Configurable Massively Parallel Solver for Lattice Problems

Nariaki Tateiwa(speaker)[1], Yuji Shinano[2], Keiichiro Yamamura[1]
Akihiro Yoshida[1], Shizuo Kaji[1], Masaya Yasuda[3], Katsuki Fujisawa[1]

[1] Kyushu University, Fukuoka, Japan
[2] Zuse Institute Berlin, Berlin, German
[3] Rikkyo University, Tokyo, Japan

1/October/2021
Second International UG Workshop 2021
Workshop on Parallel Algorithms in Tree Search and Mathematical Optimization
@online

Contribution and topics

We developed Shortest Vector Problem (SVP) solver using UG

CMAP-LAP: the *Configurable Massive Parallel Solver for Lattice Problem*

- ✓ First Generalized UG application
- ✓ SVP, the combinational problem,
  supports security of a post-quantum cryptography

Topics of this presentation

- ✓ How to use the Generalized UG to parallelize our solver?
- ✓ Unique new features for solving SVP
- ✓ Show performances of our solver via numerical experiments

Outline

Outline

1. Contribution & Introduction
2. What is SVP?
3. Key components of parallelization
4. System of our developed solver
5. Numerical experiments
6. Summary

Topics

1. Definition of Lattice & SVP
2. Features of SVP
3. Benchmark

What is SVP ?

Lattice

Definition

An n-dimensional **lattice** is

$$\mathcal{L}(\mathbf{B}) := \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i ;\ x_i \in \mathbb{Z} \right\}$$

$\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ are linearly independent vectors.
($\mathbf{B}$ is called a "*lattice basis*".)

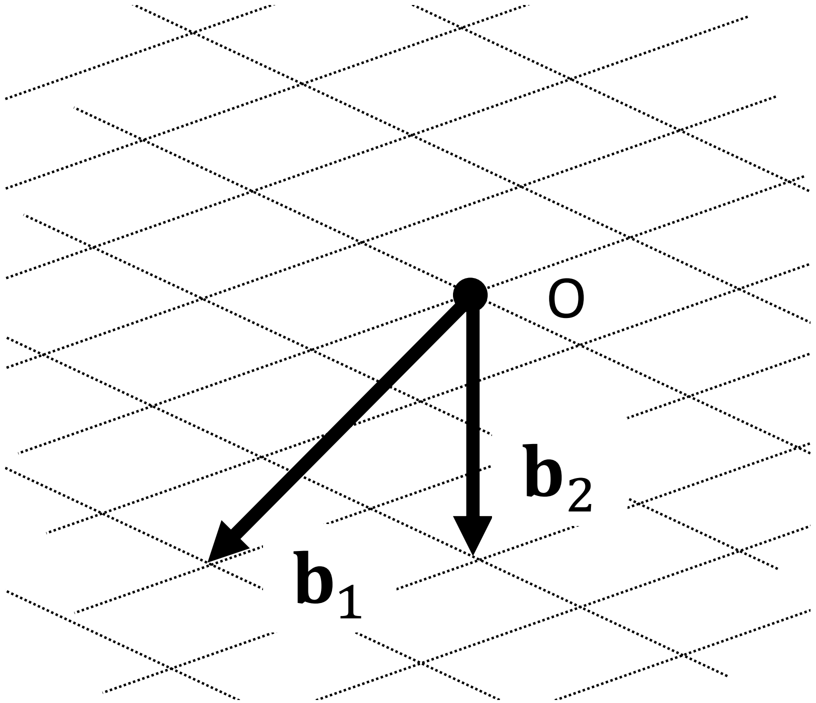


fig: 2-dimensional lattice

All intersection $\mathbf{v}$ in lattice are represented as
$\mathbf{v} = x_1\mathbf{b}_1 + x_2\mathbf{b}_2\ (x_1, x_2 \in \mathbb{Z})$

What is SVP ?

Lattice

Definition

An n-dimensional **lattice** is

$$\mathcal{L}(\mathbf{B}) := \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i;\ \ x_i \in \mathbb{Z} \right\}$$

$\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ are linearly independent vectors.
($\mathbf{B}$ is called a "*lattice basis*".)

Randomization of lattice basis

$\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{UB})$ $\forall \mathbf{U}$: Unimodular matrix
($U \in \mathbb{Z}^{n\times n}$, $\det(U) = \pm 1$)

The lattice does not change
by transformation with unimodular matrix

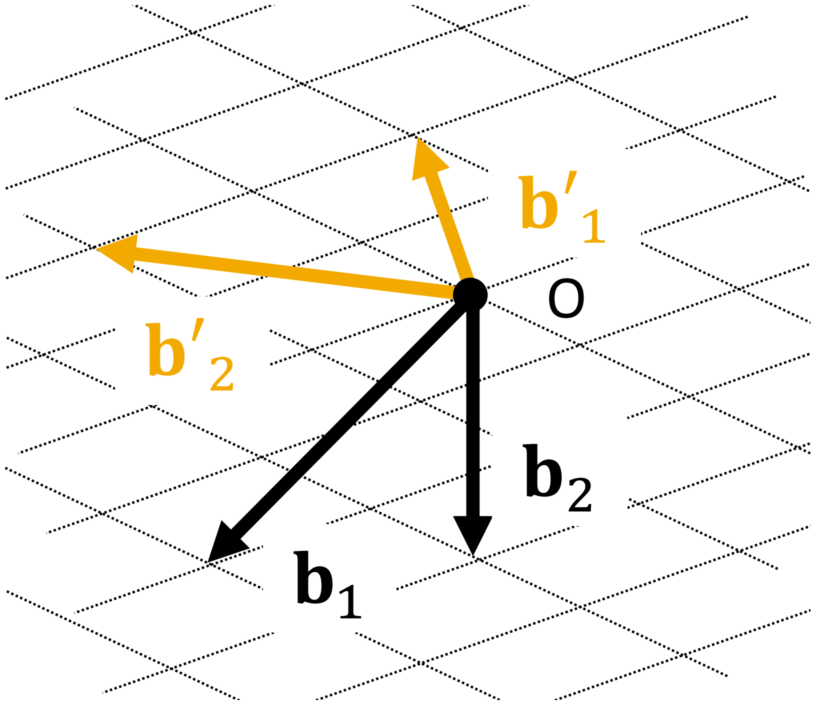


fig: 2-dimensional lattice

All intersection $\mathbf{v}$ in lattice are represented as
$\mathbf{v} = x_1\mathbf{b}_1 + x_2\mathbf{b}_2$ $(x_1, x_2 \in \mathbb{Z})$

What is SVP ?

Shortest Vector Problem

Definition

The **Shortest Vector Problem (SVP)** asks to find the *shortest non-zero vector* in the lattice

$$\begin{aligned} &\text{minimize} \quad \|\mathbf{v}\| \\ &\text{subject to} \quad \mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\} \end{aligned}$$

||

$$\begin{aligned} &\underset{\boldsymbol{x} = (x_1, \dots, x_N) \in \mathbb{Z}^N}{\text{minimize}} \quad \left\| \sum_{i=1}^{N} x_i \mathbf{b}_i \right\| \\ &\text{subject to} \quad \boldsymbol{x} \neq \mathbf{0} \end{aligned}$$

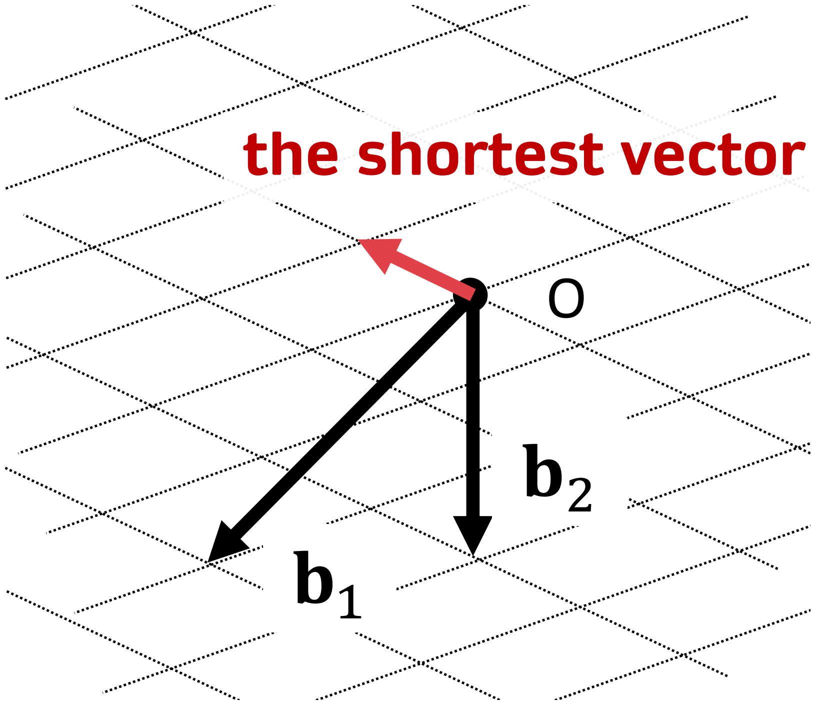


fig: 2-dimensional lattice

What is SVP ?

Shortest Vector Problem

Definition

The **Shortest Vector Problem (SVP)** asks to find the *shortest non-zero vector* in the lattice

$$\text{minimize} \quad \|\mathbf{v}\| \qquad \text{subject to} \quad \mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\}$$

||

$$\underset{\boldsymbol{x}=(x_1,\dots,x_N)\in\mathbb{Z}^N}{\text{minimize}} \quad \left\| \sum_{i=1}^{N} x_i \mathbf{b}_i \right\| \qquad \text{subject to} \quad \boldsymbol{x} \neq \mathbf{0}$$

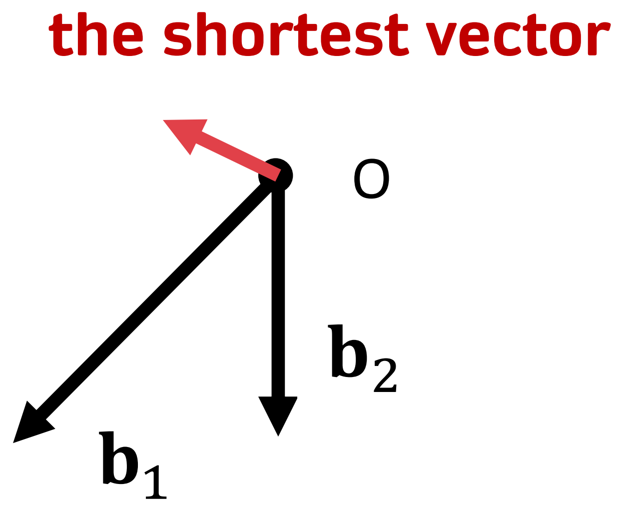


fig: 2-dimensional lattice

What is SVP ?

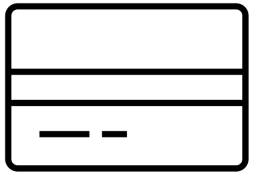
Why we try to solve Shortest Vector Problem?

Features

- no single definite algorithm
- SVP supports the security of some lattice-based cryptographies

Lattice-based cryptography

- Common cryptographies have risk to be broken by quantum computers
- **Lattice-based cryptography** is the candidate of new standard post-quantum cryptographies
- urgent to investigate
  - **security level**



IC, credit card ..

If the size of the public key is large, some of the current cryptosystems cannot be replaced new cryptosystem due to memory limitation.

Benchmark

SVP Challenge

- contest of solving **approximate SVP** of 40 – 200 dimension

1.05-approximate SVP

finding $\mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\}$
subject to $\|\mathbf{v}\| \leq 1.05\, \lambda_1(\mathcal{L}(\mathbf{B}))$

$\lambda_1(\mathcal{L}(\mathbf{B}))$: the estimated optimal value

- Sieve-algorithm solver (G6K) is major, recently

HALL OF FAME

Position	Dimension	Euclidean Norm	Contestant	Algorithm	Subm. Date	Approx. Factor
1	180	3509	L. Ducas, M. Stevens, W. van Woerden	Sieving	2021-02-8	1.04002
2	178	3447	L. Ducas, M. Stevens, W. van Woerden	Sieving	2021-02-8	1.02725
3	176	3487	L. Ducas, M. Stevens, W. van Woerden	Sieving	2020-10-13	1.04411
4	170	3438	L. Ducas, M. Stevens, W. van Woerden	Sieving	2020-05-12	1.04690
5	158	3240	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	Sieving	2021-01-22	1.02311
6	157	3320	L. Ducas, M. Stevens, W. van Woerden	Sieving	2019-05-20	1.04906
7	156	3219	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	Sieving	2021-01-22	1.01986
8	155	3165	M. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. Postlethwaite, M. Stevens, P. Karpman	Sieving	2018-09-18	1.00803
9	154	3200	Sho Hasegawa, Yuntao Wang, Eiichiro Fujisaki	Sieving	2021-02-1	1.02258
10	153	3192	Martin Albrecht, Leo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, Marc Stevens	Sieving	2018-08-30	1.02102

Outline

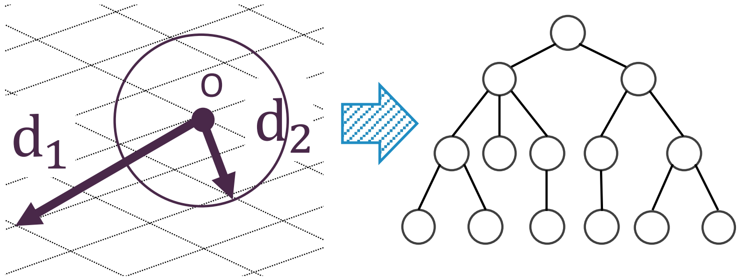
1. Contribution & Introduction
2. What is SVP?
3. Key components of parallelization
4. System of our developed solver based
5. Numerical experiments
6. Summary

Topics

1. ▷ Algorithms for SVP
2. Features of algorithms for parallelization

Lattice Problem Algorithms

Enumeration (Exactive)



- Depth-first search
- Nodes of the tree correspond to lattice vectors
- Tree contains all lattice vectors with norm $\leq R$ (parameter)

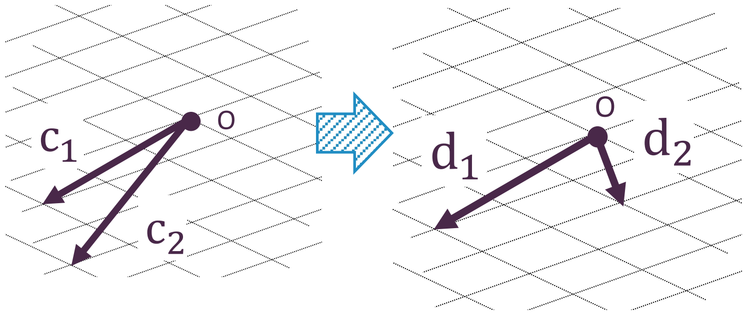
✓ **pros: Low memory usage**

✓ **cons: Huge searching time**

- ✓ **dim 100 --> 10^06 years**
- ✓ **dim 130 --> 10^25 years**

(dim = dimension)

Reduction (Approximate)

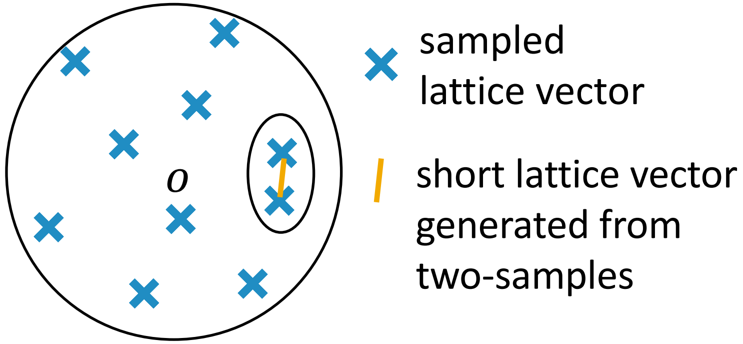


- Matrix transformation
- make the lattice basis as close to orthogonal as possible

✓ **pros: Low memory usage**

✓ **cons: No guarantee for finding the shortest vector**

Sieve (Probabilistic)



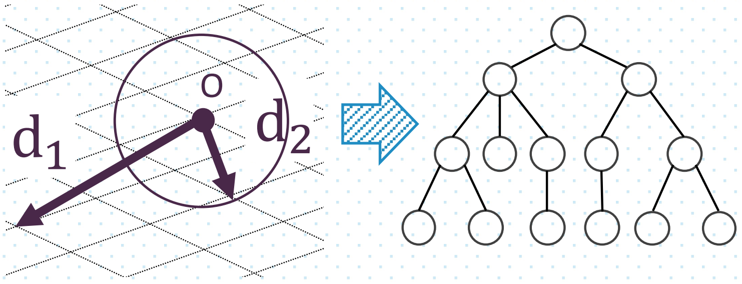
- Sampling and Reduce
- Generate shorter vectors by addition(+) and subtraction(-) sampled lattice vectors
- Based on birthday paradox

✓ **pros: High performance**

✓ **cons: Huge memory usage**

Lattice Problem Algorithms

Enumeration (Exactive)

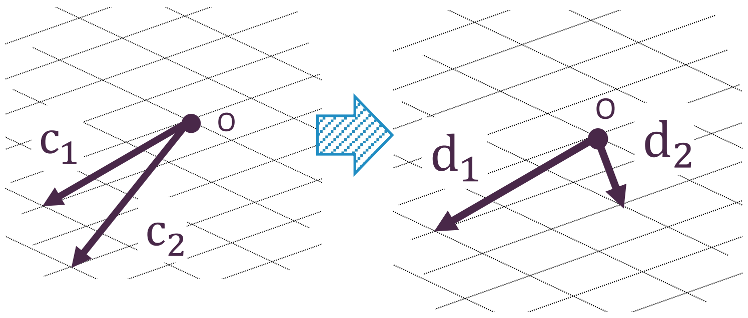


- Depth-first search
- Nodes of the tree correspond to lattice vectors
- Tree contains all lattice vectors with norm $\leq R$ (parameter)

✓ **pros: Low memory usage**
✓ **cons: Huge searching time**
  - ✓ **dim 100 --> 10^06 years**
  - ✓ **dim 130 --> 10^25 years**

(dim = dimension)

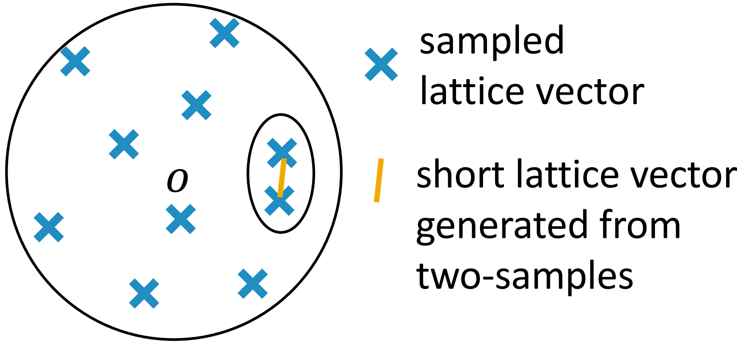
Reduction (Approximate)



- Matrix transformation
- make the lattice basis as close to orthogonal as possible

✓ **pros: Low memory usage**
✓ **cons: No guarantee for finding the shortest vector**

Sieve (Probabilistic)

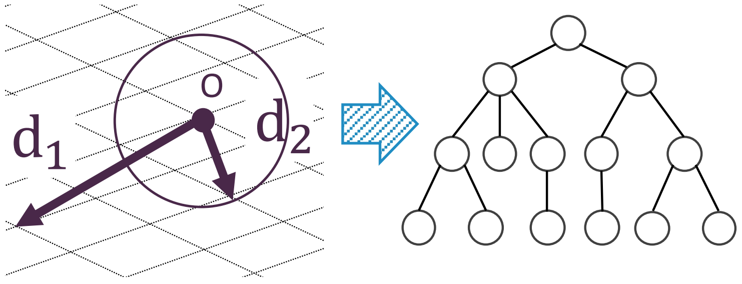


- Sampling and Reduce
- Generate shorter vectors by addition(+) and subtraction(-) sampled lattice vectors
- Based on birthday paradox

✓ **pros: High performance**
✓ **cons: Huge memory usage**

Lattice Problem Algorithms

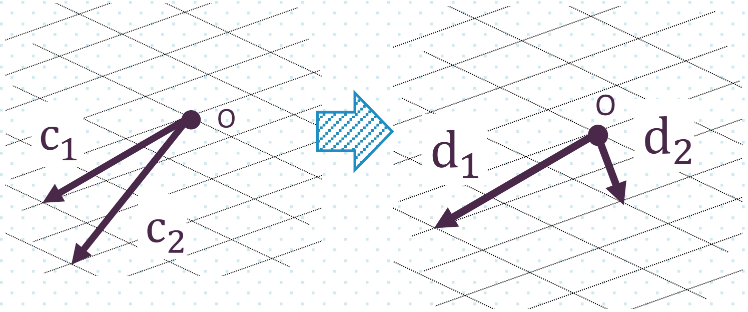
Enumeration (Exactive)



