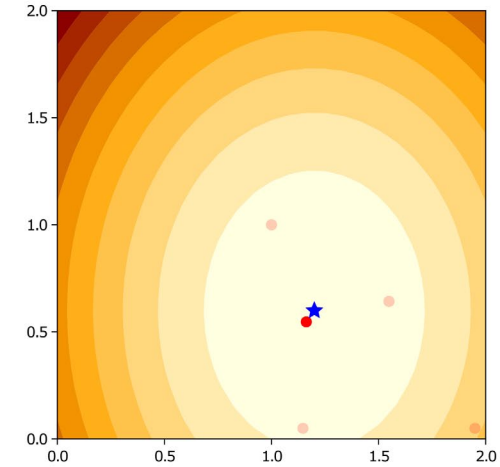


# Tasks for GPU Parallelization



# Kelley's Method

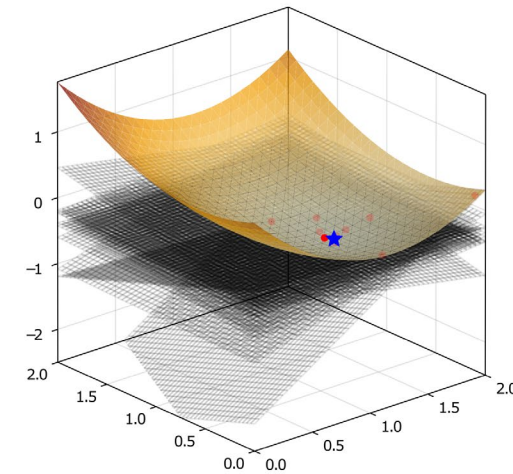
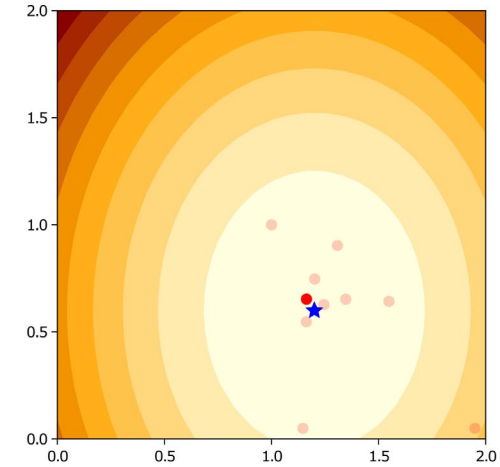
$n$	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001



[19] Kelley, J. E. The Cutting-Plane Method for Solving Convex Programs. Journal of the Society for Industrial and Applied Mathematics. 8(4): 703-712 (1960).

# Kelley's Method

$n$	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001
6	-0.3891
7	-0.3854
8	-0.3817
9	-0.3781
10	-0.3773



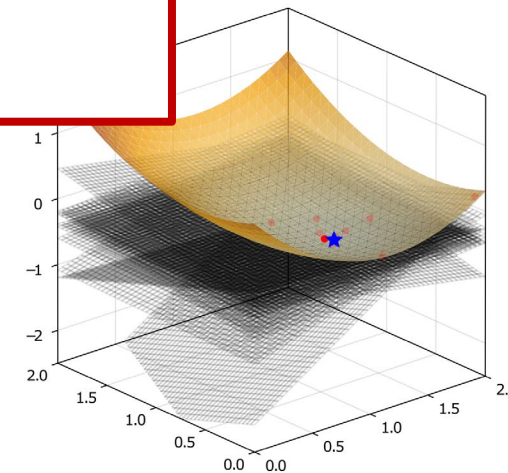
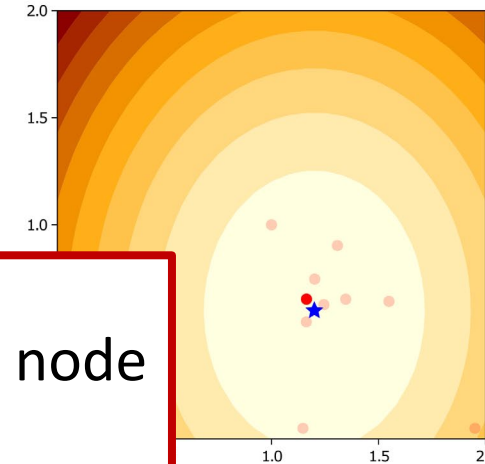
[19] Kelley, J. E. The Cutting-Plane Method for Solving Convex Programs. Journal of the Society for Industrial and Applied Mathematics. 8(4): 703-712 (1960).



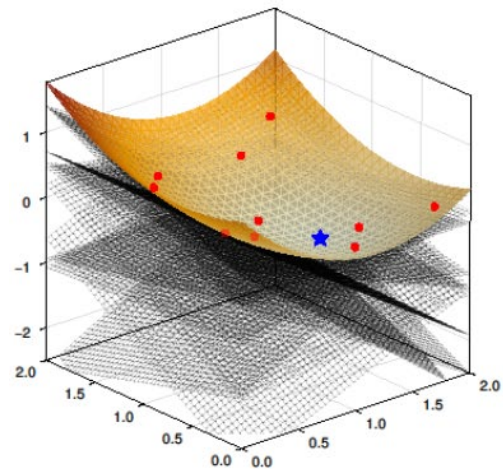
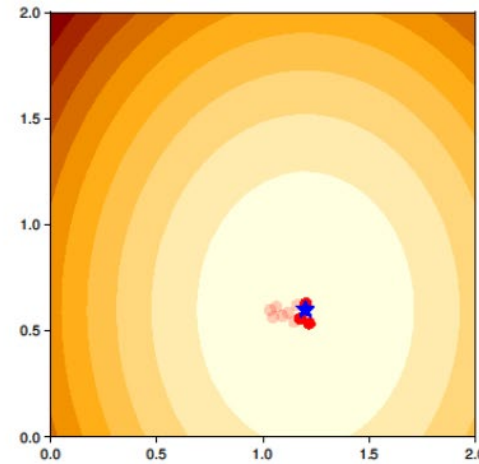
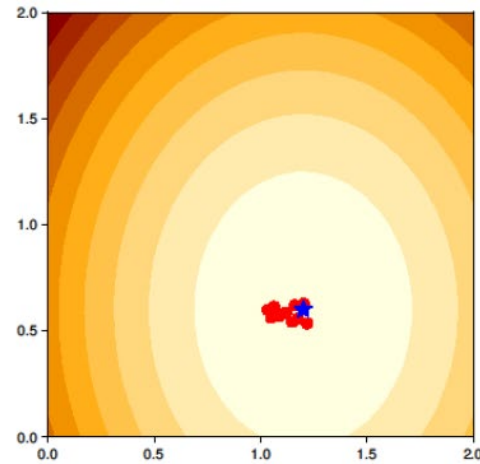
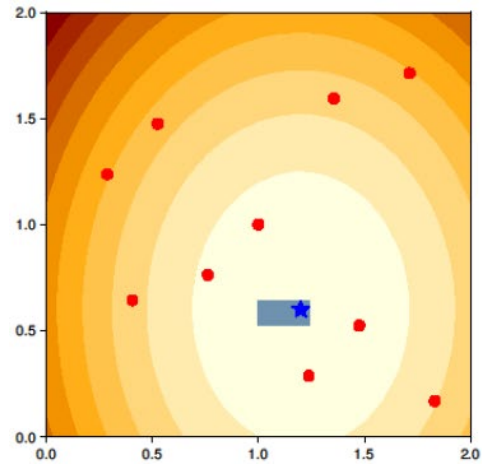
# Kelley's Method

$n$	Lower Bound
1	-0.9830
2	-0.7122
3	-0.5424
4	-0.4620
5	-0.4212
6	-0.3984
7	-0.3854
8	-0.3817
9	-0.3781
10	-0.3773

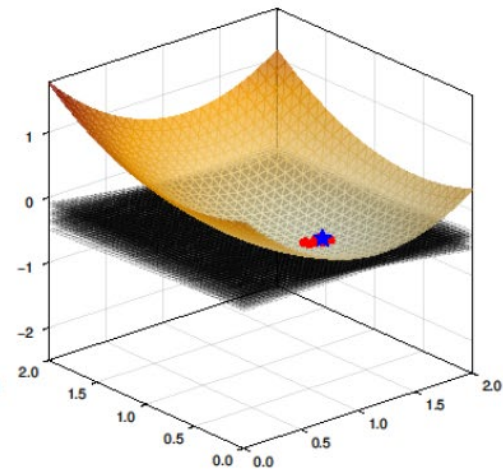
- Requires  $n$  LP solves per node
- Sequential



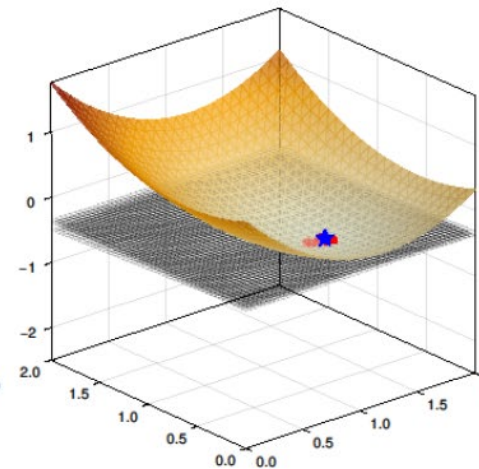
# Sobol Sampling



Lower Bound:  
-0.4506



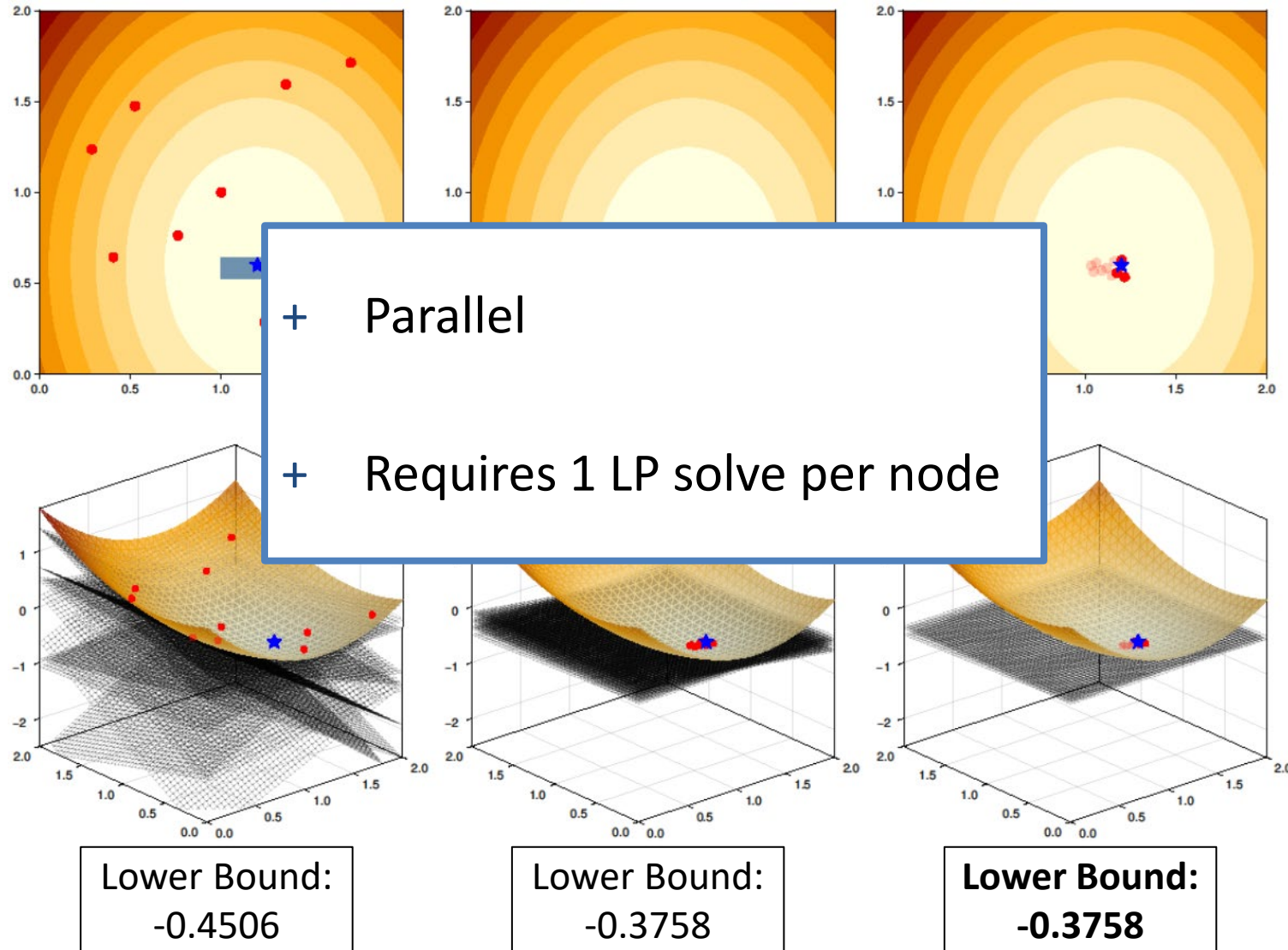
Lower Bound:  
-0.3758



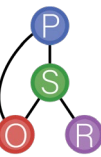
Lower Bound:  
-0.3758



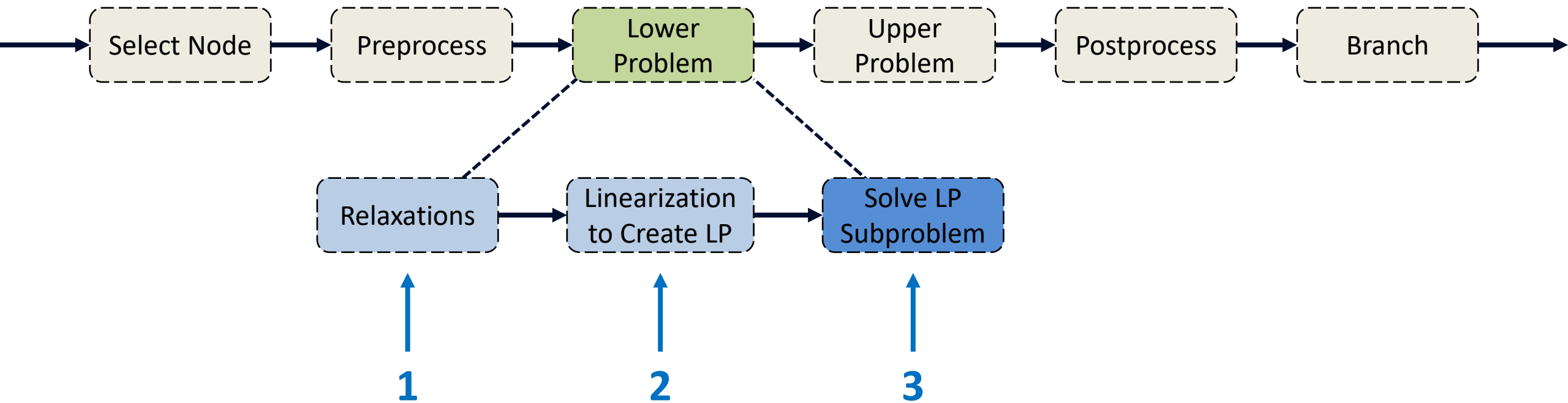
# Sobol Sampling



JuMP-dev 2020



# Tasks for GPU Parallelization



# PDLP

- Competitive first-order method for solving LPs

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & c^\top x \\ \text{subject to:} & Gx \geq h \\ & Ax = b \\ & l \leq x \leq u \end{array} \qquad \begin{array}{ll} \underset{y \in \mathbb{R}^{m_1+m_2}, \lambda \in \mathbb{R}^n}{\text{maximize}} & q^\top y + l^\top \lambda^+ - u^\top \lambda^- \\ \text{subject to:} & c - K^\top y = \lambda \\ & y_{1:m_1} \geq 0 \\ & \lambda \in \Lambda, \end{array}$$

$$\min_{x \in X} \max_{y \in Y} \mathcal{L}(x, y) := c^\top x - y^\top Kx + q^\top y$$

$$\begin{aligned} x^{k+1} &= \text{proj}_X(x^k - \tau(c - K^\top y^k)) \\ y^{k+1} &= \text{proj}_Y(y^k + \sigma(q - K(2x^{k+1} - x^k))) \end{aligned}$$

---

## Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient

---

David Applegate  
Google Research  
dapplagate@google.com

Mateo Díaz  
California Institute of Technology\*  
mateodd@caltech.edu

Oliver Hinder  
Google Research  
University of Pittsburgh  
ohinder@pitt.edu

Haihao Lu  
University of Chicago†  
haihao.lu@chicagobooth.edu

Miles Lubin  
Google Research  
mlubin@google.com

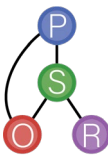
Brendan O'Donoghue  
DeepMind  
bodonoghue@deepmind.com

Warren Schudy  
Google Research  
wschudy@google.com

### Abstract

We present PDLP, a practical first-order method for linear programming (LP) that can solve to the high levels of accuracy that are expected in traditional LP applications. In addition, it can scale to very large problems because its core operation is matrix-vector multiplications. PDLP is derived by applying the primal-dual hybrid gradient (PDHG) method, popularized by Chambolle and Pock (2011), to a saddle-point formulation of LP. PDLP enhances PDHG for LP by combining several new techniques with older tricks from the literature; the enhancements include diagonal preconditioning, presolving, adaptive step sizes, and adaptive restarting. PDLP improves the state of the art for first-order methods applied to LP. We compare PDLP with SCS, an ADMM-based solver, on a set of 383 LP instances derived from MIPLIB 2017. With a target of  $10^{-8}$  relative accuracy and 1 hour time limit, PDLP achieves a 6.3x reduction in the geometric mean of solve times and a 4.6x reduction in the number of instances unsolved (from 227 to 49). Furthermore, we highlight standard benchmark instances and a large-scale application (PageRank) where our open-source prototype of PDLP, written in Julia, outperforms a commercial LP solver.