- Depth-first search
- Nodes of the tree correspond to lattice vectors
- Tree contains all lattice vectors with norm $\leq R$ (parameter)

✓ **pros: Low memory usage**
✓ **cons: Huge searching time**
  ✓ **dim 100 --> 10^06 years**
  ✓ **dim 130 --> 10^25 years**
(dim = dimension)

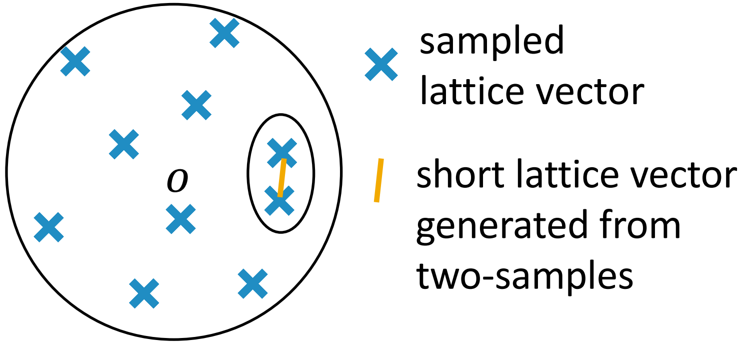
Reduction (Approximate)



- Matrix transformation
- make the lattice basis as close to orthogonal as possible

✓ **pros: Low memory usage**
✓ **cons: No guarantee for finding the shortest vector**

Sieve (Probabilistic)

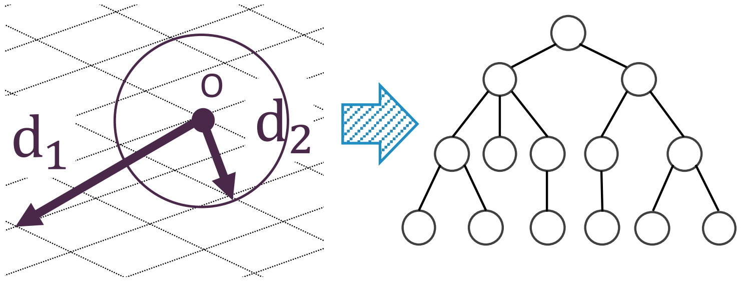


- Sampling and Reduce
- Generate shorter vectors by addition(+) and subtraction(-) sampled lattice vectors
- Based on birthday paradox

✓ **pros: High performance**
✓ **cons: Huge memory usage**

Lattice Problem Algorithms

Enumeration (Exactive)



- Depth-first search
- Nodes of the tree correspond to lattice vectors
- Tree contains all lattice vectors with norm $\leq R$ (parameter)

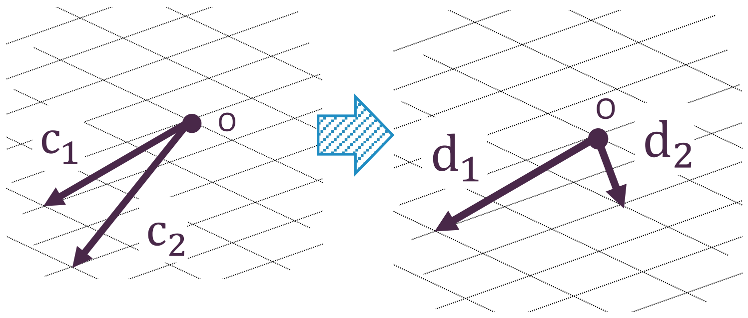
✓ **pros: Low memory usage**

✓ **cons: Huge searching time**

- ✓ **dim 100 --> 10^06 years**
- ✓ **dim 130 --> 10^25 years**

(dim = dimension)

Reduction (Approximate)

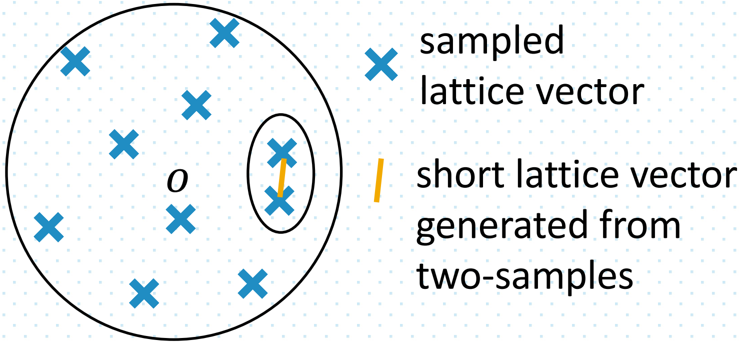


- Matrix transformation
- make the lattice basis as close to orthogonal as possible

✓ **pros: Low memory usage**

✓ **cons: No guarantee for finding the shortest vector**

Sieve (Probabilistic)



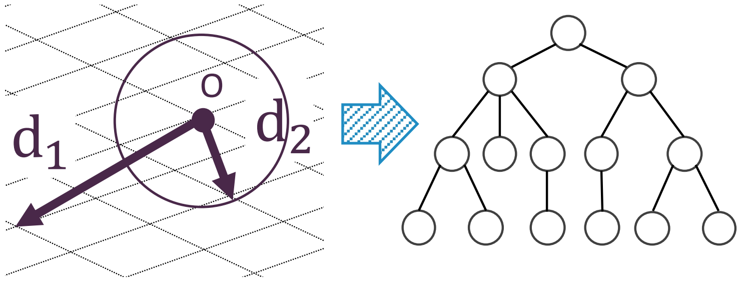
- Sampling and Reduce
- Generate shorter vectors by addition(+) and subtraction(-) sampled lattice vectors
- Based on birthday paradox

✓ **pros: High performance**

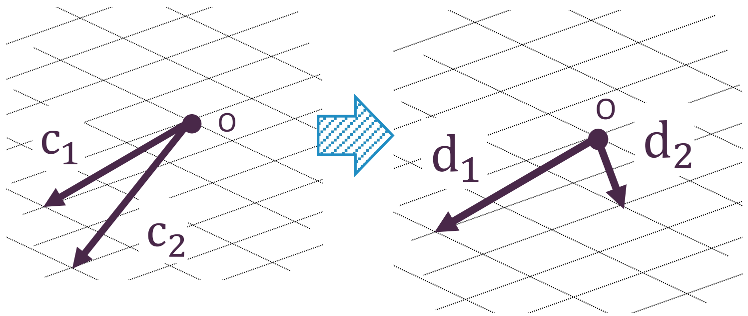
✓ **cons: Huge memory usage**

Key components of parallelization

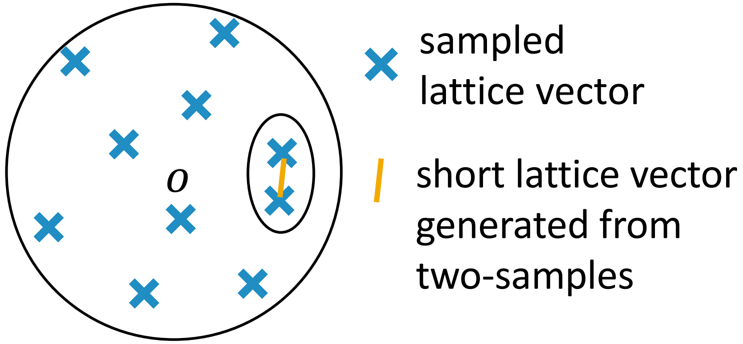
Enumeration (Exactive)



Reduction (Approximate)



Sieve (Probabilistic)

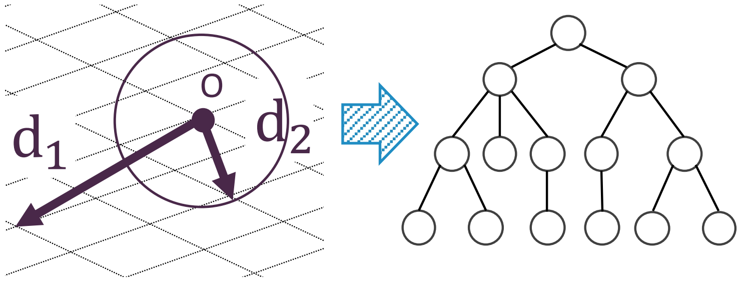


common features of algorithms

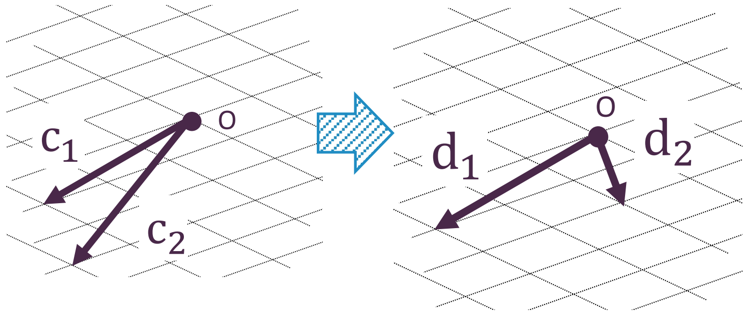
✓ Behavior changes depending on **input basis**
✓ Algorithms can also find **short vectors** (not only the shortest one)
✓ Interactions of different algorithms

Key components of parallelization

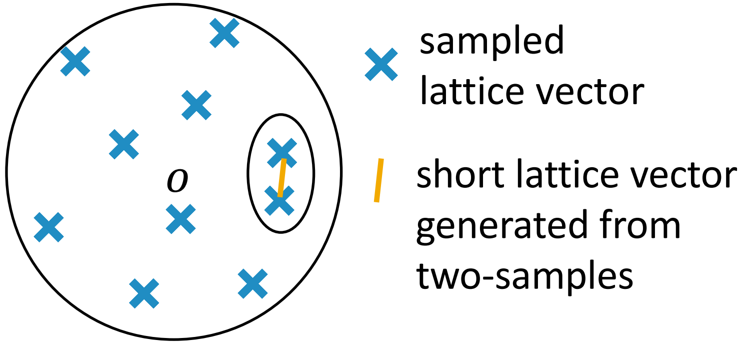
Enumeration
(Exactive)



Reduction
(Approximate)



Sieve
(Probabilistic)



common features of algorithms

- ✓ Behavior changes depending on **input basis**
- ✓ Algorithms can also find **short vectors** (not only t
- ✓ Interactions of different algorithms

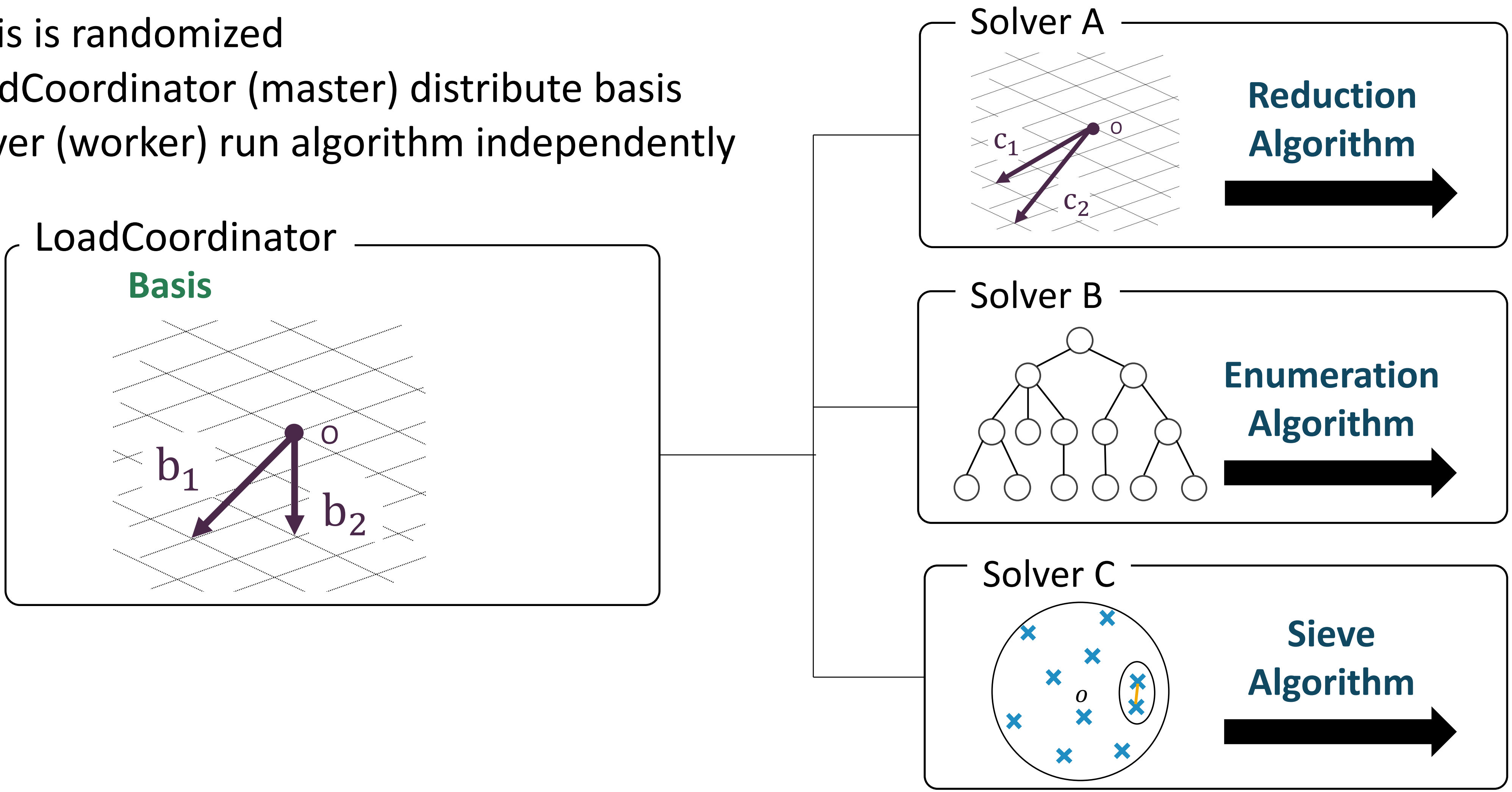
Randomization of lattice basis

$\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{UB})$ $\forall\mathbf{U}$: Unimodular matrix
$(U \in \mathbb{Z}^{n\times n}, \det(U) = \pm 1)$

The lattice does not change
by transformation with unimodular matrix

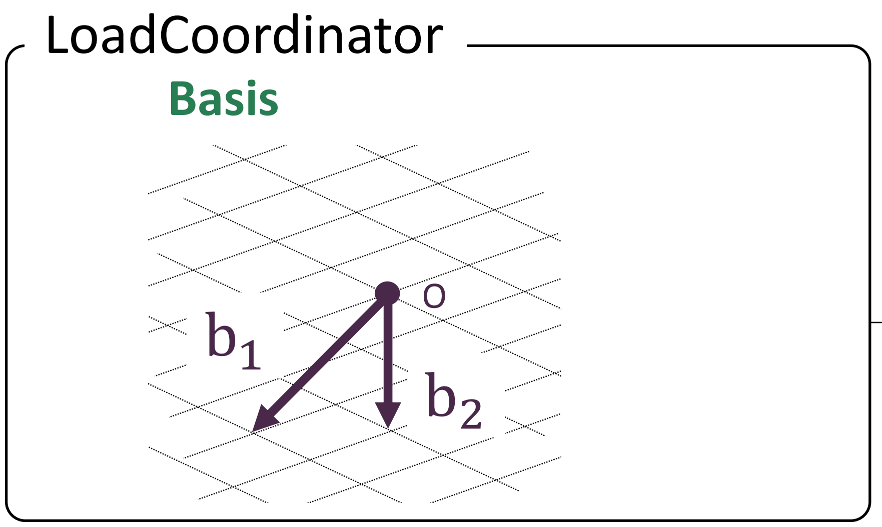
Parallel Strategy Idea

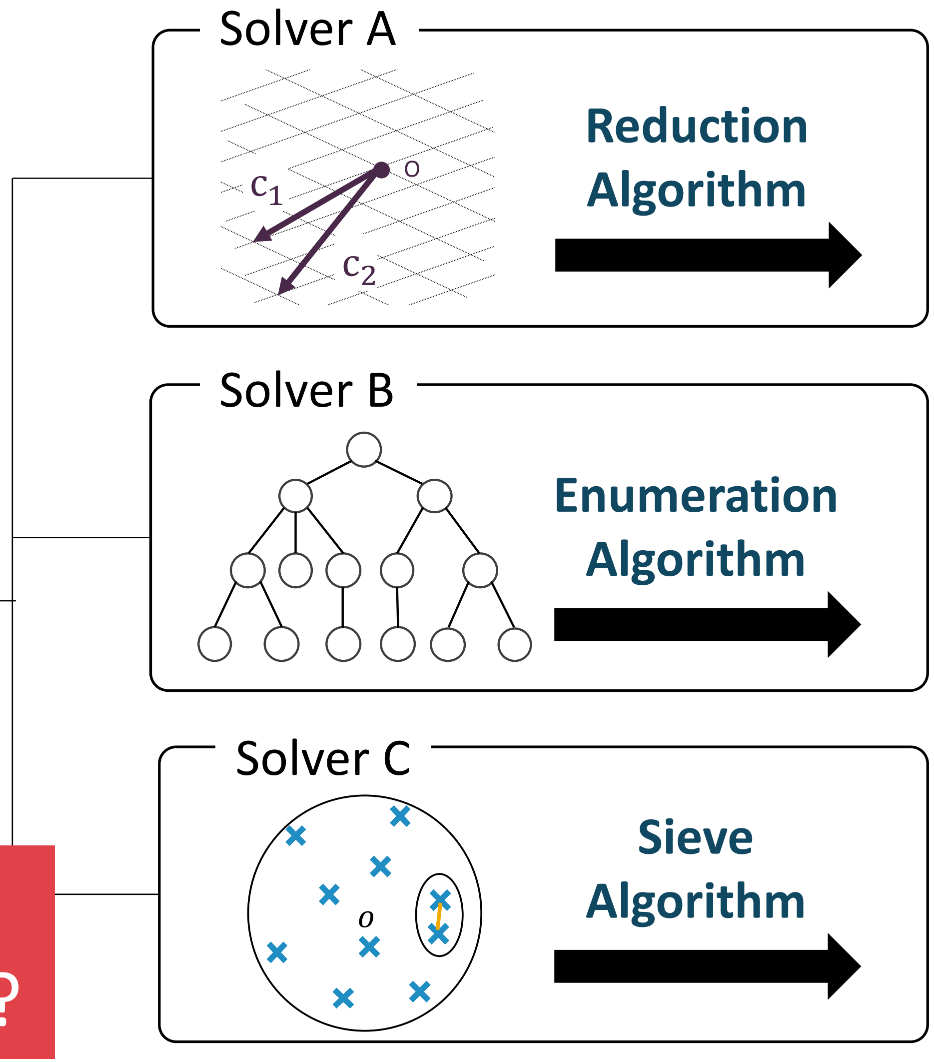
- ✓ Task parallel strategy
  - ✓ basis is randomized
  - ✓ LoadCoordinator (master) distribute basis
  - ✓ Solver (worker) run algorithm independently



Parallel Strategy Idea

- ✓ Task parallel strategy
  - ✓ basis is randomized
  - ✓ LoadCoordinator (master) distribute basis
  - ✓ Solver (worker) run algorithm independently

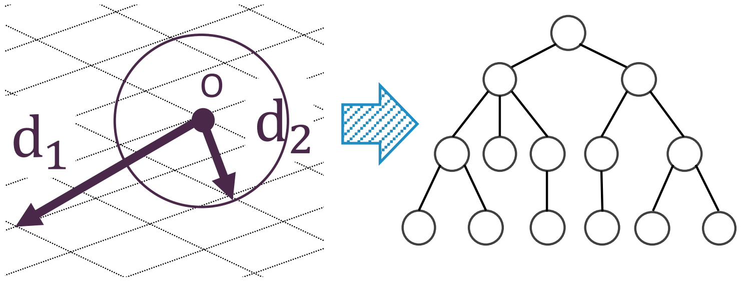




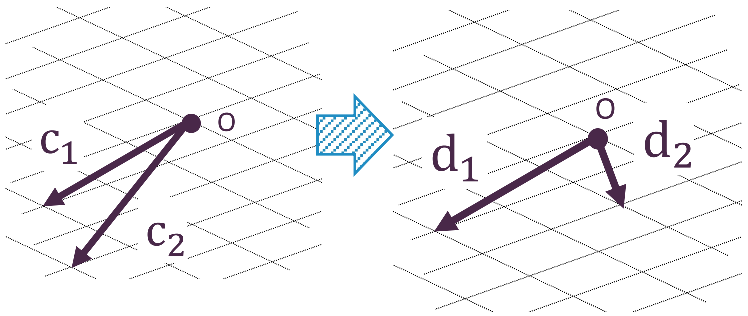
Can we improve the performance by taking advantage of lattice properties?

Key components of parallelization

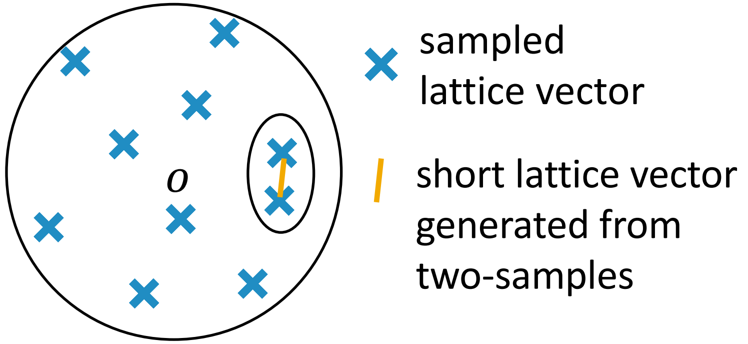
Enumeration
(Exactive)



Reduction
(Approximate)



Sieve
(Probabilistic)



common features of algorithms

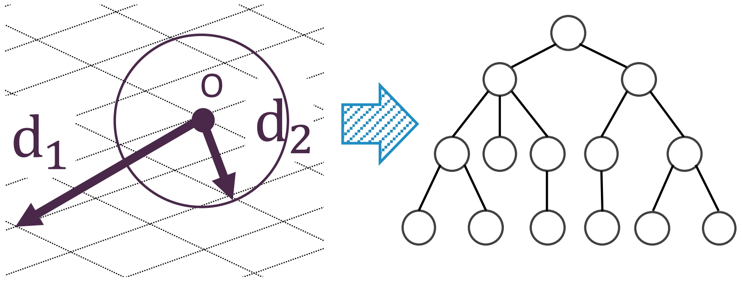
✓ Behavior changes depending on **input basis**

✓ Algorithms can also find **short vectors** (not only the shortest one)

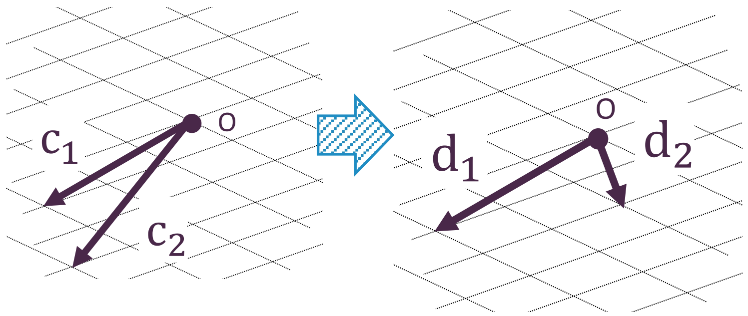
✓ Interactions of different algorithms

Key components of parallelization

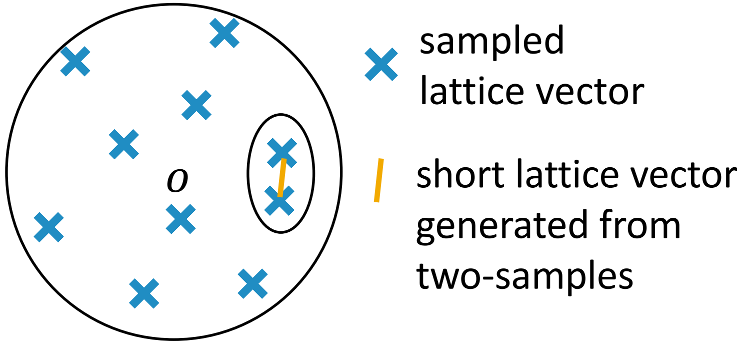
Enumeration
(Exactive)



Reduction
(Approximate)



Sieve
(Probabilistic)



common features of algorithms

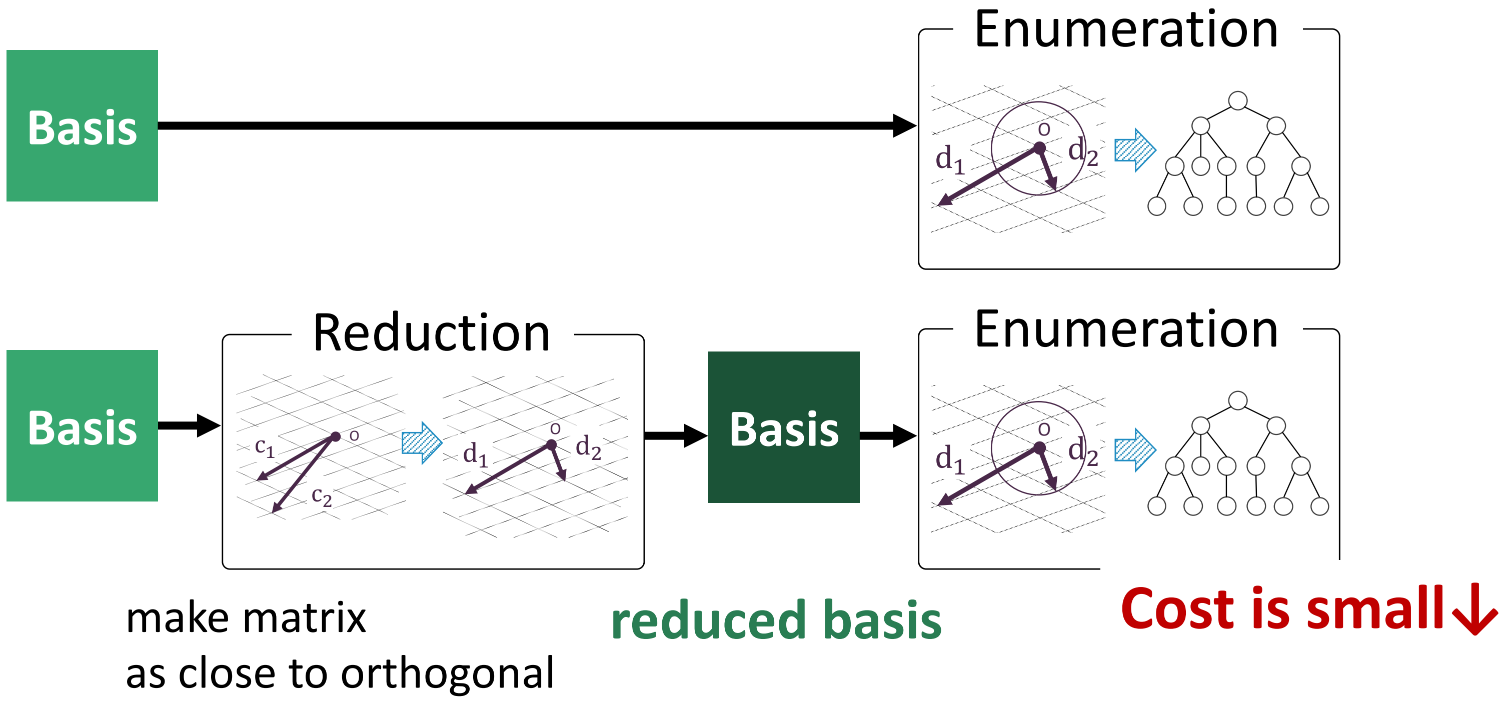
- ✓ Behavior changes depending on **input basis**
- ✓ Algorithms can also find **short vectors** (not only the shortest one)
- ✓ Interactions of different algorithms

Key components of parallelization

Interactions of algorithms

- ✓ lattice algorithms find
  - **short lattice vectors, not only shortest one**
  - **reduced basis**
- ✓ These can be used as **input** and **booster** for other algorithms

Case of Enumeration



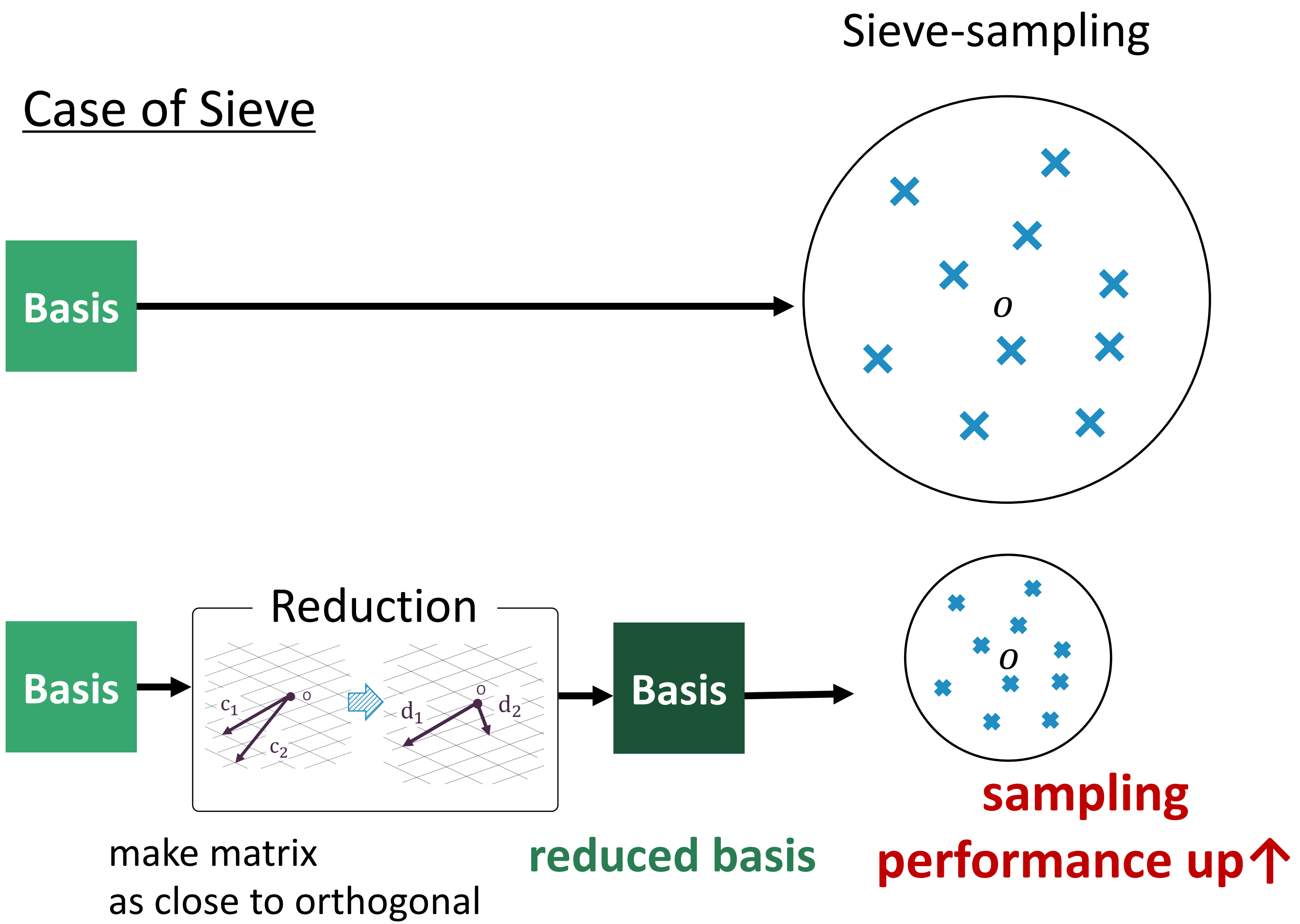
Key components of parallelization

Interactions of algorithms

✓ lattice algorithms find
- **short lattice vectors, not only shortest one**
- **reduced basis**

✓ These can be used as **input** and **booster** for other algorithms

Case of Sieve

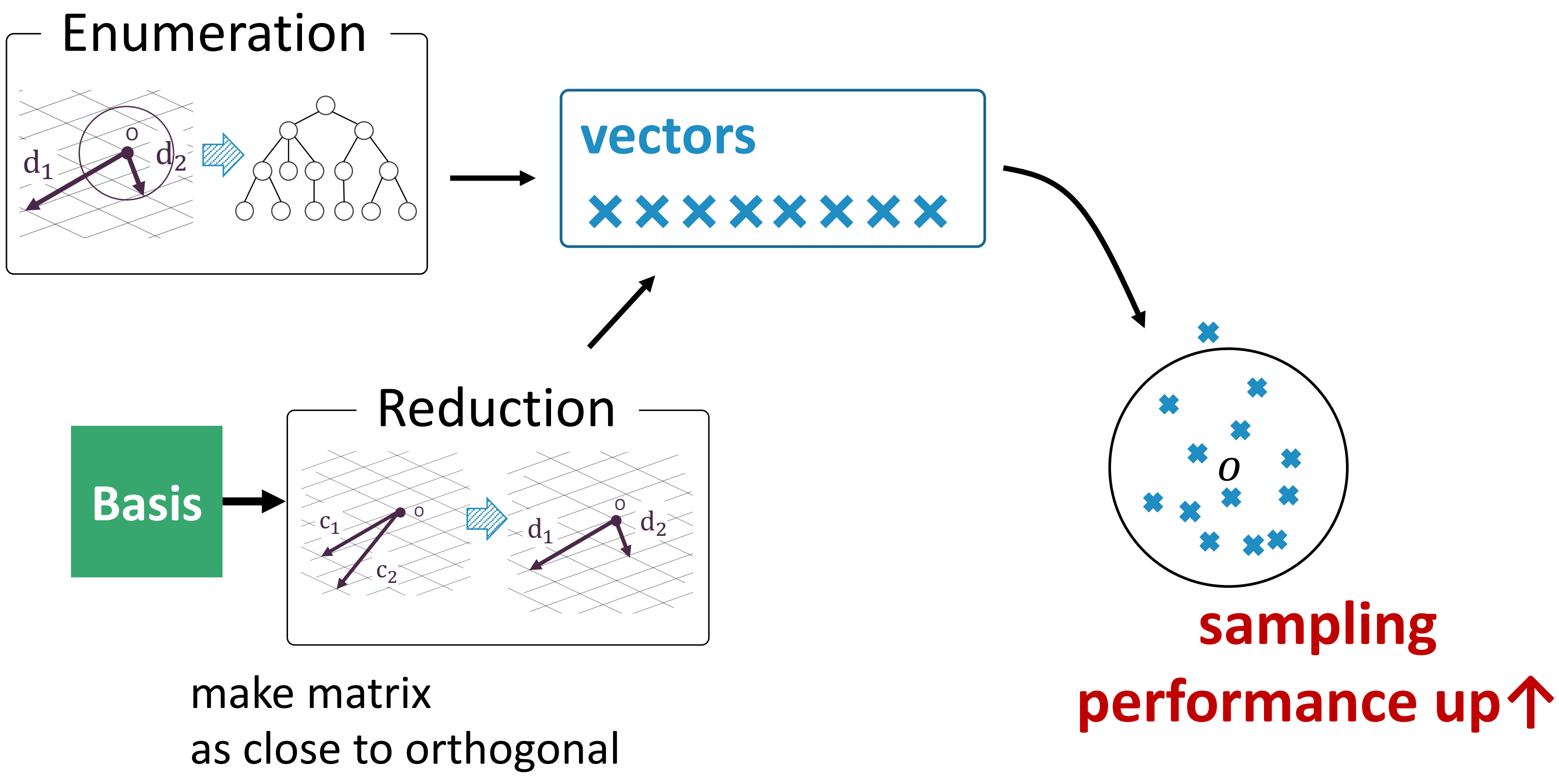


Key components of parallelization

Interactions of algorithms

- ✓ lattice algorithms find
  - **short lattice vectors, not only shortest one**
  - **reduced basis**
- ✓ These can be used as **input** and **booster** for other algorithms

Case of Sieve



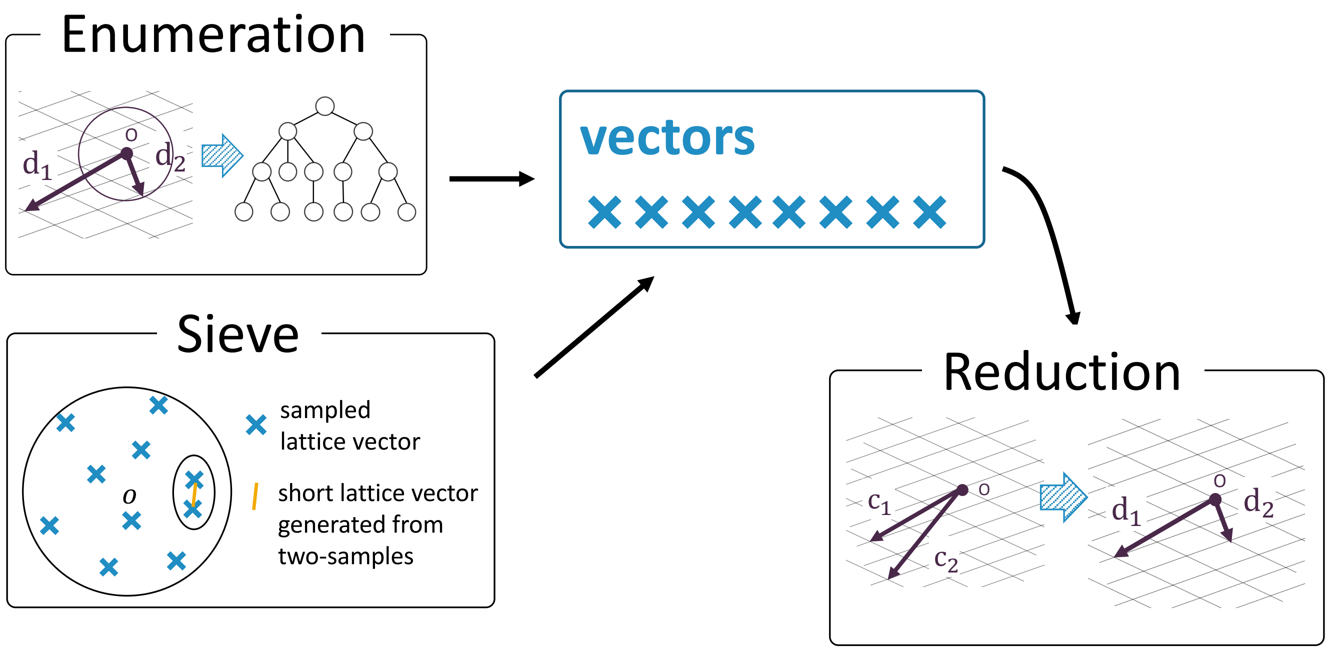
Key components of parallelization

Interactions of algorithms

- ✓ lattice algorithms find
  - **short lattice vectors, not only shortest one**
  - **reduced basis**

- ✓ These can be used as **input** and **booster** for other algorithms

<u>Case of Reduction</u>



output more orthogonality basis ↑

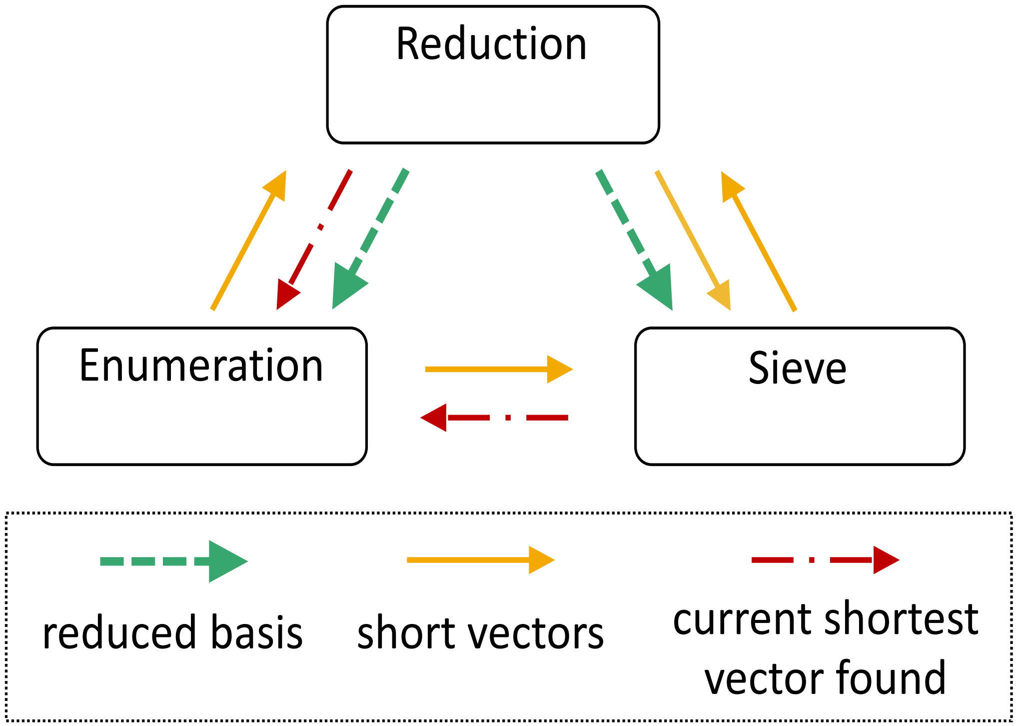
Key components of parallelization

By-products of the lattice algorithm

✓ lattice algorithms find
  - **short lattice vectors, not only shortest one**
  - **reduced basis**