[20] Applegate, D., et al. Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient. 35<sup>th</sup> Conference on Neural Information Processing Systems (NeurIPS 2021). (2021).



# BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

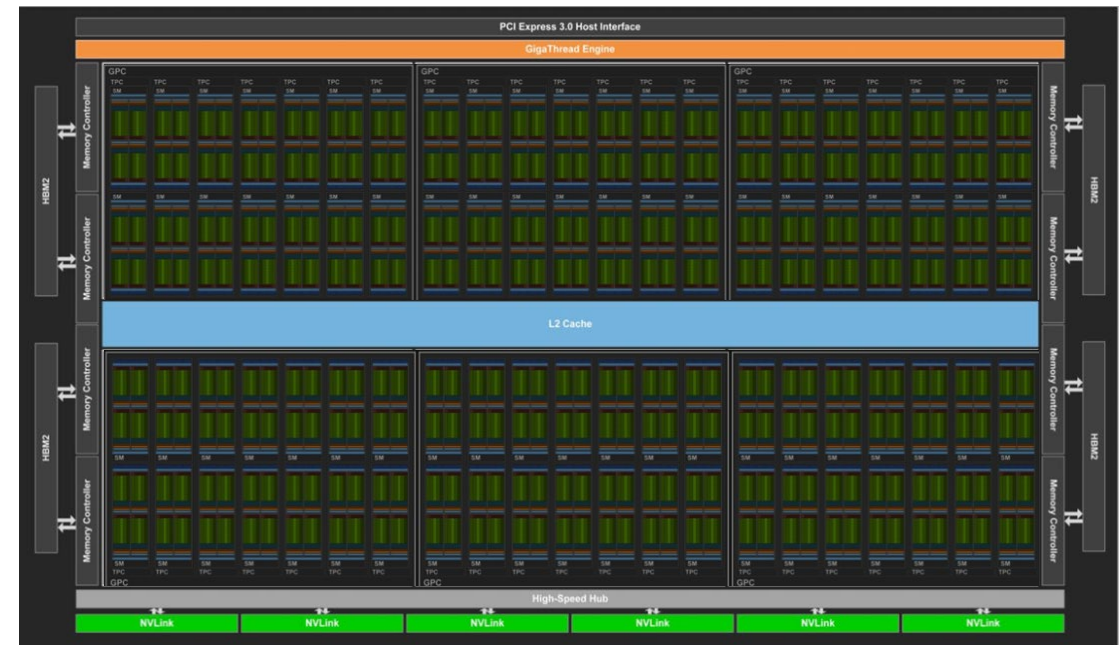
Algorithm 1: Single-Kernel PDLP

```

1 foreach LP do
2   Input: An initial solution  $z_i^{0,0}$ ;
3   Initialize outer loop counter  $n_i \leftarrow 0$ , total iterations  $k_i \leftarrow 0$ , step size
    $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$ , primal weight  $\omega_i^0 \leftarrow \text{InitializePrimalWeight}(c_i, q_i)$ ;
4   repeat
5      $t_i \leftarrow 0$ ;
6     repeat
7        $z_i^{n_i, t_i+1}, \eta_i^{n_i, t_i+1}, \hat{\eta}_i^{n_i, t_i+1} \leftarrow \text{AdaptivePDHGStep}(z_i^{n_i, t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i, t_i}, k_i)$ ;
8        $\bar{z}_i^{n_i, t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i, j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i, j} z_i^{n_i, j}$ ;
9        $z_{c,i}^{n_i, t_i+1} \leftarrow \text{GetRestartCandidate}(z_i^{n_i, t_i+1}, \bar{z}_i^{n_i, t_i+1}, \omega_i^{n_i})$ ;
10       $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$ ;
11    until restart or termination criteria holds;
12    Restart the outer loop.  $z_i^{n_i+1, 0} \leftarrow z_{c,i}^{n_i, t_i}, n_i \leftarrow n_i + 1$ ;
13     $\omega_i^{n_i} \leftarrow \text{PrimalWeightUpdate}(z_i^{n_i, 0}, z_i^{n_i-1, 0}, \omega_i^{n_i-1})$ ;
14  until termination criteria holds;
15  Output:  $z_i^{n_i, 0}$ .
16 end

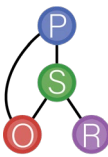
```

[14]



[14] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review



# BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

```

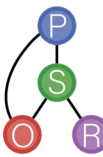
1 foreach LP do
2   Input: An initial solution  $z_i^{0,0}$ ;
3   Initialize outer loop counter  $n_i \leftarrow 0$ , total iterations  $k_i \leftarrow 0$ , step size
    $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$ , primal weight  $\omega_i^0 \leftarrow \text{InitializePrimalWeight}(c_i, q_i)$ ;
4   repeat
5      $t_i \leftarrow 0$ ;
6     repeat
7        $z_i^{n_i, t_i+1}, \eta_i^{n_i, t_i+1}, \hat{\eta}_i^{n_i, t_i+1} \leftarrow \text{AdaptivePDHGStep}(z_i^{n_i, t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i, t_i}, k_i)$ ;
8        $\bar{z}_i^{n_i, t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i, j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i, j} z_i^{n_i, j}$ ;
9        $z_{c,i}^{n_i, t_i+1} \leftarrow \text{GetRestartCandidate}(z_i^{n_i, t_i+1}, \bar{z}_i^{n_i, t_i+1}, \omega_i^{n_i})$ ;
10       $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$ ;
11    until restart or termination criteria holds;
12    Restart the outer loop.  $z_i^{n_i+1, 0} \leftarrow z_{c,i}^{n_i, t_i}, n_i \leftarrow n_i + 1$ ;
13     $\omega_i^{n_i} \leftarrow \text{PrimalWeightUpdate}(z_i^{n_i, 0}, z_i^{n_i-1, 0}, \omega_i^{n_i-1})$ ;
14  until termination criteria holds;
15  Output:  $z_i^{n_i, 0}$ .
16 end
  
```

[14]



[14] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review



# BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

```

1 foreach LP do
2   Input: An initial solution  $z_i^{0,0}$ ;
3   Initialize outer loop counter  $n_i \leftarrow 0$ , total iterations  $k_i \leftarrow 0$ , step size
    $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$ , primal weight  $\omega_i^0 \leftarrow \text{InitializePrimalWeight}(c_i, q_i)$ ;
4   repeat
5      $t_i \leftarrow 0$ ;
6     repeat
7        $z_i^{n_i, t_i+1}, \eta_i^{n_i, t_i+1}, \hat{\eta}_i^{n_i, t_i+1} \leftarrow \text{AdaptivePDHGStep}(z_i^{n_i, t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i, t_i}, k_i)$ ;
8        $\bar{z}_i^{n_i, t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i, j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i, j} z_i^{n_i, j}$ ;
9        $z_{c,i}^{n_i, t_i+1} \leftarrow \text{GetRestartCandidate}(z_i^{n_i, t_i+1}, \bar{z}_i^{n_i, t_i+1}, \omega_i^{n_i})$ ;
10       $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$ ;
11    until restart or termination criteria holds;
12    Restart the outer loop.  $z_i^{n_i+1, 0} \leftarrow z_{c,i}^{n_i, t_i}, n_i \leftarrow n_i + 1$ ;
13     $\omega_i^{n_i} \leftarrow \text{PrimalWeightUpdate}(z_i^{n_i, 0}, z_i^{n_i-1, 0}, \omega_i^{n_i-1})$ ;
14  until termination criteria holds;
15  Output:  $z_i^{n_i, 0}$ .
16 end

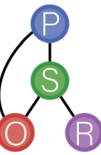
```



[14]

[14] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review



# BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

[14]