✓ These can be used as **input** and **booster** for other algorithms

✓ **However, there is no SVP solver which effectively utilizes these interactions**



Outline

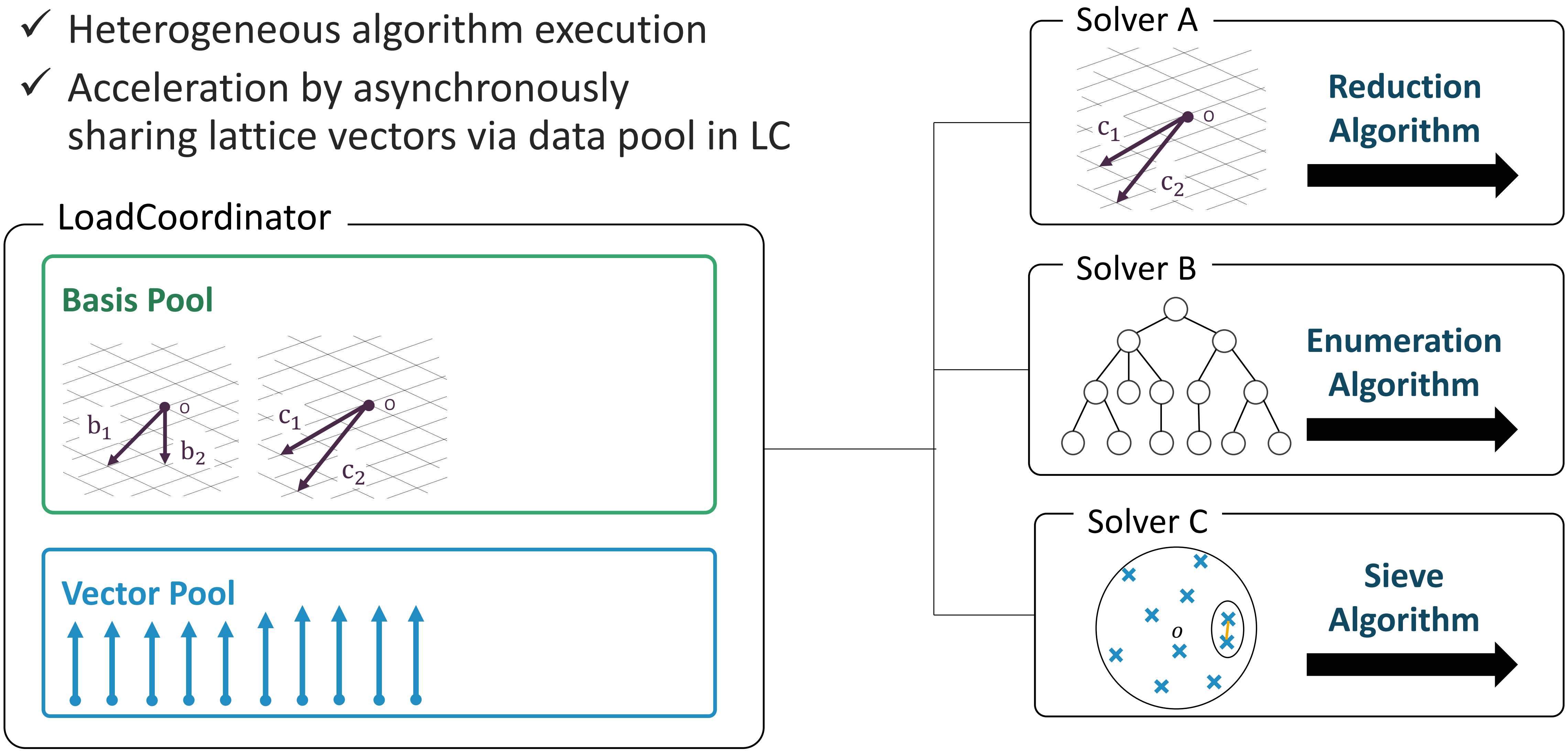
1. Contribution & Introduction
2. What is SVP?
3. Key components of parallelization
4. System of our solver based on UG
5. Numerical experiments
6. Summary

Topics

1. ▷ Overview
2. Communication of Task
3. Checkpointing
4. Asynchronously Communication

CMAP-LAP: Our new solver

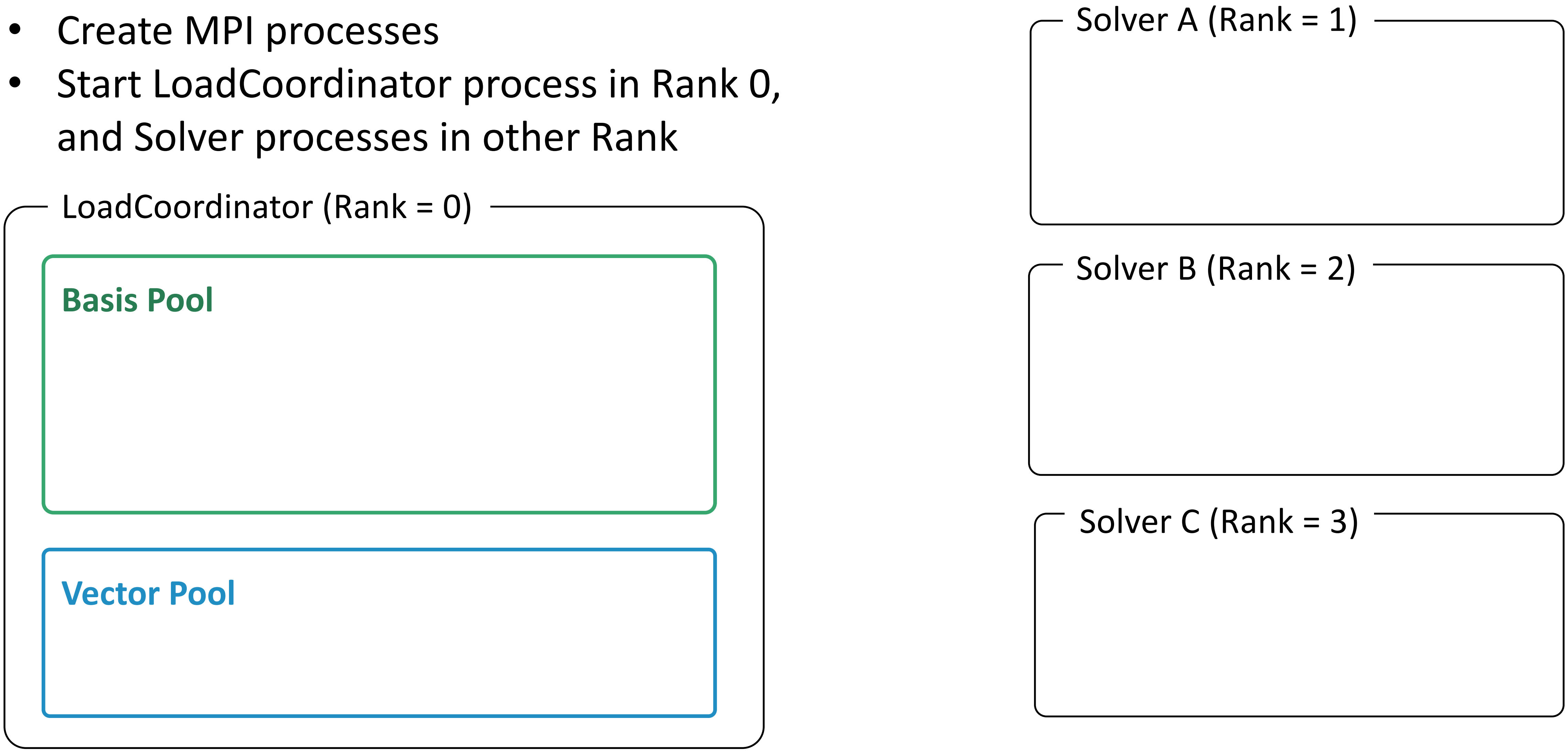
- ✓ Supervisor-Worker parallelization type
- ✓ Heterogeneous algorithm execution
- ✓ Acceleration by asynchronously sharing lattice vectors via data pool in LC



CMAP-LAP: Our new solver

Flow of execution

- Create MPI processes
- Start LoadCoordinator process in Rank 0, and Solver processes in other Rank