Algorithm 1: Single-Kernel PDLP

```

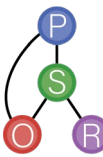
1 foreach LP do
2   Input: An initial solution  $z_i^{0,0}$ ;
3   Initialize outer loop counter  $n_i \leftarrow 0$ , total iterations  $k_i \leftarrow 0$ , step size
    $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$ , primal weight  $\omega_i^0 \leftarrow \text{InitializePrimalWeight}(c_i, q_i)$ ;
4   repeat
5      $t_i \leftarrow 0$ ;
6     repeat
7        $z_i^{n_i, t_i+1}, \eta_i^{n_i, t_i+1}, \hat{\eta}_i^{n_i, t_i+1} \leftarrow \text{AdaptivePDHGStep}(z_i^{n_i, t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i, t_i}, k_i)$ ;
8        $\bar{z}_i^{n_i, t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i, j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i, j} z_i^{n_i, j}$ ;
9        $z_{c,i}^{n_i, t_i+1} \leftarrow \text{GetRestartCandidate}(z_i^{n_i, t_i+1}, \bar{z}_i^{n_i, t_i+1}, \omega_i^{n_i})$ ;
10       $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$ ;
11    until restart or termination criteria holds;
12    Restart the outer loop.  $z_i^{n_i+1, 0} \leftarrow z_{c,i}^{n_i, t_i}, n_i \leftarrow n_i + 1$ ;
13     $\omega_i^{n_i} \leftarrow \text{PrimalWeightUpdate}(z_i^{n_i, 0}, z_i^{n_i-1, 0}, \omega_i^{n_i-1})$ .
14  until termination criteria holds;
15  Output:  $z_i^{n_i, 0}$ .
16 end
  
```

Other LPs continue  
even if one finishes



[14] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

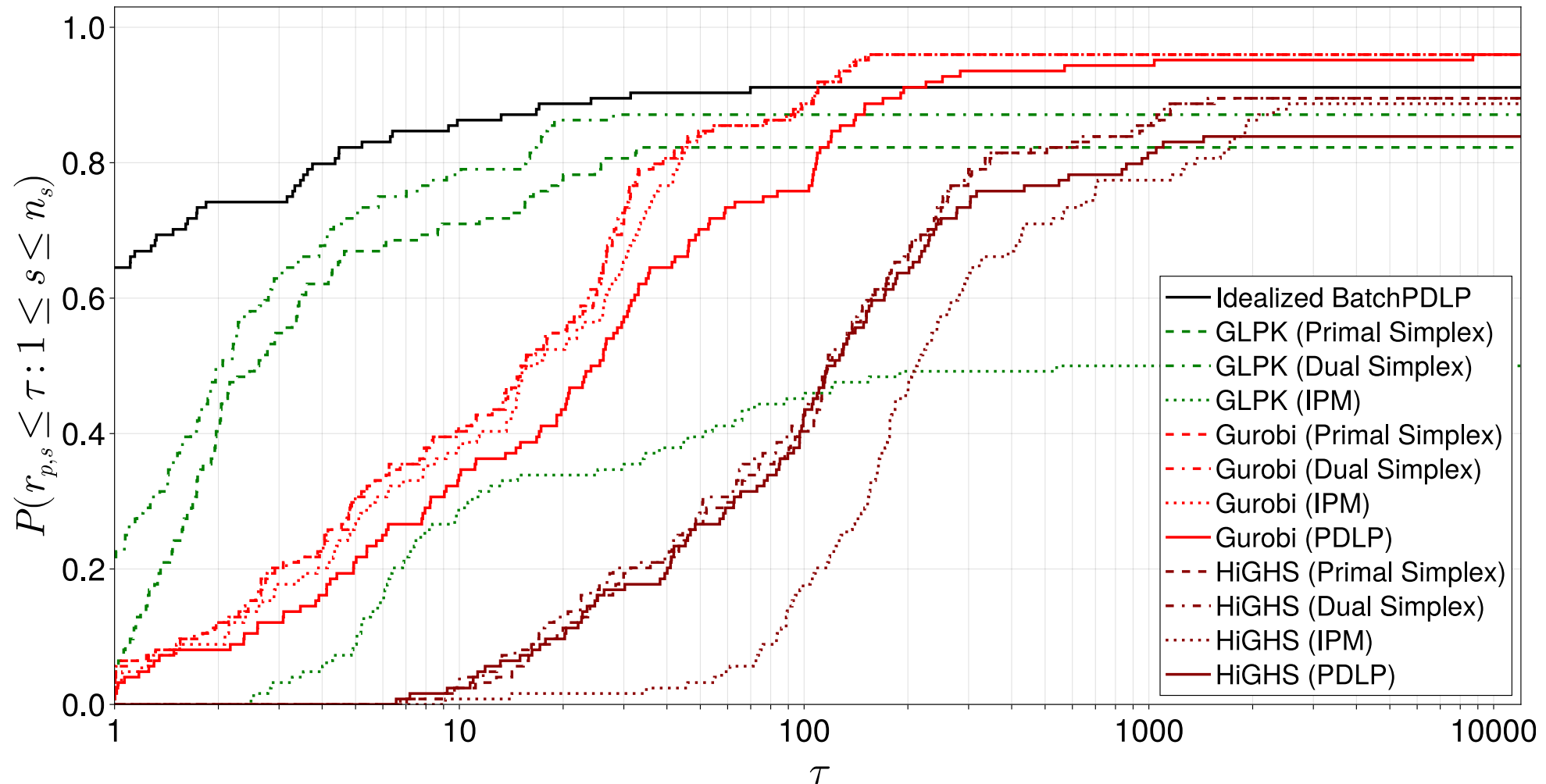
[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review



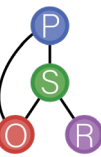
# BatchPDLP.jl Benchmarking

- 124 NLPs from MINLPLib
  - ≤ 100 variables
  - ≤ 1,000 constraints
  - Generate 10,000 LP subproblems for each

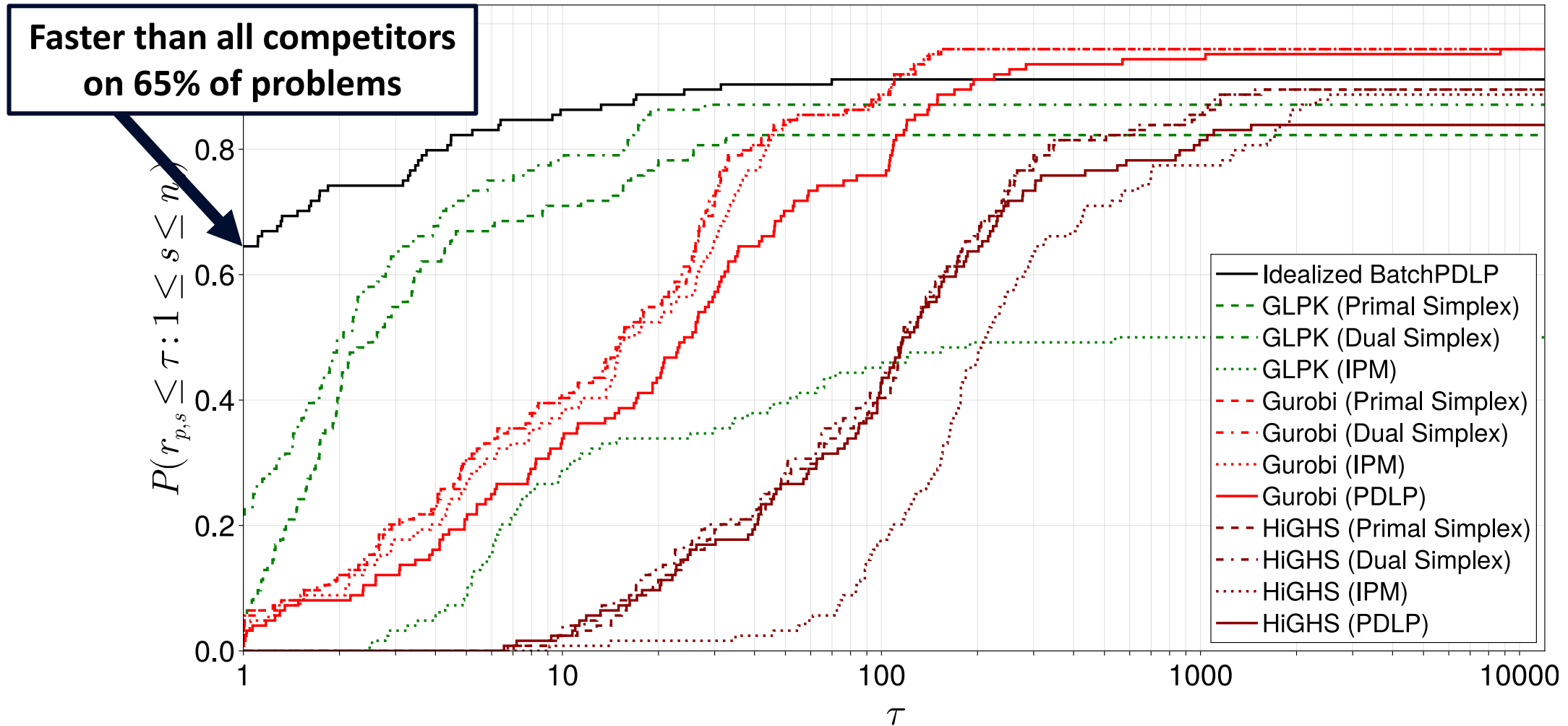
# BatchPDLP.jl Benchmarking



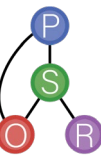
[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.