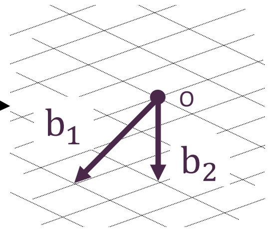
CMAP-LAP: Our new solver

Flow of execution

Give Instance

LoadCoordinator

Basis Pool



Vector Pool

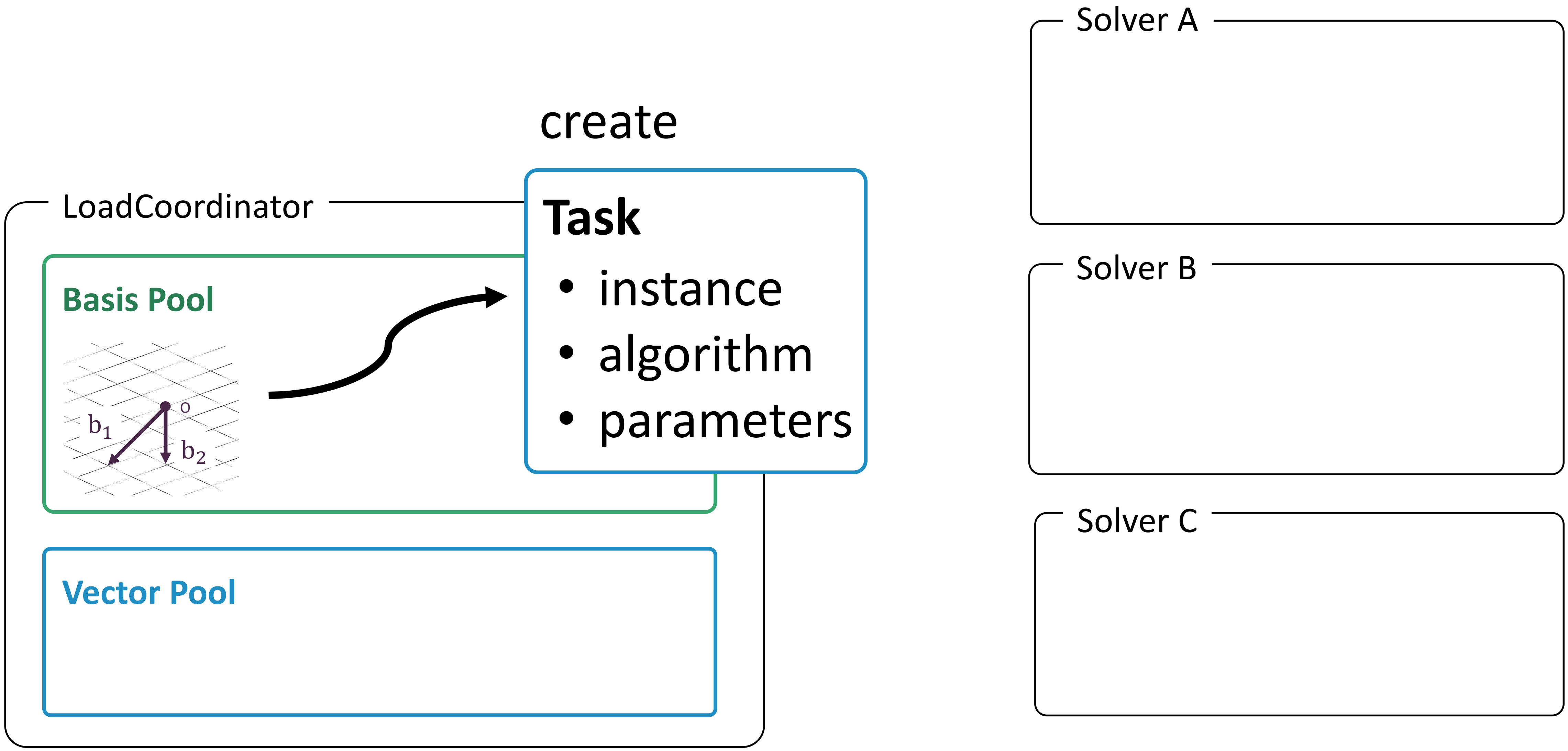
Solver A

Solver B

Solver C

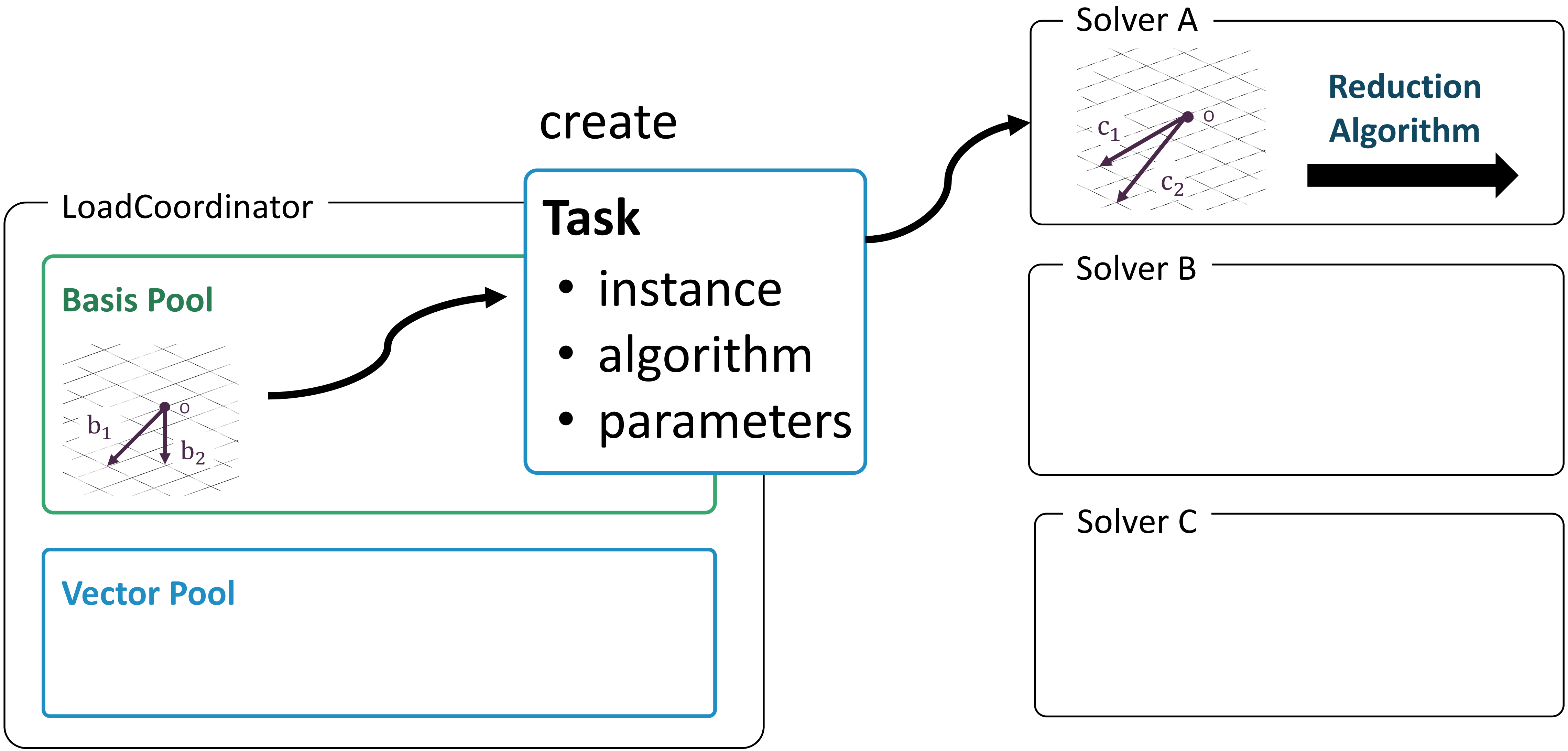
CMAP-LAP: Our new solver

Flow of execution



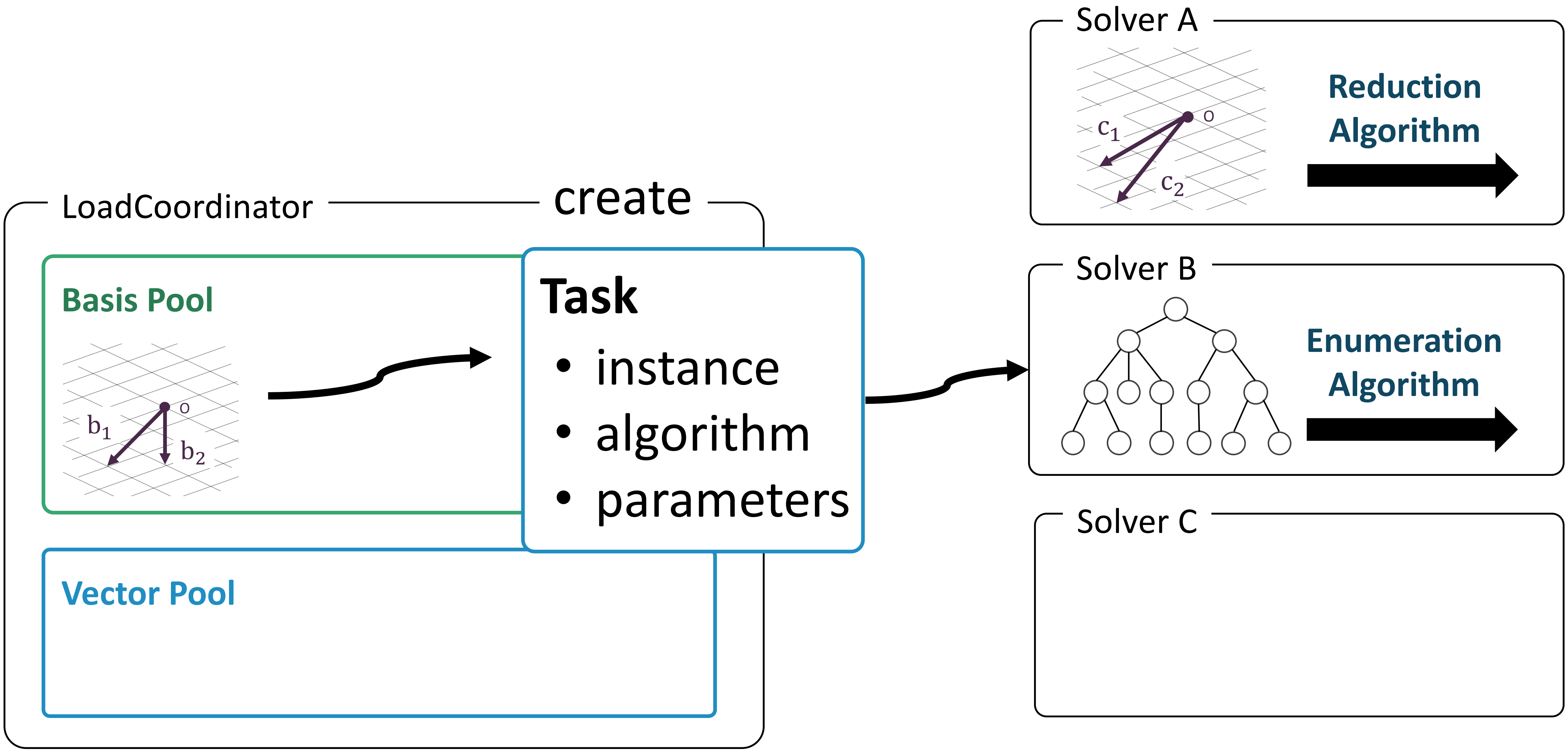
CMAP-LAP: Our new solver

Flow of execution



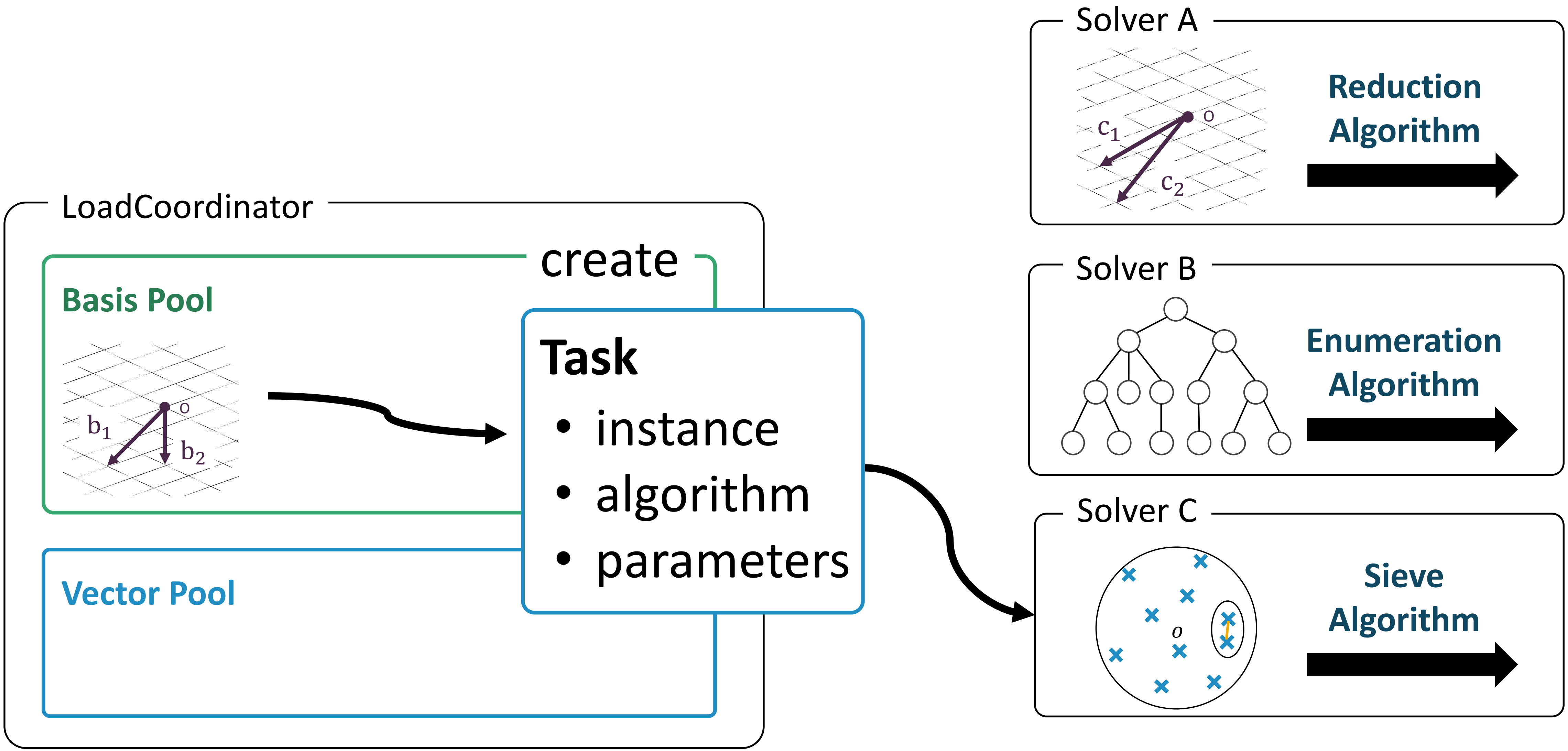
CMAP-LAP: Our new solver

Flow of execution



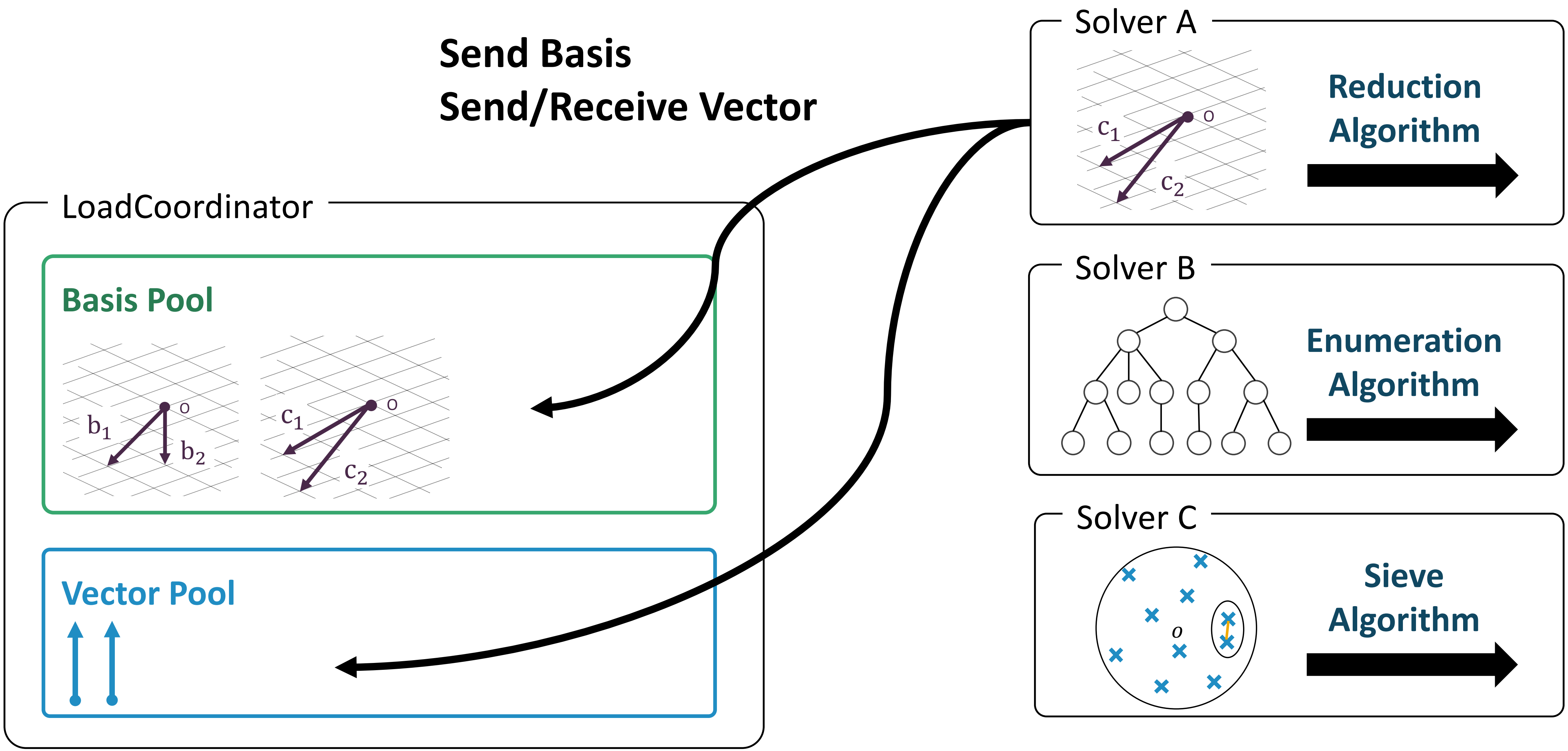
CMAP-LAP: Our new solver

Flow of execution



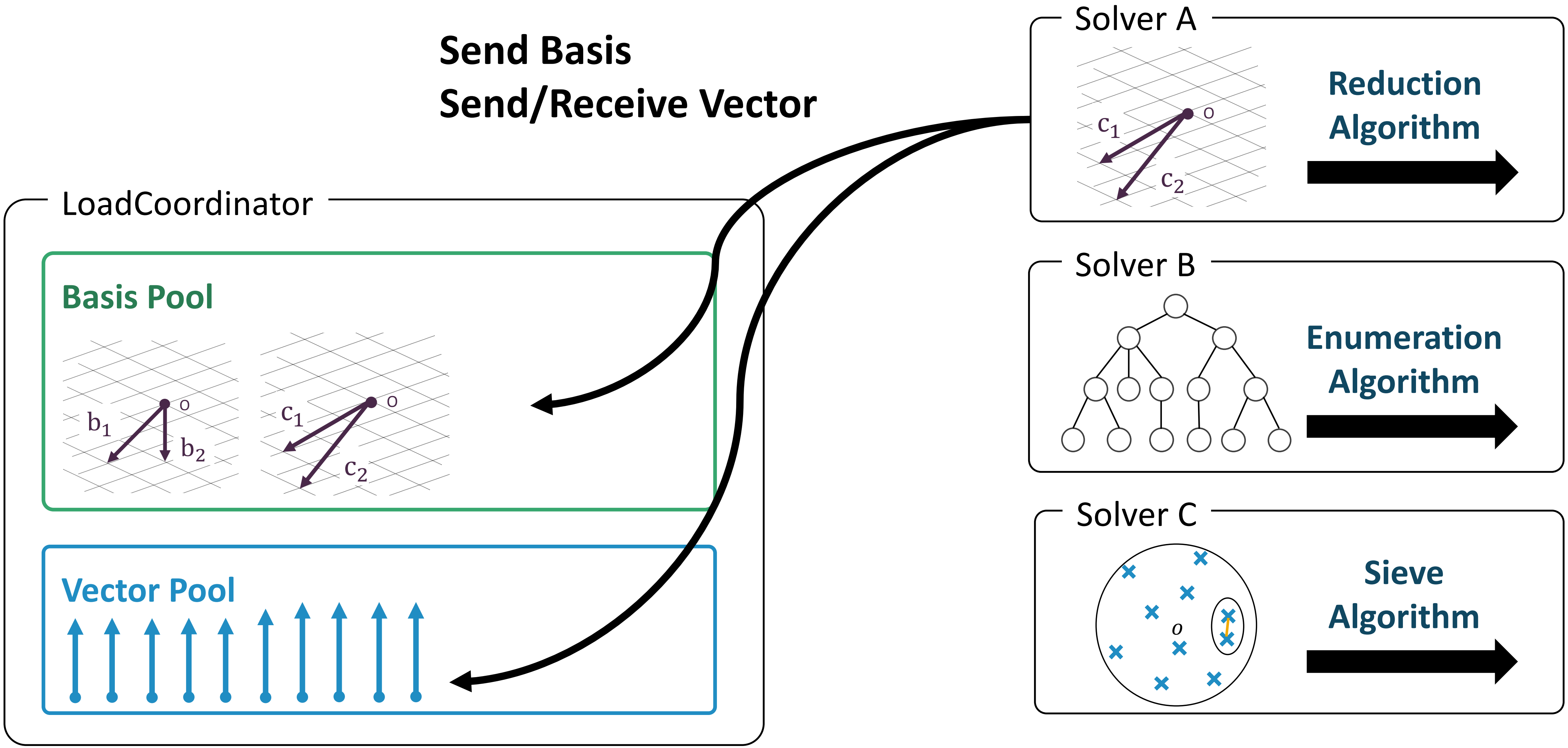
CMAP-LAP: Our new solver

Flow of execution



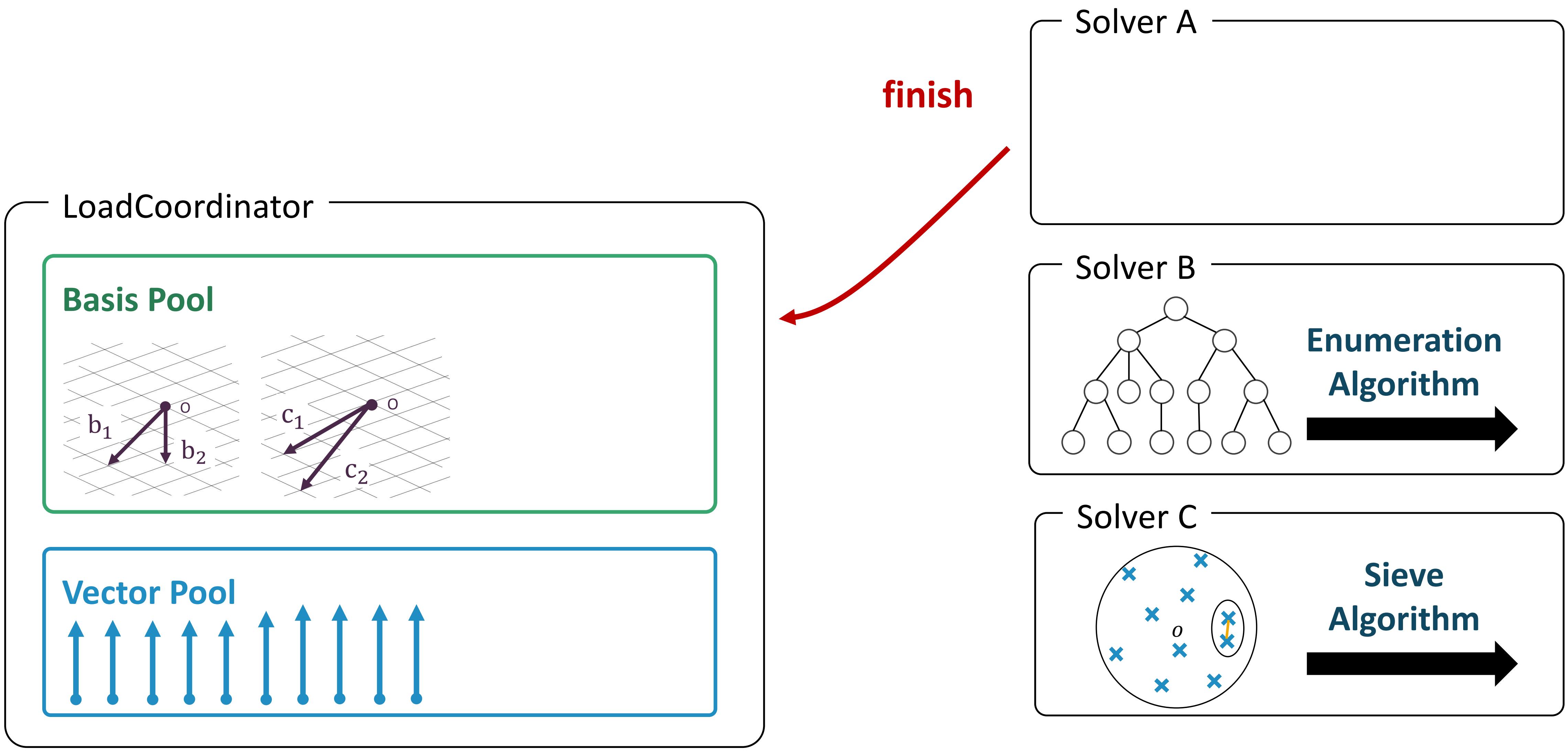
CMAP-LAP: Our new solver

Flow of execution



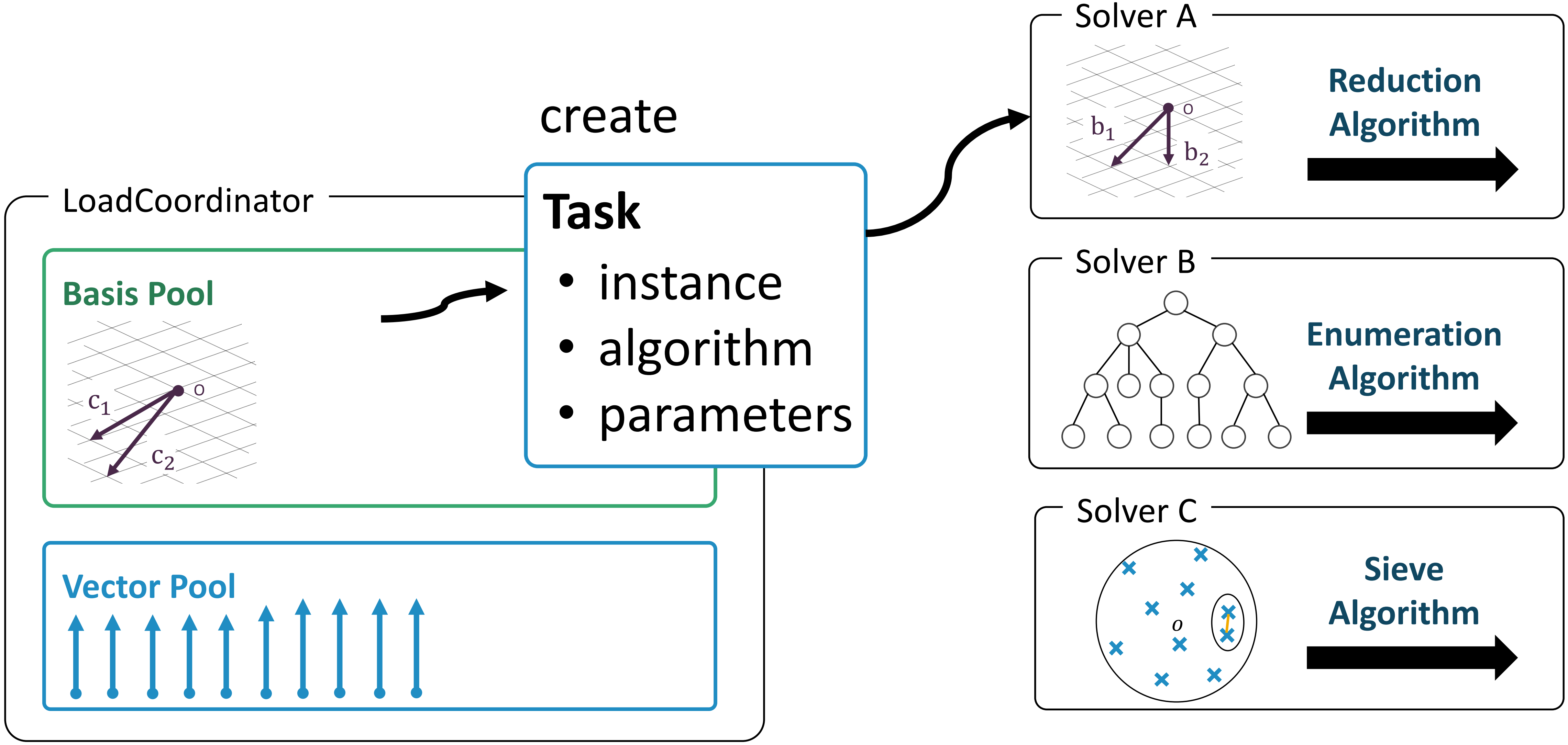
CMAP-LAP: Our new solver

Flow of execution

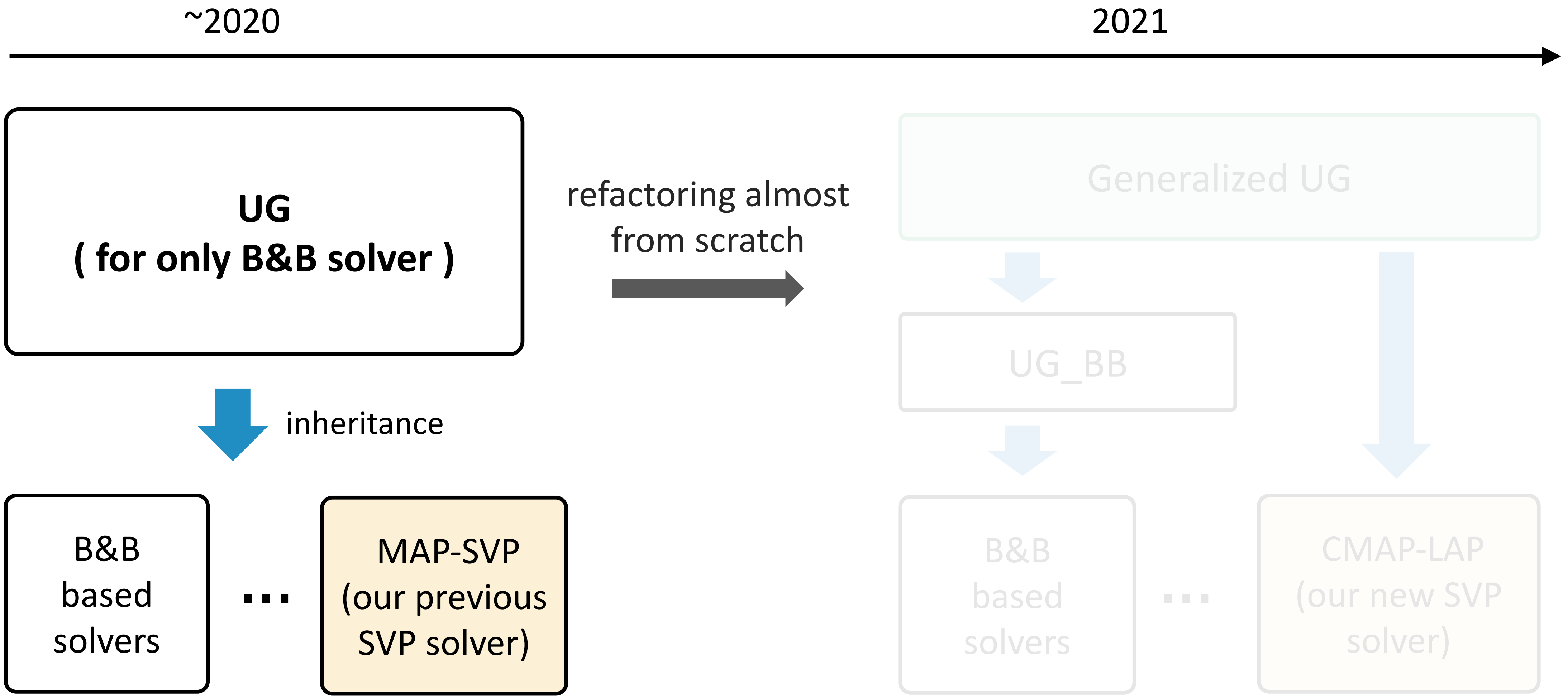


CMAP-LAP: Our new solver

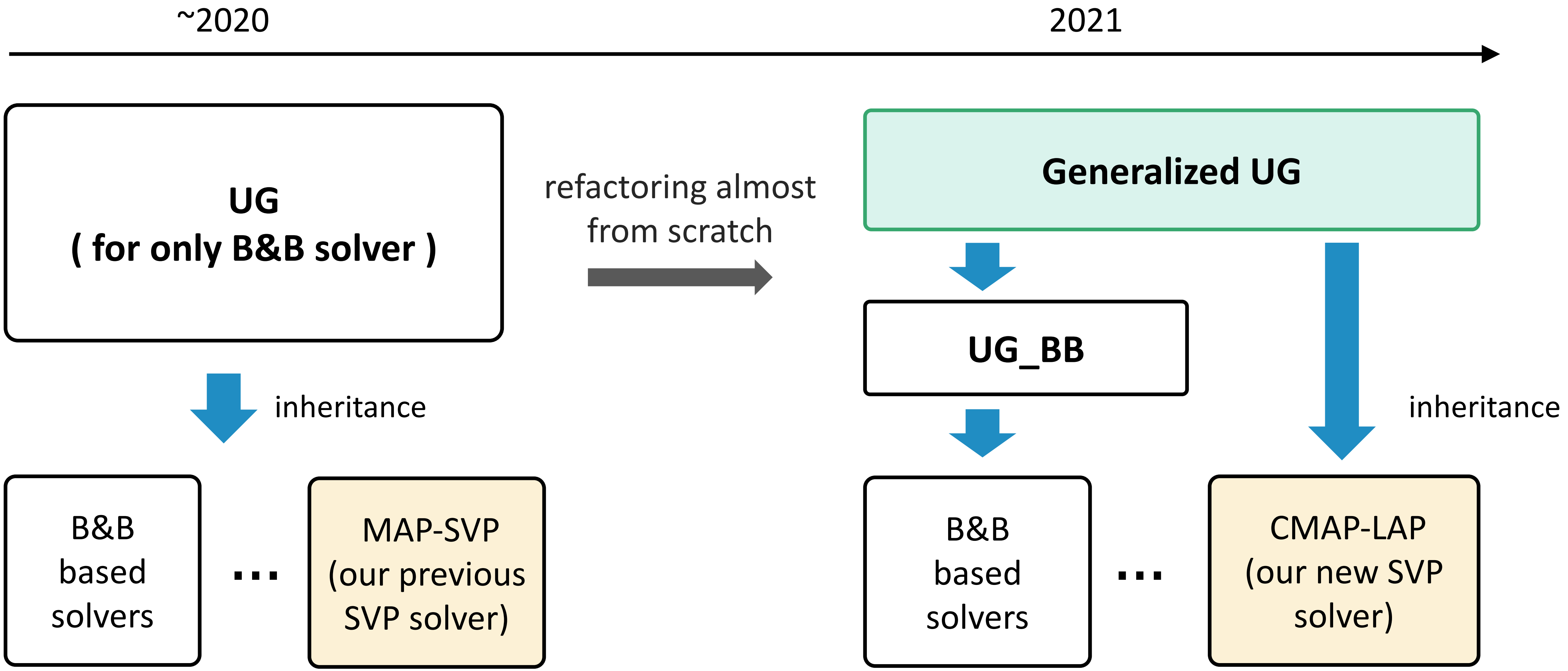
Flow of execution



Implementation of Our new solver



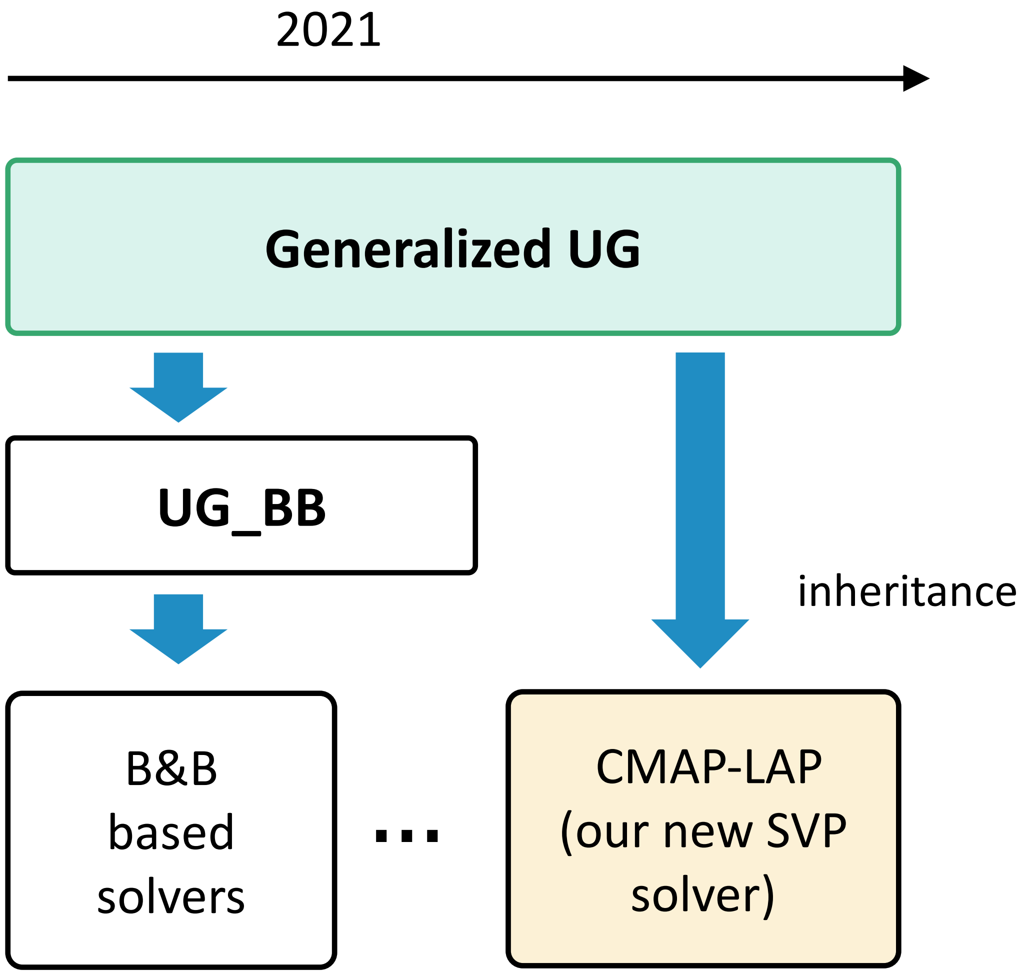
Implementation of Our new solver



Implementation of Our new solver

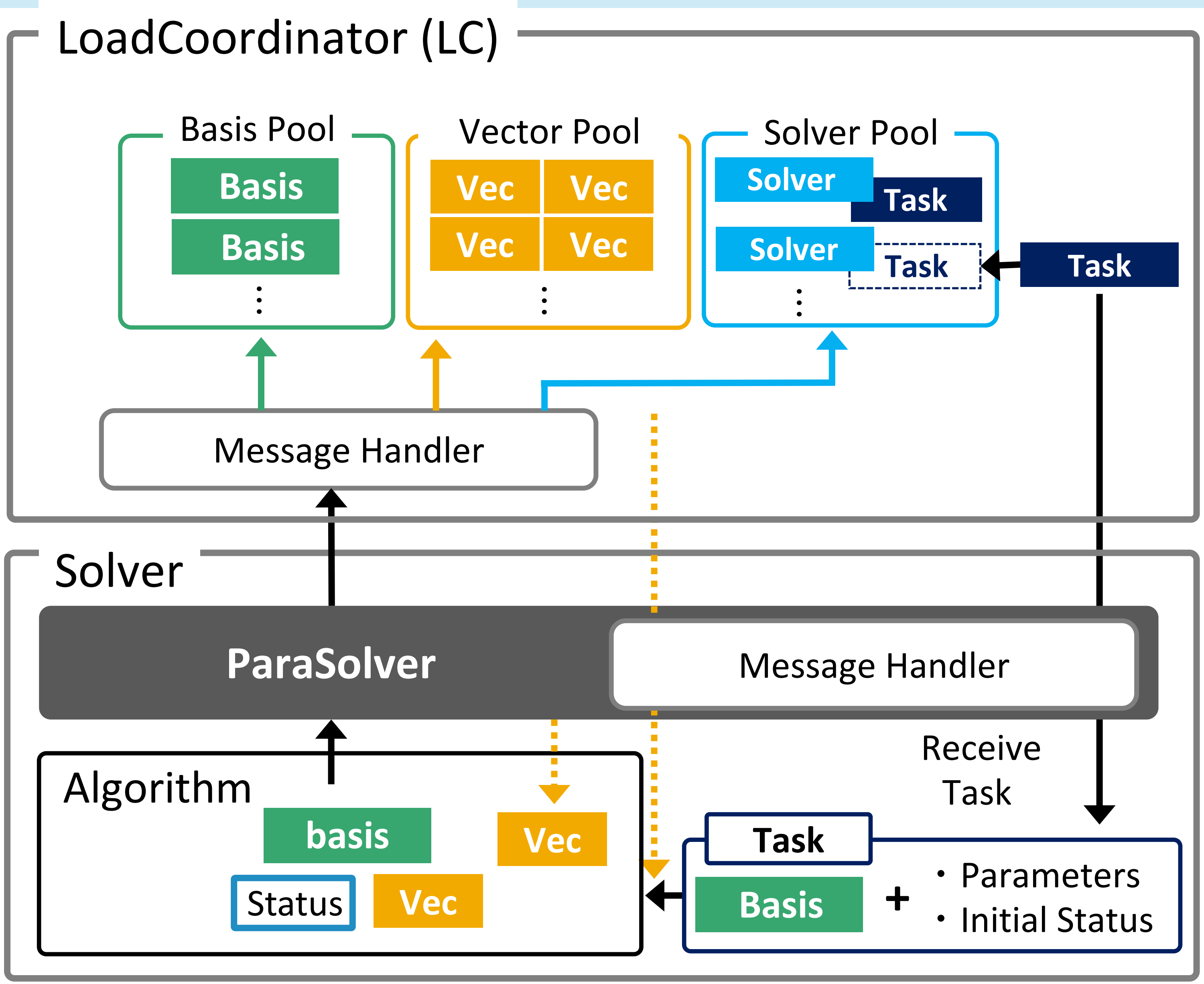
Generalized UG provides

- Customable and asynchronous **communication API** for Task and other information
- **Checkpointing** and **restart** functionality
- Both **MPI** and **Pthread** communicators can be selected, and hybrid parallelization is possible by combining them



Implementation of Our new solver

- Some data pool created in LoadCoordinator for sharing lattice basis and vector, and checkpointing



Outline

1. Contribution & Introduction
2. What is SVP?
3. Key components of parallelization
4. System of our solver based on UG
5. Numerical experiments
6. Summary

Topics

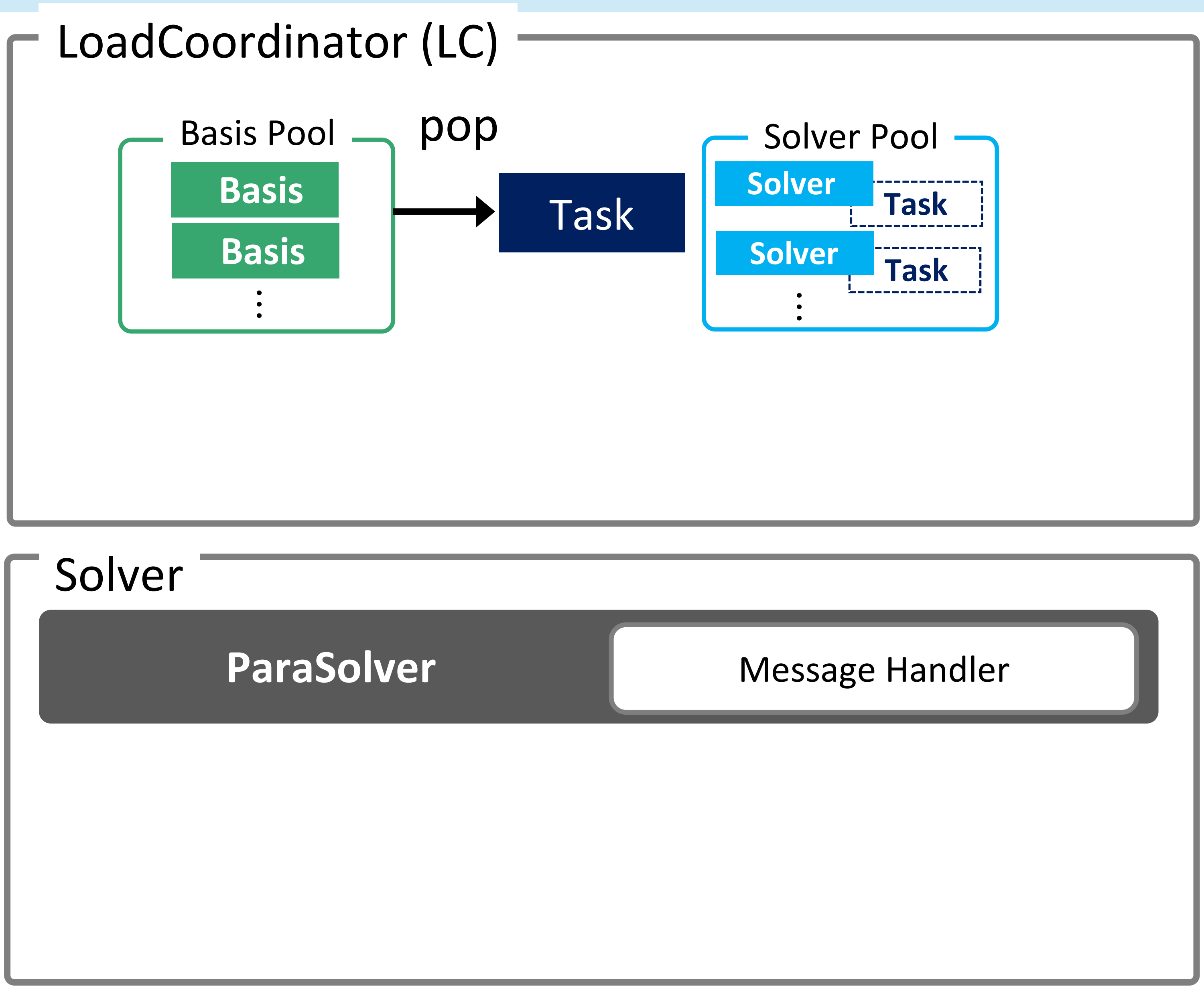
1. Overview
2. ▷ Communication of Task
3. Asynchronously Communication
4. Checkpointing

Task

Create Send Receive Task

Task is Triple of

- **Instance**
  - Basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$
- **Algorithm**
  - type of algorithm
- **Parameters**
  - Parameters change during execution of the algorithm

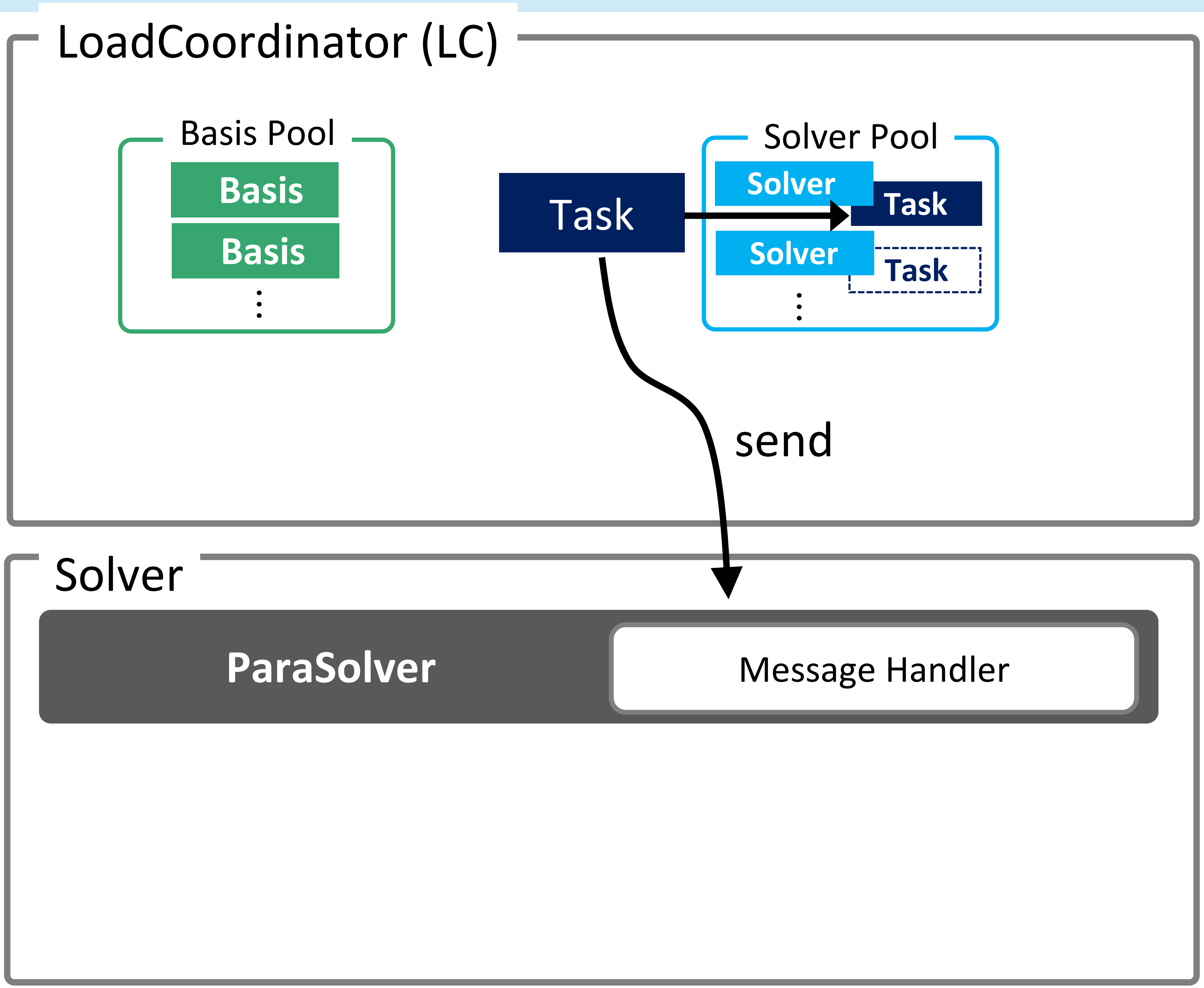


Task

Create Send Receive Task

Task is Triple of

- **Instance**
  - Basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$
- **Algorithm**
  - type of algorithm
- **Parameters**
  - Parameters change during execution of the algorithm