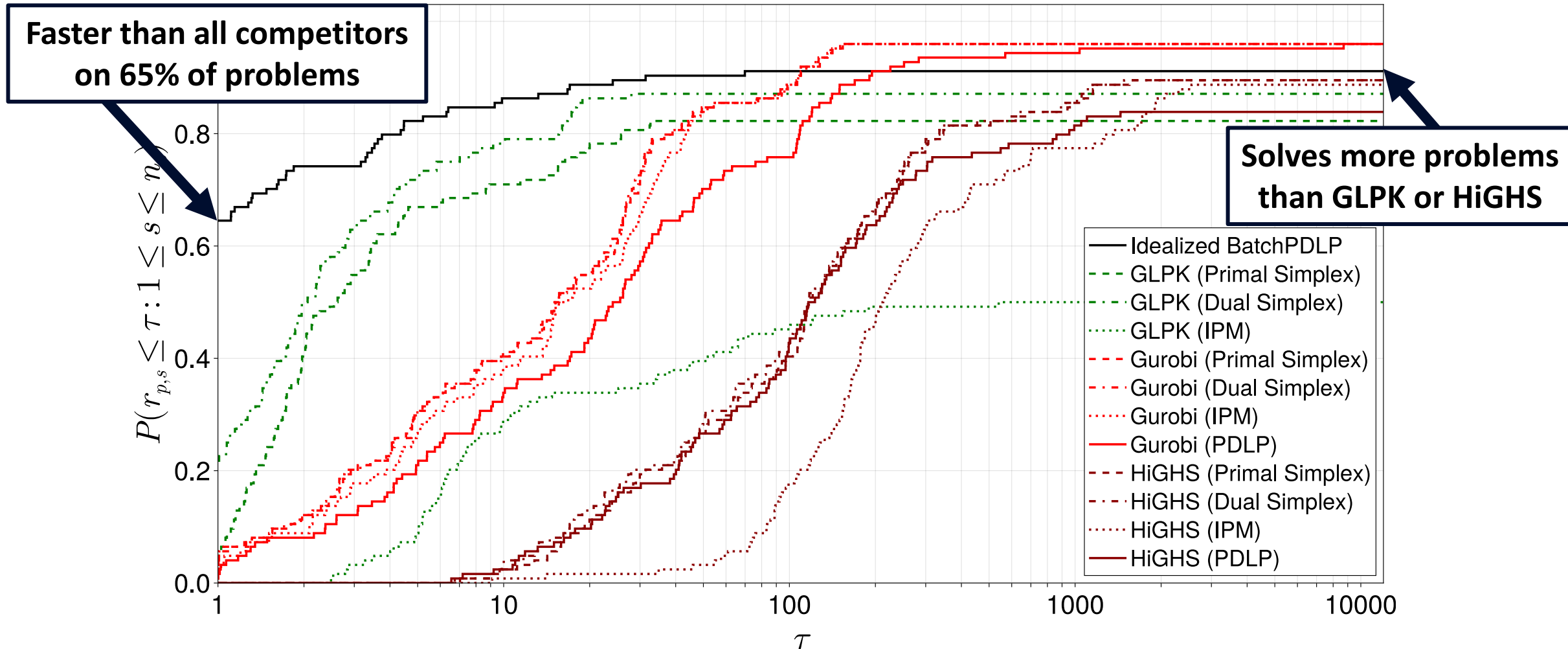
# BatchPDLP.jl Benchmarking



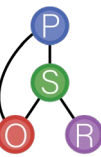
[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.



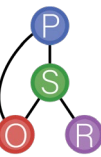
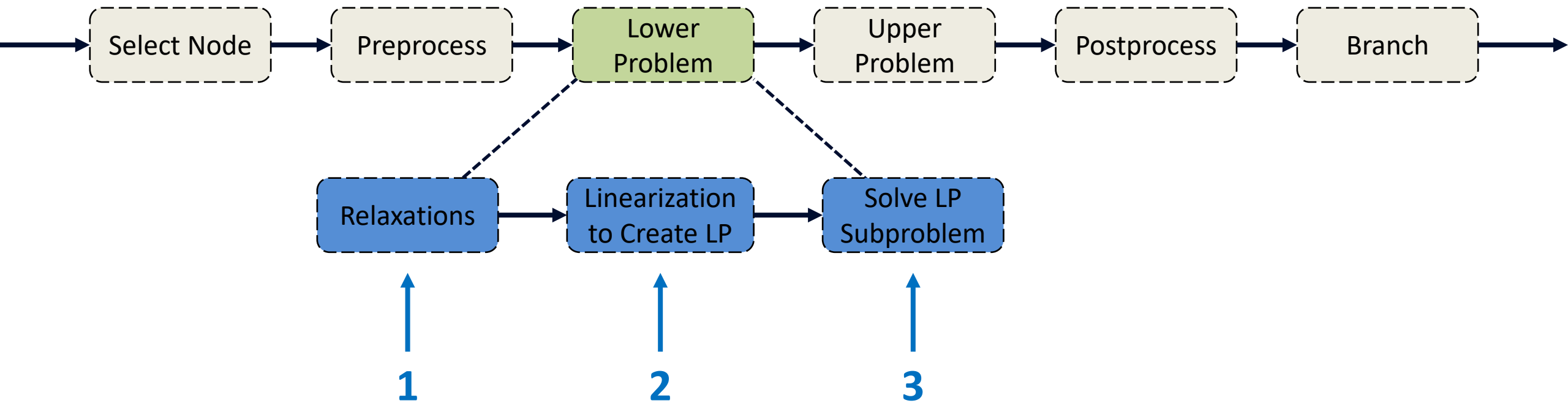
# BatchPDLP.jl Benchmarking



[21] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.