Asynchronously Communication

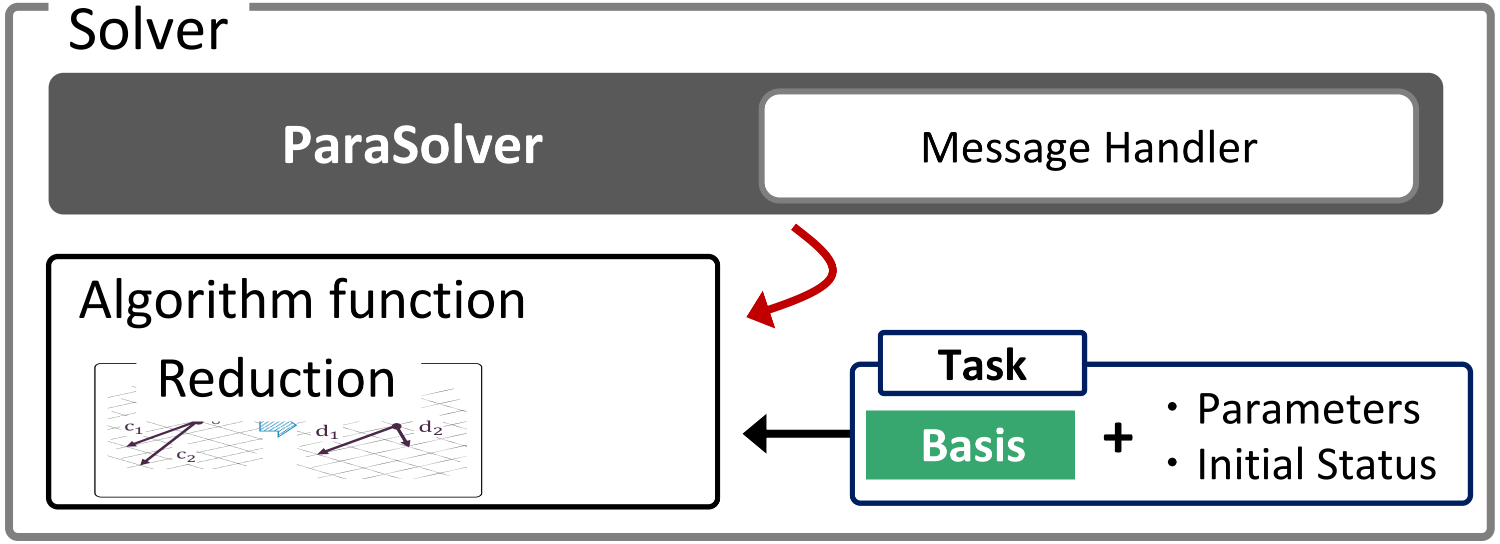
Create Send Receive Task

ParaSolver class object executes

```
runAlgorithm(
        basis, parameters, this)
```

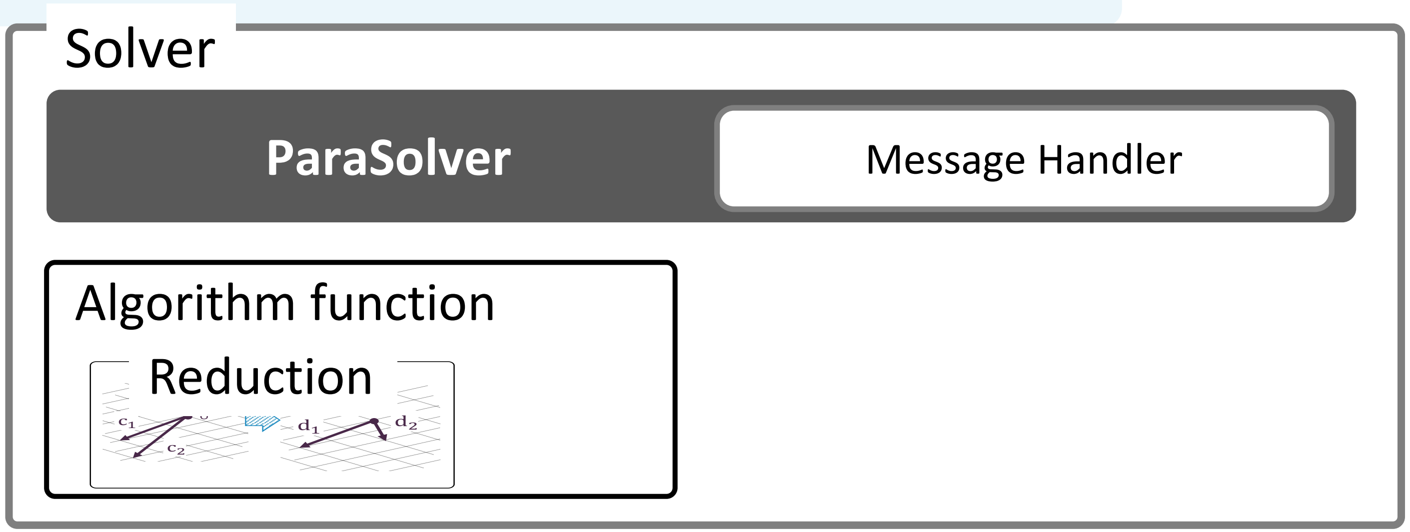
‖

ParaSolver object pointer



Asynchronously Communication

```
function runAlgorithm(basis, params, *paraSolver){
     while( algorithm is not finished ){
          runSubroutine(basis, params, paraSolver);
          communicateToLC(paraSolver);
          // send or receive lattice vectors and basis
          // asynchronously
     }
}
```



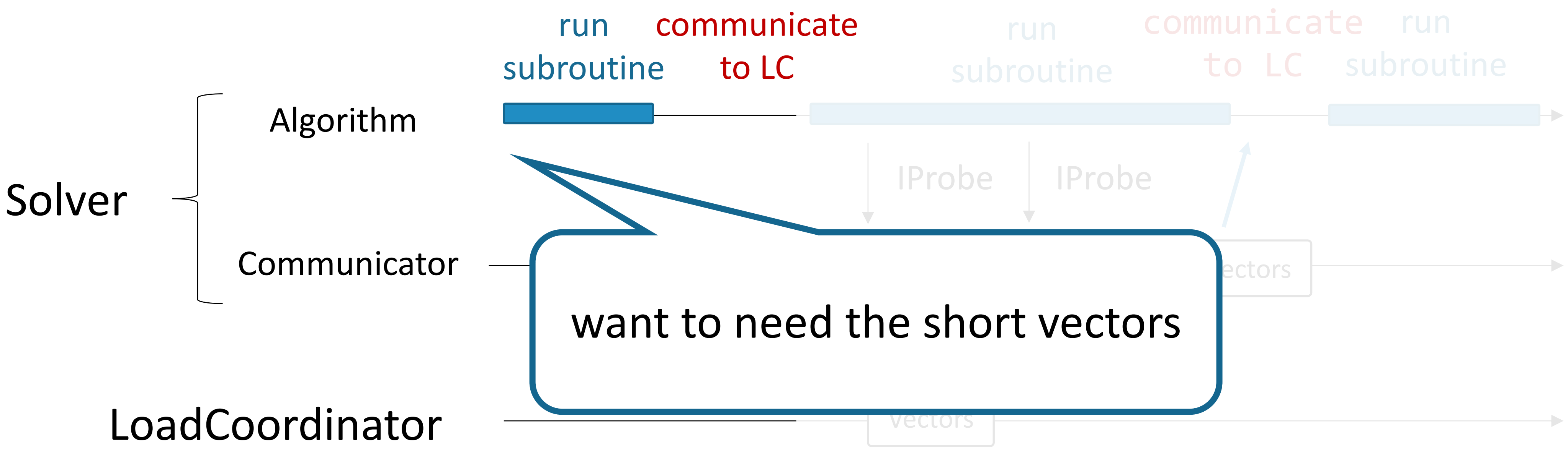
Outline

1. Contribution & Introduction
2. What is SVP?
3. Key components of parallelization
4. System of our solver based on UG
5. Numerical experiments
6. Summary

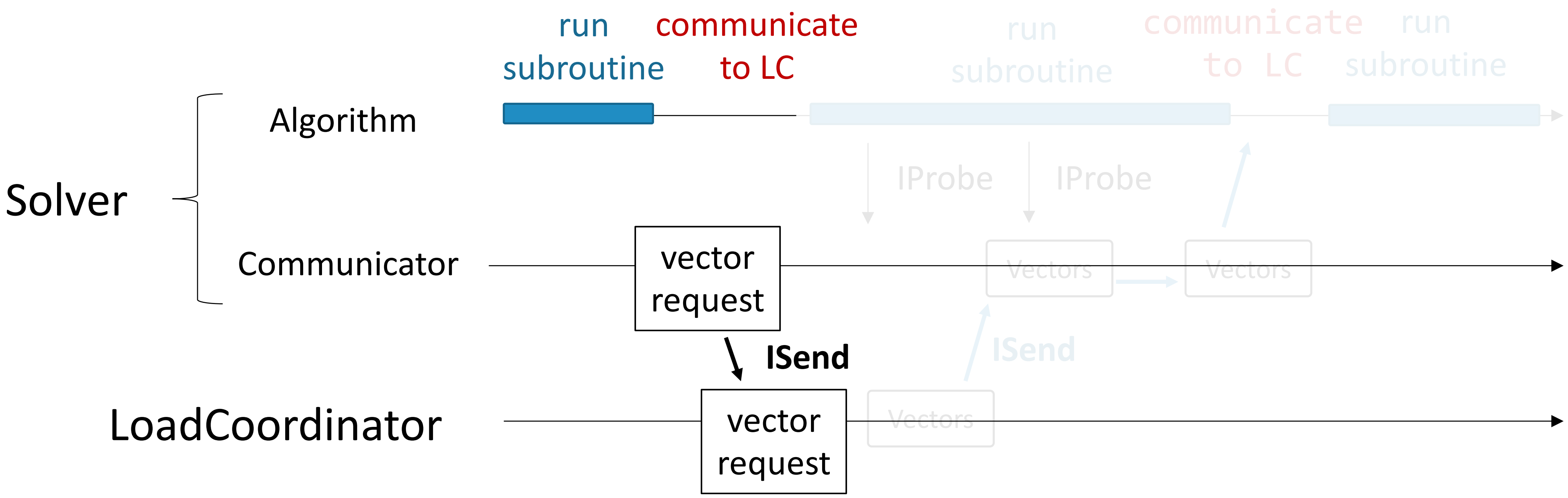
Topics

1. Overview
2. Communication of Task
3. ▷ Asynchronously Communication
4. Checkpointing

Asynchronously Communication – non-blocking version –

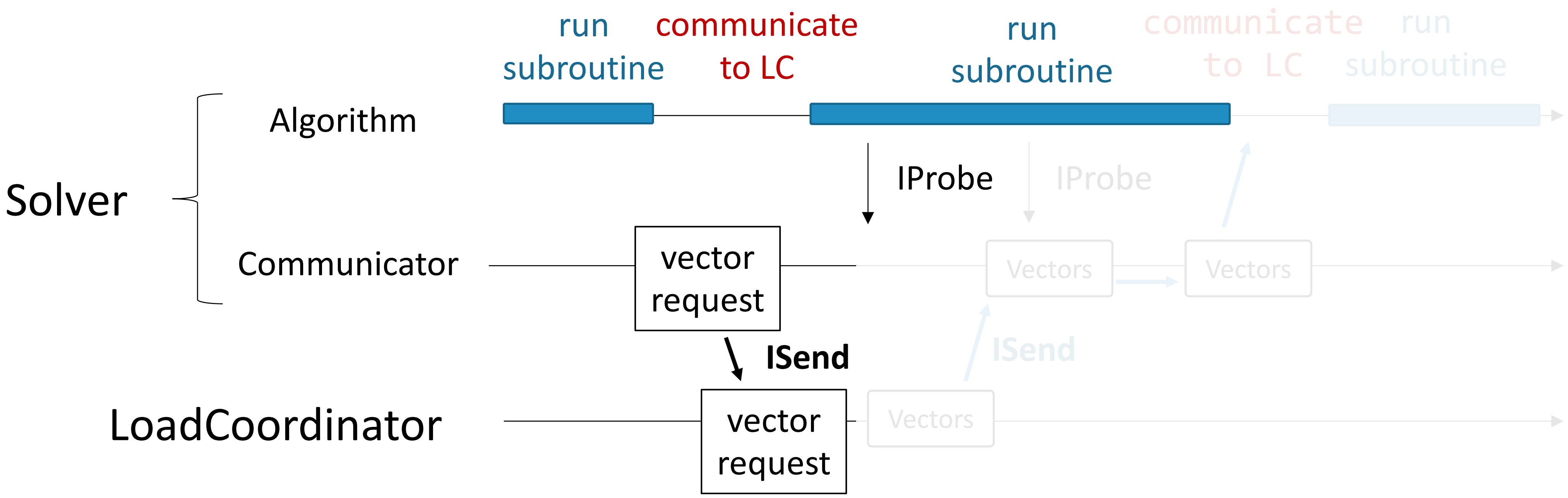


Asynchronously Communication – non-blocking version –



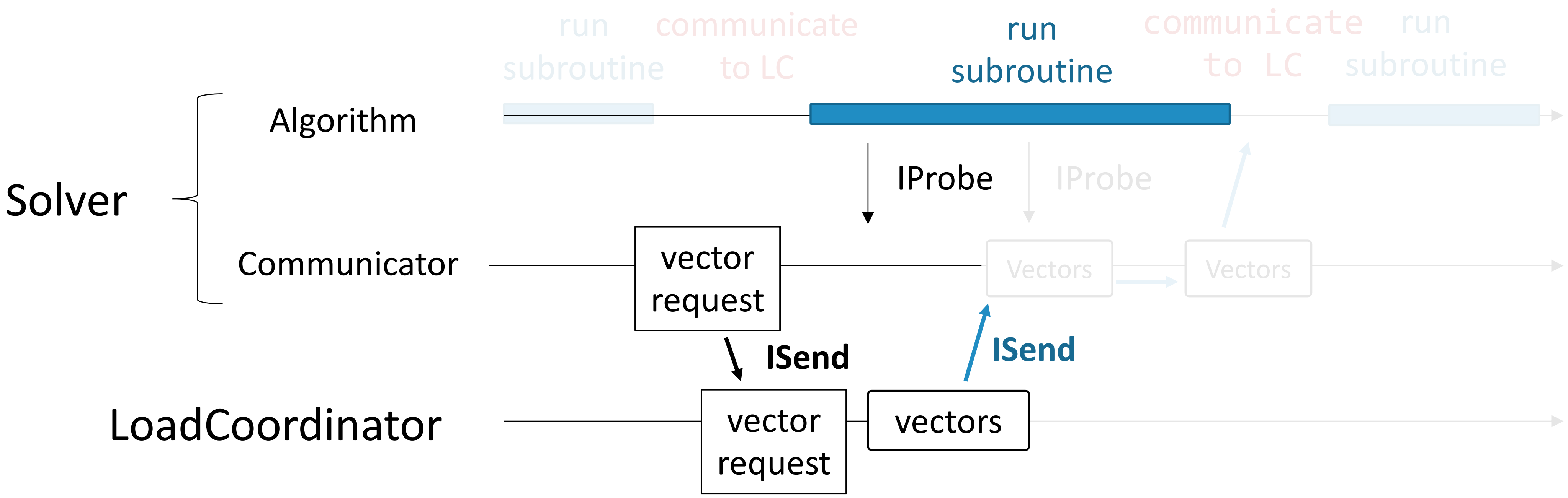
- communicator send “vector request” to LC by **`Isend (MPI_Isend)`**, which is non-blocking function

Asynchronously Communication – non-blocking version –



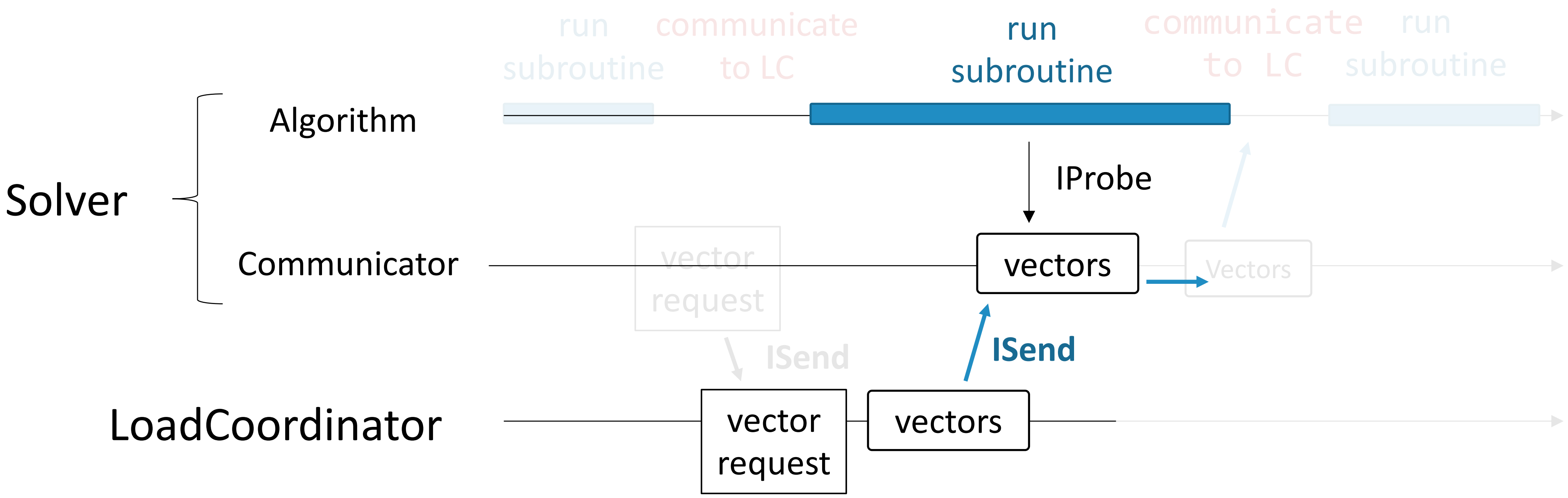
- Solver return to running algorithm
- In subroutine, **`iProbe (MPI_Iprobe)`** is called to check the message

Asynchronously Communication – non-blocking version –



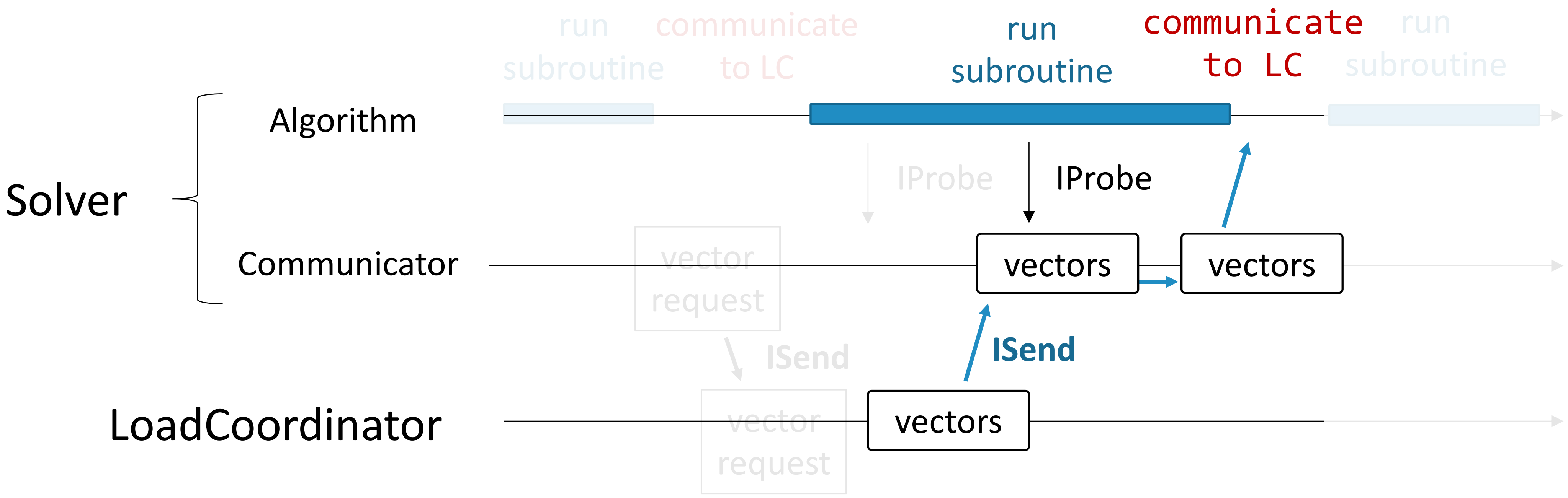
- LC prepare vectors according to the vector request
- LC send vectors by **`Isend (MPI_Isend)`**