# Tasks for GPU Parallelization

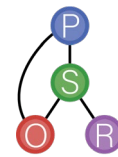


# ARION.jl

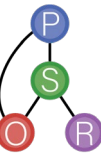
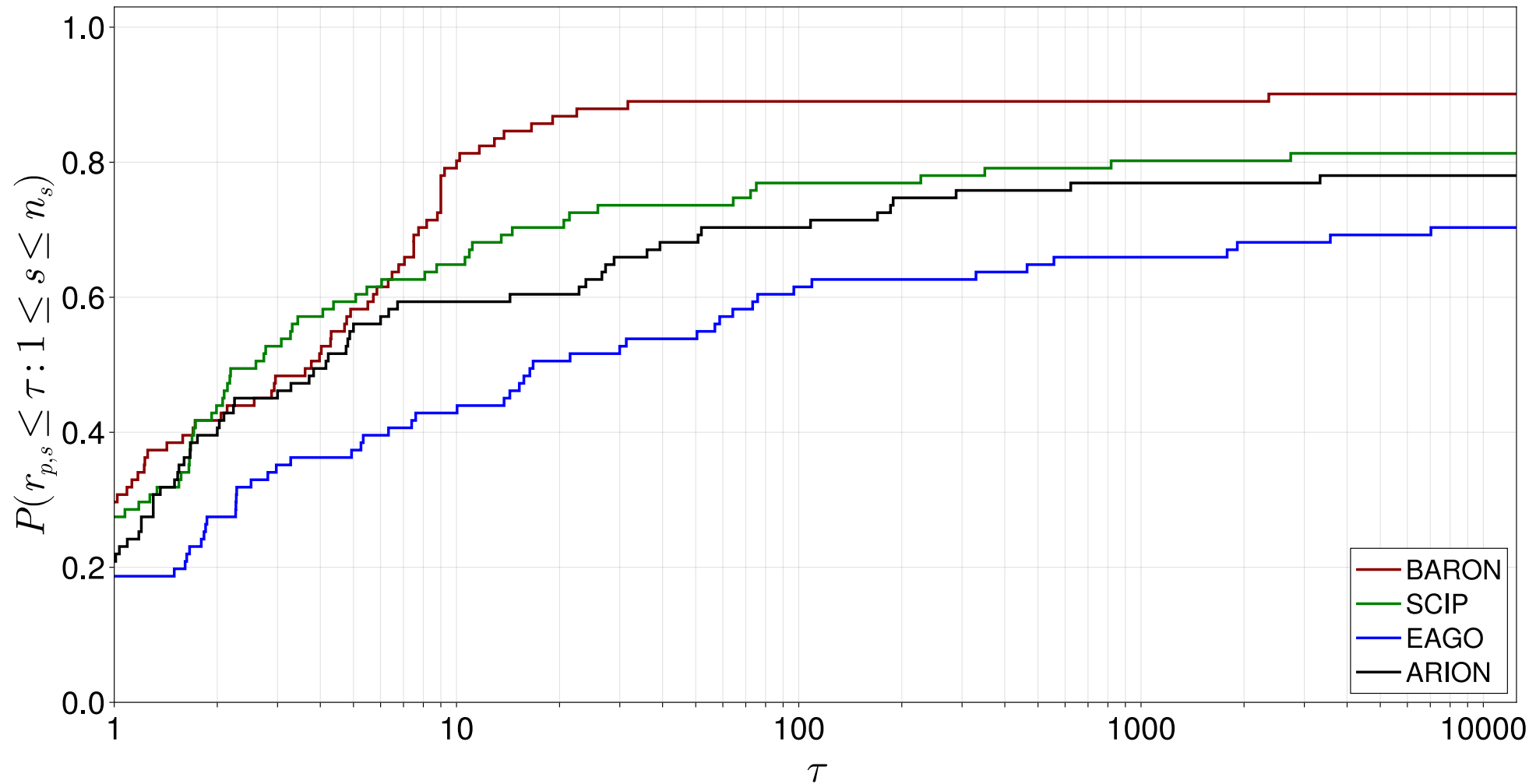
```
module ARION
    using CUDA, DocStringExtensions, LinearAlgebra, Sobol, MultiFloats, Printf
    using SourceCodeMcCormick, BatchPDL
    using EAGO, GLPK
    import EAGO: optimize_hook!
    import MathOptInterface as MOI

    include("./extension.jl")
    include("./kernels.jl")
    include("./subroutines.jl")

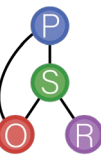
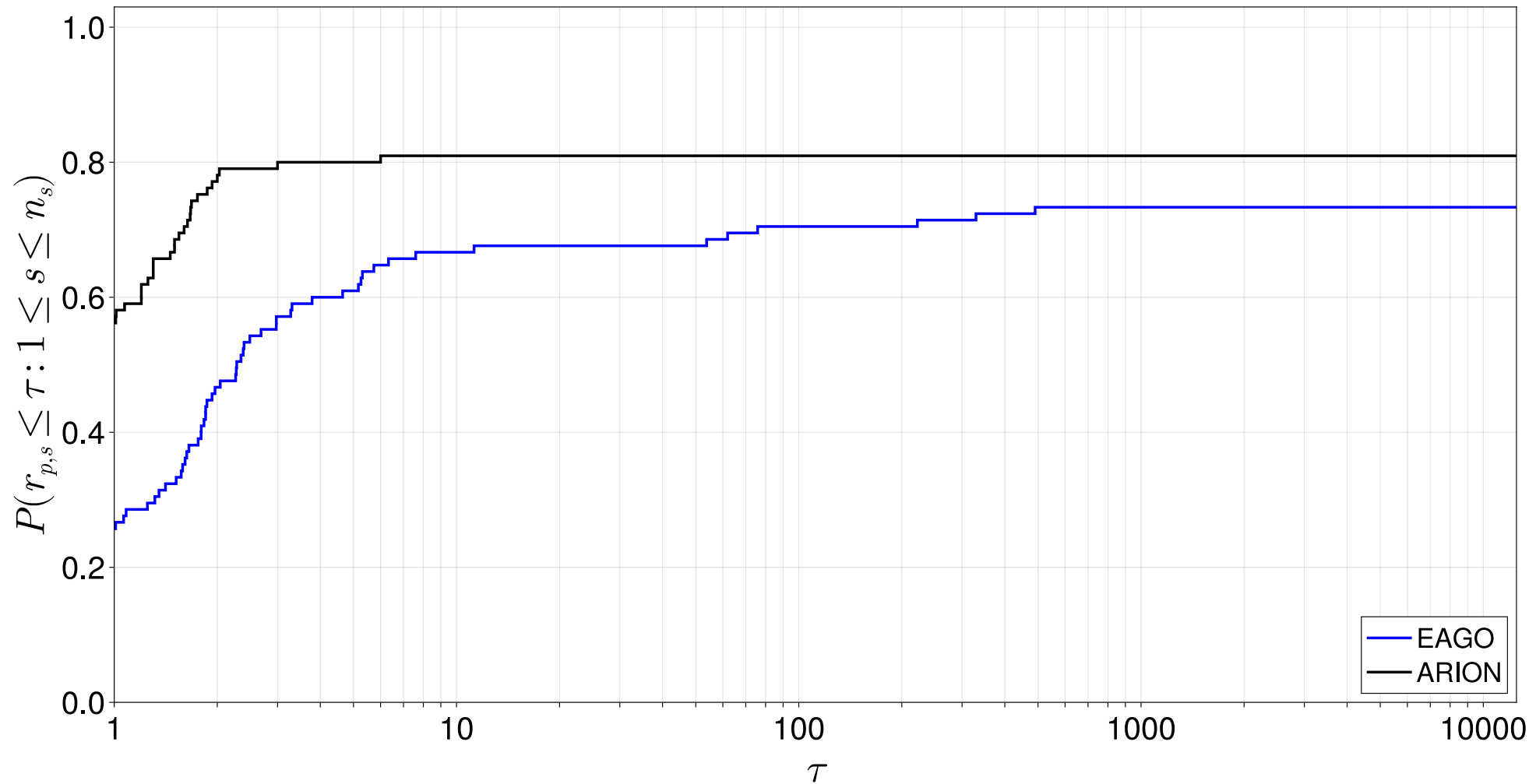
    export GroupMethod, GroupMethod_MultiGPU, KelleyMethod
    export Problem, LoadedProblem
end
```



# ARION.jl Benchmarking

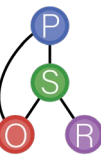


# ARION.jl Benchmarking



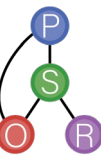
# Extending EAGO

```
Base.@kwdef mutable struct ExtendGPU <: EAGO.ExtensionType
    "Linearization point selection method. Options are {`:Sobol`, `:nSimplex`}."
    linearization_method::Symbol
    "A SCMC/kgen-generated function taking arguments `[OUT, x...]`"
    obj_fun::Function
    "Vector of kgen-generated functions representing EQ constraints in the original problem"
    eq_cons::Vector{Function}
    "Vector of kgen-generated functions representing LEQ constraints in the original problem"
    leq_cons::Vector{Function}
    "Vector of kgen-generated functions representing GEQ constraints in the original problem"
    geq_cons::Vector{Function}
    "Sparsity pattern for the objective function"
    obj_sp::Vector{Bool}
    "Sparsity patterns for the EQ constraints"
    eq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the LEQ constraints"
    leq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the GEQ constraints"
    geq_sp::Vector{Vector{Bool}}
    "The number of problem variables"
    n_vars::Int
    "The maximum number of nodes to consider simultaneously"
    max_parallel_nodes::Int64
    "Storage for inputs, outputs, buffers, and other intermediate fields needed during the PDLP algorithm"
    PDLP_data::BatchPDLP.PDLData
end
```



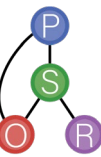
# Extending EAGO

```
Base.@kwdef mutable struct ExtendGPU <: EAGO.ExtensionType
    "Linearization point selection method. Options are {`:Sobol`, `:nSimplex`}."
    linearization_method::Symbol
    "A SCMC/kgen-generated function taking arguments `[OUT, x...]`"
    obj_fun::Function
    "Vector of kgen-generated functions representing EQ constraints in the original problem"
    eq_cons::Vector{Function}
    "Vector of kgen-generated functions representing LEQ constraints in the original problem"
    leq_cons::Vector{Function}
    "Vector of kgen-generated functions representing GEQ constraints in the original problem"
    geq_cons::Vector{Function}
    "Sparsity pattern for the objective function"
    obj_sp::Vector{Bool}
    "Sparsity patterns for the EQ constraints"
    eq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the LEQ constraints"
    leq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the GEQ constraints"
    geq_sp::Vector{Vector{Bool}}
    "The number of problem variables"
    n_vars::Int
    "The maximum number of nodes to consider simultaneously"
    max_parallel_nodes::Int64
    "Storage for inputs, outputs, buffers, and other intermediate fields needed during the PDLP algorithm"
    PDLP_data::BatchPDLP.PDLData
end
```



# Extending EAGO

```
Base.@kwdef mutable struct ExtendGPU <: EAGO.ExtensionType
    "Linearization point selection method. Options are {`:Sobol`, `:nSimplex`}."
    linearization_method::Symbol
    "A SCMC/kgen-generated function taking arguments `[OUT, x...]`"
    obj_fun::Function
    "Vector of kgen-generated functions representing EQ constraints in the original problem"
    eq_cons::Vector{Function}
    "Vector of kgen-generated functions representing LEQ constraints in the original problem"
    leq_cons::Vector{Function}
    "Vector of kgen-generated functions representing GEQ constraints in the original problem"
    geq_cons::Vector{Function}
    "Sparsity pattern for the objective function"
    obj_sp::Vector{Bool}
    "Sparsity patterns for the EQ constraints"
    eq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the LEQ constraints"
    leq_sp::Vector{Vector{Bool}}
    "Sparsity patterns for the GEQ constraints"
    geq_sp::Vector{Vector{Bool}}
    "The number of problem variables"
    n_vars::Int
    "The maximum number of nodes to consider simultaneously"
    max_parallel_nodes::Int64
    "Storage for inputs, outputs, buffers, and other intermediate fields needed during the PDLP algorithm"
    PDLP_data::BatchPDLP.PDLData
```



# Extending EAGO

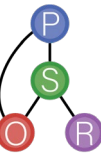
```
optimize_hook!(t::EAGO.ExtensionType, m::EAGO.Optimizer) = nothing

function throw_optimize_hook!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}
    optimize_hook!(_ext(m), m)
end

function MOI.optimize!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}

    m._global_optimizer._start_time = time()
    # Runs the branch and bound routine
    if !m.enable_optimize_hook
        initial_parse!(m)
        parse_classify_problem!(m._global_optimizer)
        m._global_optimizer._parse_time = m._global_optimizer._start_time - time()
        optimize!(m._working_problem._problem_type, m)
    else
        throw_optimize_hook!(m)
    end

    return
end
```

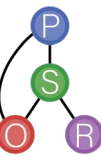


# Extending EAGO

```
optimize_hook!(t::EAGO.ExtensionType, m::EAGO.Optimizer) = nothing
```

```
function throw_optimize_hook!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}
|   optimize_hook!(_ext(m), m)
end
```

```
function MOI.optimize!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}
|
|   m._global_optimizer._start_time = time()
|   # Runs the branch and bound routine
|   if !m.enable_optimize_hook
|       initial_parse!(m)
|       parse_classify_problem!(m._global_optimizer)
|       m._global_optimizer._parse_time = m._global_optimizer._start_time - time()
|       optimize!(m._working_problem._problem_type, m)
|   else
|       throw_optimize_hook!(m)
|   end
|
|   return
end
```

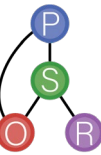


# Extending EAGO

```
optimize_hook!(t::EAGO.ExtensionType, m::EAGO.Optimizer) = nothing
```

```
function throw_optimize_hook!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}  
    optimize_hook!(_ext(m), m)  
end
```

```
function MOI.optimize!(m::EAGO.Optimizer{Q,S,T}) where {Q,S,T}  
  
    m._global_optimizer._start_time = time()  
    # Runs the branch and bound routine  
    if !m.enable_optimize_hook  
        initial_parse!(m)  
        parse_classify_problem!(m._global_optimizer)  
        m._global_optimizer._parse_time = m._global_optimizer._start_time - time()  
        optimize!(m._working_problem._problem_type, m)  
    else  
        throw_optimize_hook!(m)  
    end  
  
    return  
end
```



# Extending EAGO

```
using JuMP, EAGO
```

```
import EAGO.optimize_hook!
```

```
function EAGO.optimize_hook!(t::ExtendGPU, m::EAGO.Optimizer)
```

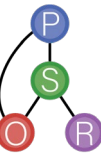
```
    EAGO.initial_parse!(m)
```

```
    optimize_gpu!(t, m)
```

```
end
```

```
> function optimize_gpu!(t::ExtendGPU, m::EAGO.Optimizer{R,S,Q}) where {R,S,Q<:EAGO.ExtensionType} ...
```

```
end
```



# Extending EAGO

```
using JuMP, EAGO
```

```
import EAGO.optimize_hook!
```

```
function EAGO.optimize_hook!(t::ExtendGPU, m::EAGO.Optimizer)
```

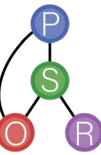
```
    EAGO.initial_parse!(m)
```

```
    optimize_gpu!(t, m)
```

```
end
```

```
> function optimize_gpu!(t::ExtendGPU, m::EAGO.Optimizer{R,S,Q}) where {R,S,Q<:EAGO.ExtensionType} ...
```

```
end
```

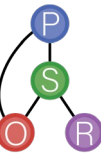


# Extending EAGO

```
using SourceCodeMcCormick, JuMP, EAGO

Symbolics.@variables x, y
expr = exp(x/y) - x*y^2/(y+1)
func = kgen(expr)

model = JuMP.Model(EAGO.Optimizer(SubSolvers(; t=ExtendGPU(func))))
JuMP.@variables(model, begin
    0 <= x <= 1
    0 <= y <= 1
end)
@constraint(model, exp(x/y) - x*y^2/(y+1) <= 1)
JuMP.optimize!(model)
```



# Extending EAGO

```
using SourceCodeMcCormick, JuMP, EAGO
```

```
Symbolics.@variables x, y  
expr = exp(x/y) - x*y^2/(y+1)  
func = kgen(expr)
```

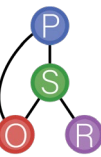
```
model = JuMP.Model(EAGO.Optimizer(SubSolvers(; t=ExtendGPU(func))))  
JuMP.@variables(model, begin  
    0 <= x <= 1  
    0 <= y <= 1  
end)  
@constraint(model, exp(x/y) - x*y^2/(y+1) <= 1)  
JuMP.optimize!(model)
```

# ARION.jl

```
module ARION
    using CUDA, DocStringExtensions, LinearAlgebra, Sobol, MultiFloats, Printf
    using SourceCodeMcCormick, BatchPDL
    using EAGO, GLPK
    import EAGO: optimize_hook!
    import MathOptInterface as MOI

    include("./extension.jl")
    include("./kernels.jl")
    include("./subroutines.jl")

    export GroupMethod, GroupMethod_MultiGPU, KelleyMethod
    export Problem, LoadedProblem
end
```



# Future Work

- Make ARION an option in EAGO

```
using JuMP, EAGO
```

```
model = JuMP.Model(EAGO.Optimizer)  
set_attribute(model, "use_gpu", true)
```

- Update SourceCodeMcCormick to use MathOptInterface

```
using SourceCodeMcCormick
```

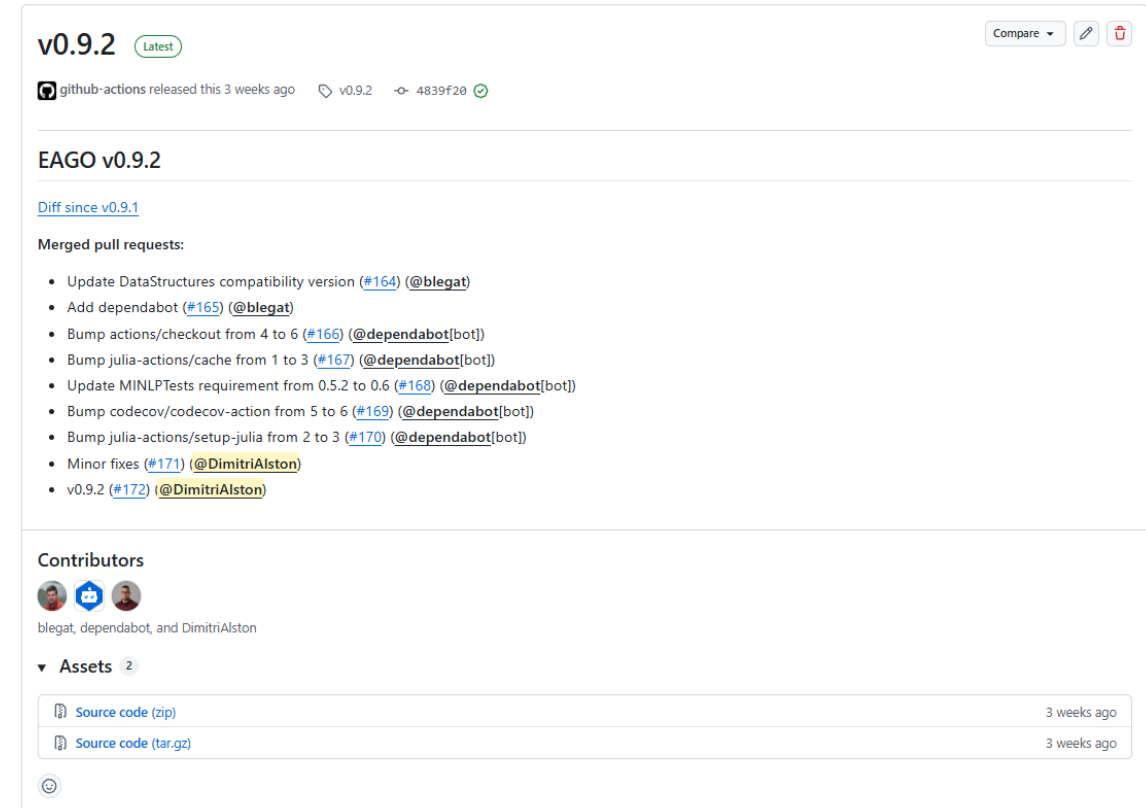
```
Symbolics.@variables x, y  
expr = exp(x/y) - x*y^2/(y+1)  
func = kgen(expr)
```

- Extend ARION to all GPU types

# Current State of EAGO

## EAGO v0.9.x

- Updated domain reduction routines
  - Constraint propagation
  - Optimality-based bounds-tightening
  - Feasibility-based bounds-tightening



v0.9.2 Latest Compare ✎ 🗑️

github-actions released this 3 weeks ago 🔗 v0.9.2 ↔️ 4839f20 🔒


### EAGO v0.9.2

[Diff since v0.9.1](#)



Merged pull requests:


- Update DataStructures compatibility version (#164) (@blegat)
- Add dependabot (#165) (@dependabot)
- Bump actions/checkout from 4 to 6 (#166) (@dependabot[bot])
- Bump julia-actions/cache from 1 to 3 (#167) (@dependabot[bot])
- Update MINLPTests requirement from 0.5.2 to 0.6 (#168) (@dependabot[bot])
- Bump codecov/codecov-action from 5 to 6 (#169) (@dependabot[bot])
- Bump julia-actions/setup-julia from 2 to 3 (#170) (@dependabot[bot])
- Minor fixes (#171) (@DimitriAlston)
- v0.9.2 (#172) (@DimitriAlston)

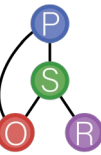
### Contributors

 blegat, dependabot, and DimitriAlston

### Assets 2

-  [Source code \(zip\)](#) 3 weeks ago
-  [Source code \(tar.gz\)](#) 3 weeks ago





# Current State of EAGO

## EAGO v0.9.x

@constraint(model,  $\exp(x/y) - x*y^2/(y+1) \leq 1$ )

Node	Expression	Interval
1	-(2, 16)	[-1.0, ∞)
2	-(3, 7)	[0.0, ∞)
3	exp(4,)	[1.0, ∞)
4	/(5, 6)	[0.0, ∞)
5	x[1]	[0.0, 1.0]
6	x[2]	[0.0, 1.0]
7	/(8, 13)	[0.0, 1.0]
8	*(9, 10)	[0.0, 1.0]
9	x[1]	[0.0, 1.0]
10	^(11, 12)	[0.0, 1.0]
11	x[2]	[0.0, 1.0]
12	c[1]	2.0
13	+(14, 15)	[1.0, 2.0]
14	x[2]	[0.0, 1.0]
15	c[2]	1.0
16	c[3]	1.0



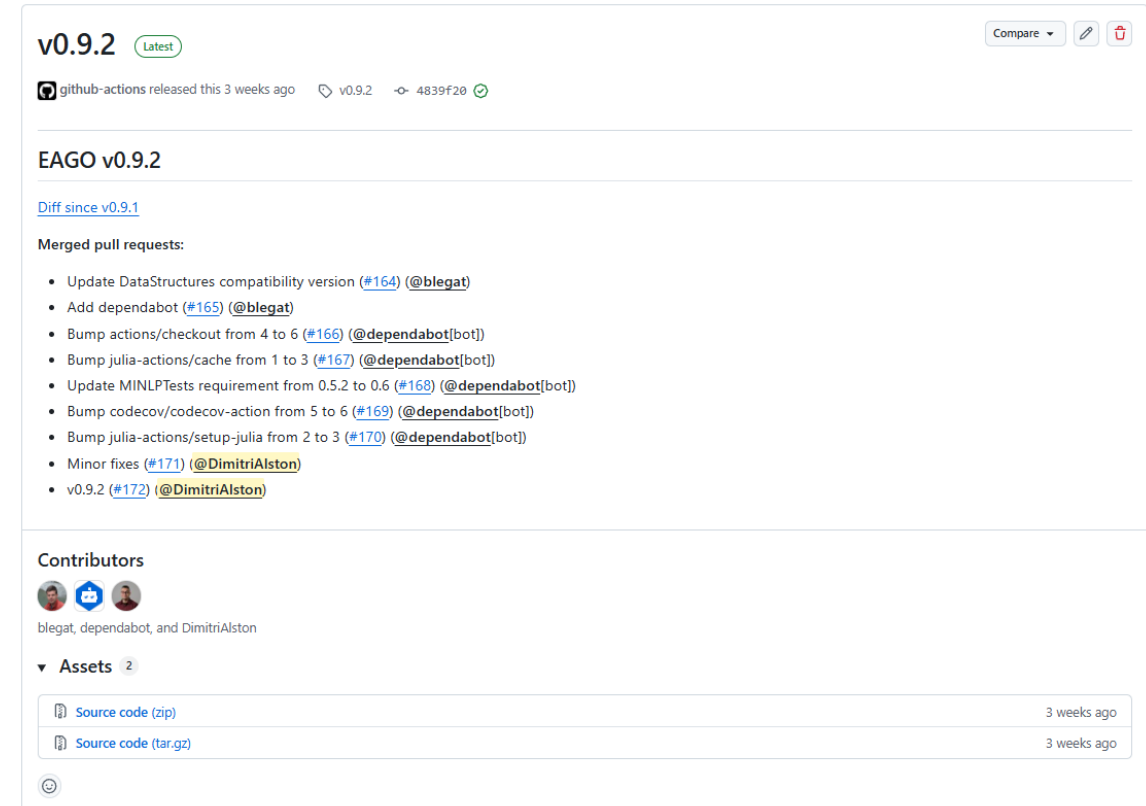
Node	Expression	Interval
1	-(2, 16)	[-0.693147, ∞)
2	-(3, 7)	[0.306853, ∞)
3	exp(4,)	[1.0, ∞)
4	/(5, 6)	[0.0, ∞)
5	x[1]	[0.0, 0.693147]
6	x[2]	[0.0, 1.0]
7	/(8, 13)	[0.0, 0.693147]
8	*(9, 10)	[0.0, 0.693147]
9	x[1]	[0.0, 0.693147]
10	^(11, 12)	[0.0, 1.0]
11	x[2]	[0.0, 1.0]
12	c[1]	2.0
13	+(14, 15)	[1.0, 2.0]
14	x[2]	[0.0, 1.0]
15	c[2]	1.0
16	c[3]	1.0

# Current State of EAGO

## EAGO v0.9.x

- Updated domain reduction routines
  - Constraint propagation
  - Optimality-based bounds-tightening
  - Feasibility-based bounds-tightening

## EAGO v0.10.x



v0.9.2 Latest Compare 📄 🗑️

github-actions released this 3 weeks ago 🔗 v0.9.2 🔗 4839f20 🔗


### EAGO v0.9.2

[Diff since v0.9.1](#)



Merged pull requests:

- Update DataStructures compatibility version (#164) (@blegat)
- Add dependabot (#165) (@dependabot)
- Bump actions/checkout from 4 to 6 (#166) (@dependabot[bot])
- Bump julia-actions/cache from 1 to 3 (#167) (@dependabot[bot])
- Update MINLPTests requirement from 0.5.2 to 0.6 (#168) (@dependabot[bot])
- Bump codecov/codecov-action from 5 to 6 (#169) (@dependabot[bot])
- Bump julia-actions/setup-julia from 2 to 3 (#170) (@dependabot[bot])
- Minor fixes (#171) (@DimitriAlston)
- v0.9.2 (#172) (@DimitriAlston)

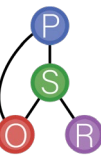
### Contributors

 blegat, dependabot, and DimitriAlston

### Assets 2

-  [Source code \(zip\)](#) 3 weeks ago
-  [Source code \(tar.gz\)](#) 3 weeks ago

©



# Acknowledgements

Members of the Process Systems and Operations Research Laboratory at the University of Connecticut (<https://psor.uconn.edu>)



Pratt & Whitney

Institute for Advanced Systems Engineering

UNIVERSITY OF CONNECTICUT

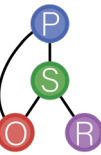


## Funding:

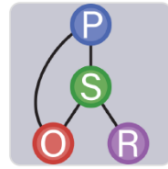
**National Science Foundation, Award No.: 2051084**

**Pratt & Whitney Endowed Professorship**

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the United States Government.



# Questions?



## Process Systems and Operations Research Laboratory

The PSOR Laboratory at UConn develops numerical analysis methods and software for process systems engineering applications.

27 followers

University of Connecticut, Storrs, CT, ...

<https://psor.uconn.edu>

[stuber@uconn.edu](mailto:stuber@uconn.edu)

<https://psor.uconn.edu>

<https://www.github.com/PSORLab>



### MS104

## Accelerating Optimization Using GPUs - Part II of II

Deterministic Global Optimization Using GPUs for the Lower-Bounding Problem

*Dimitri Alston, Robert Gottlieb, Matthew Stuber*

Wednesday, June 3<sup>rd</sup>, 2026, 10:15 AM