Asynchronously Communication – non-blocking version –



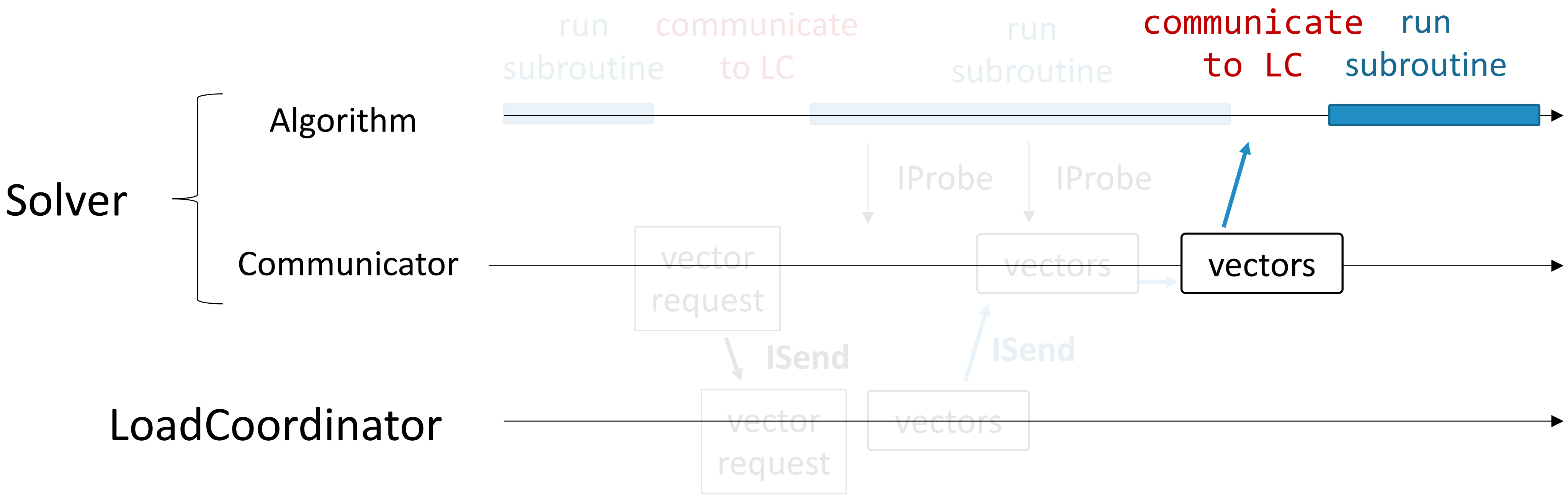
- communicator receives vectors and keep them

Asynchronously Communication – non-blocking version –



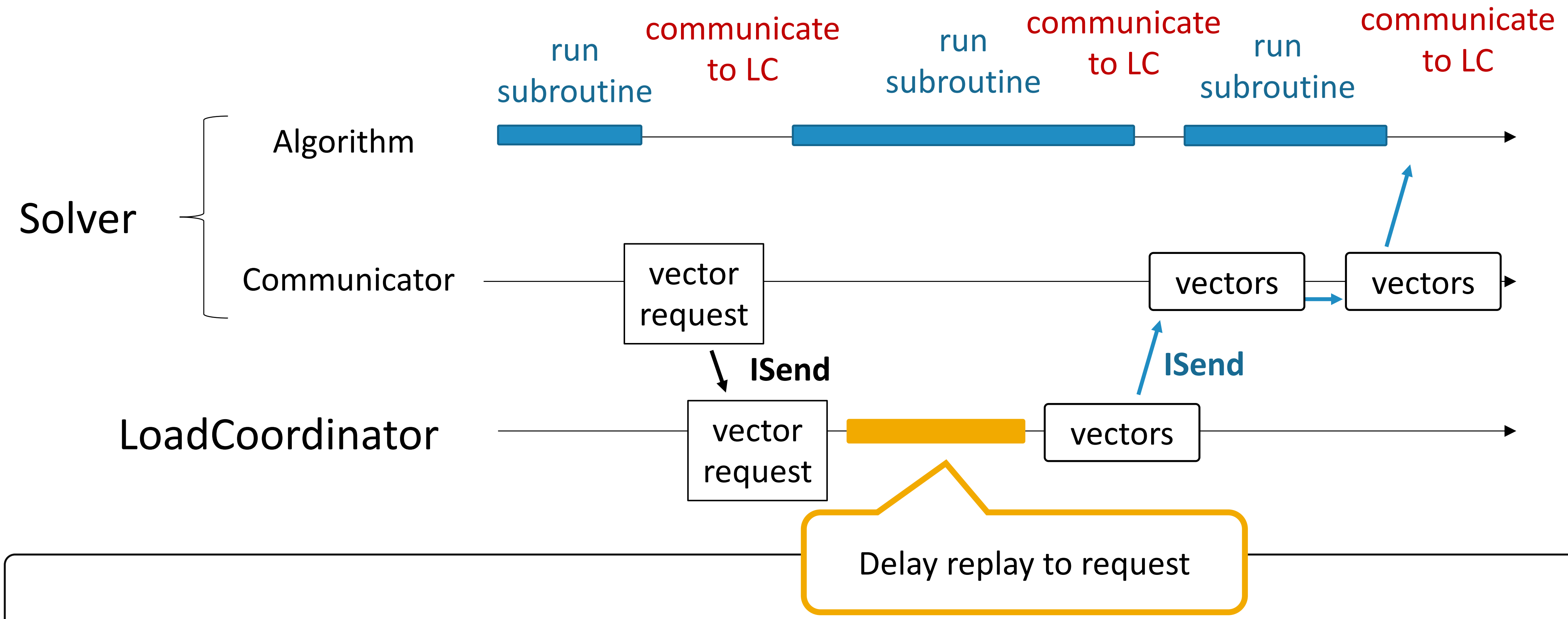
- In the following communication part,
  the algorithm can use the vector received by the communicator

Asynchronously Communication – non-blocking version –



- Then, Solver run subroutine again …

Asynchronously Communication – non-blocking version –



- Even if LC delays in replying to a vector request,
  algorithm can receive vectors more next communication part
- **SVP algorithm can incorporate vectors at any time**

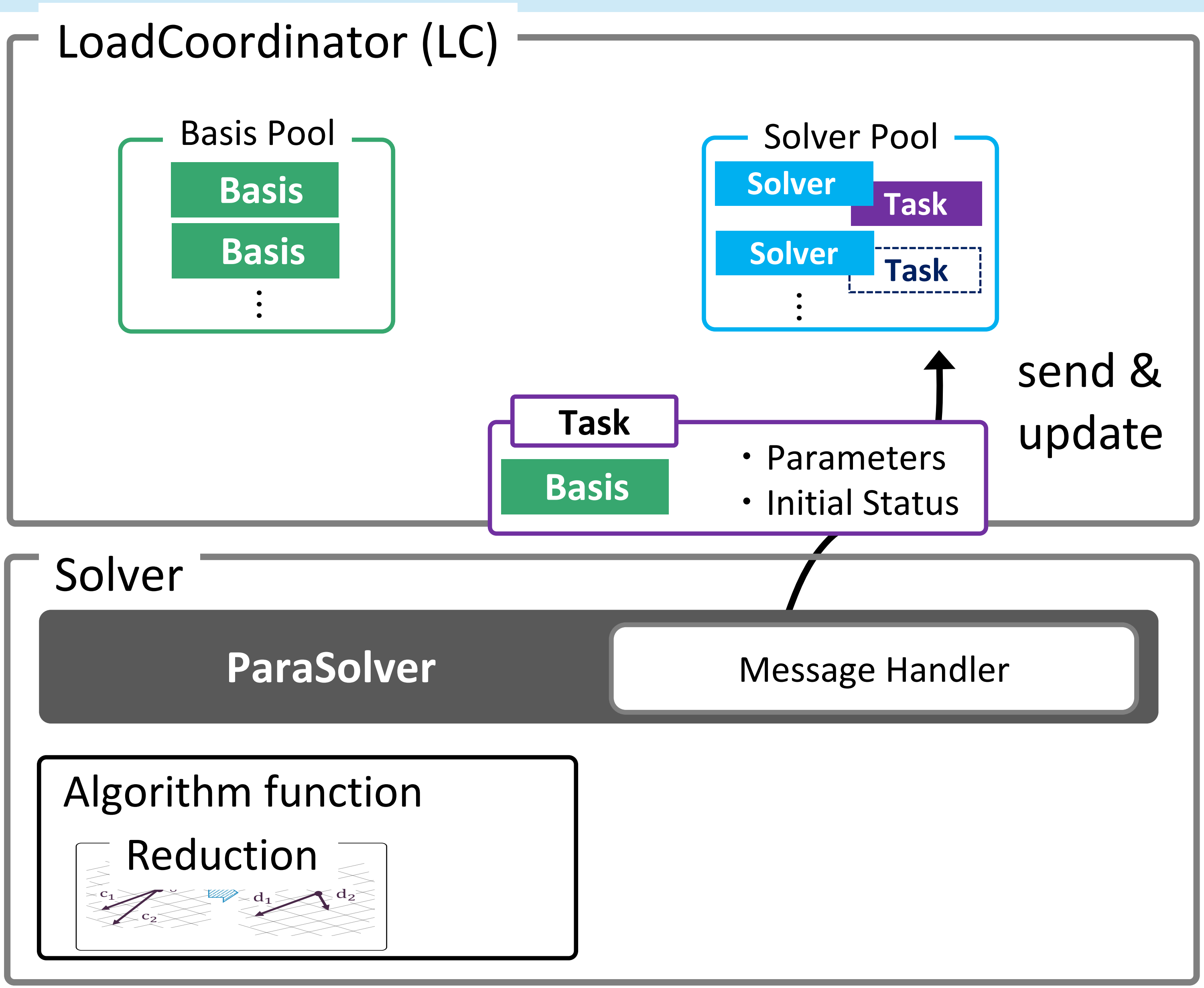
Outline

<u>Topics</u>

1. Overview
2. Communication of Task
3. Asynchronously Communication
4. ▷ Checkpointing

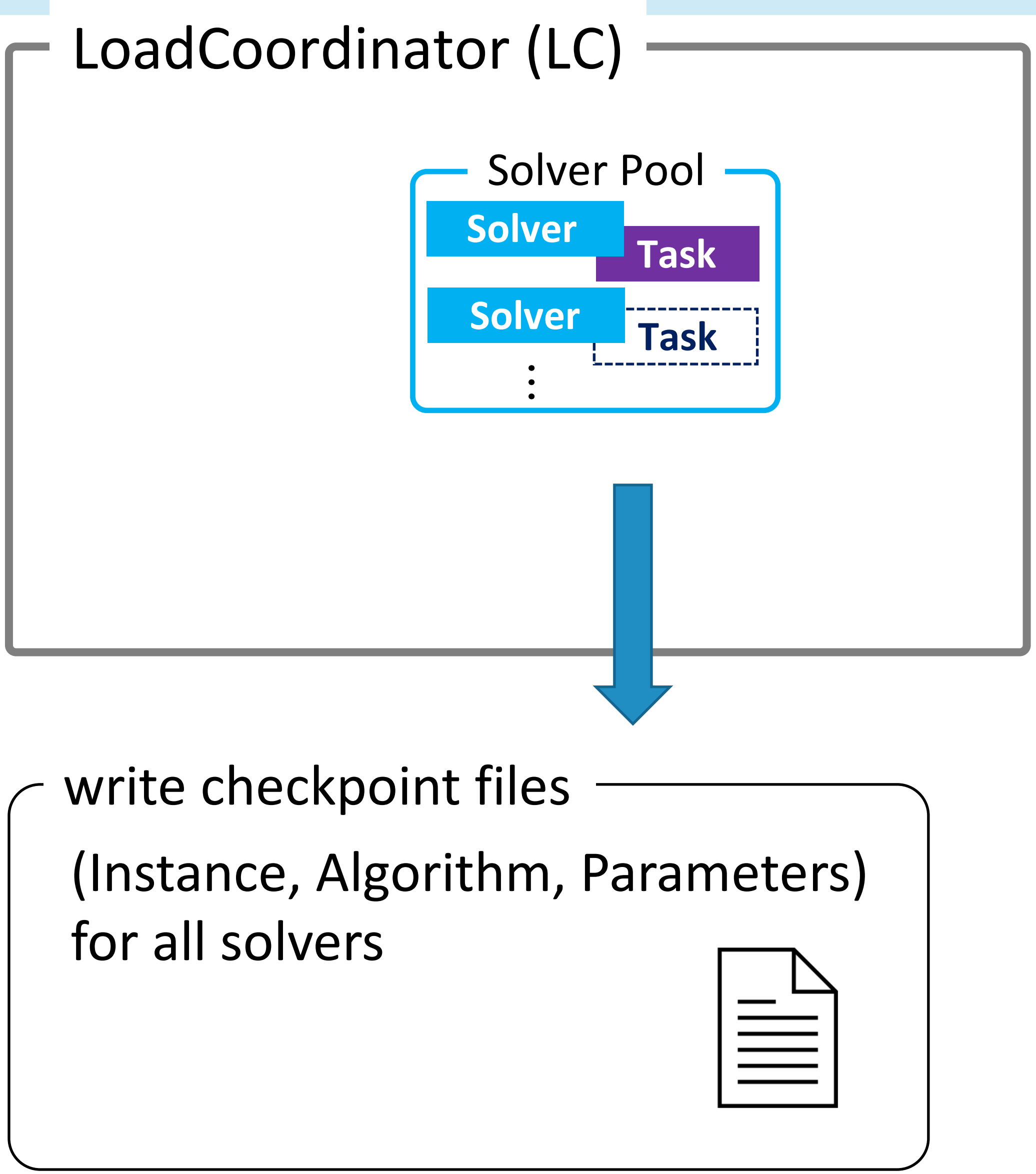
Checkpointing

- Solver update task according to the progress of algorithm
- Solver send update Task to LC, and LC replaces it from old task in solver pool



Checkpointing

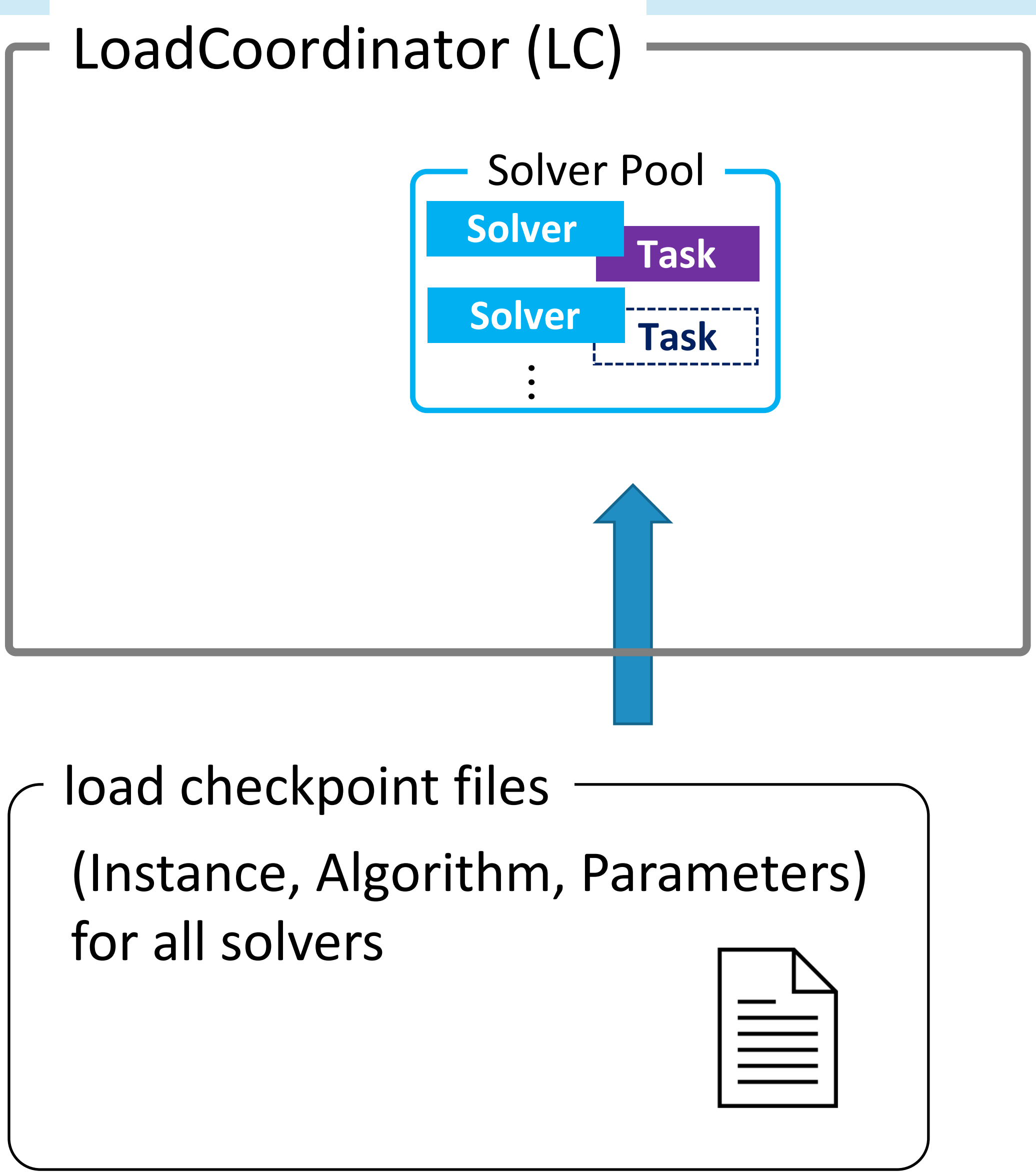
- Write compressed data in Solver Pool into checkpointing files, and data in other pools write to files, too



Checkpointing

Restart

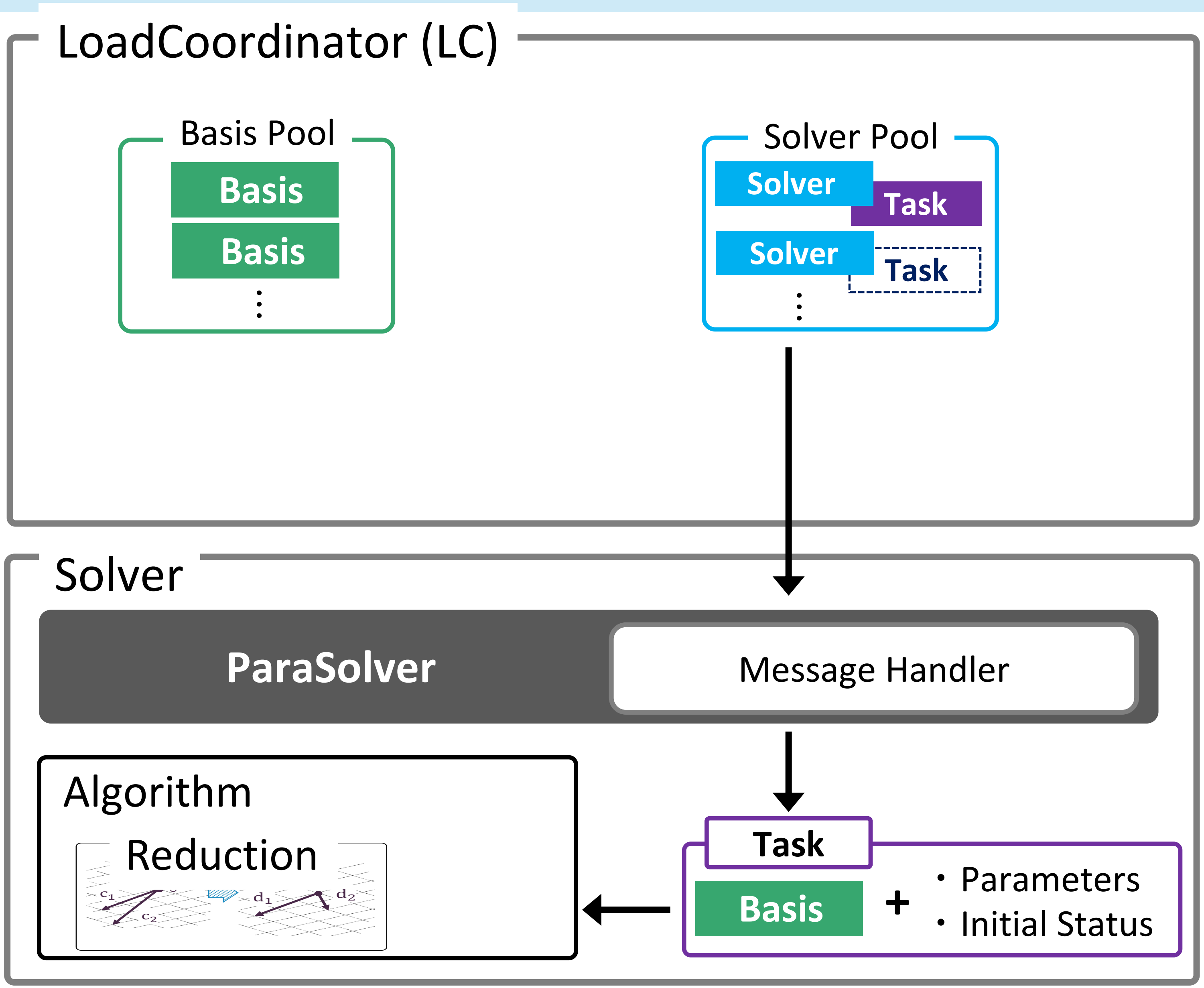
Load checkpoint file and store data into solver pool



Checkpointing

Restart

Load checkpoint file and store data into solver pool



Outline

Numerical Experiments

New solution for SVP Challenge

- CMAP-LAP had succeeded in finding shorter lattice vectors in **104**, **111**, **121**, and **127** dimensions of the SVP Challenge.
- CMAP-LAP finds a sufficiently short vector in a reasonably short time.
  - The G6K, a famous SVP solver, reported taking 14 days = 336 hours to find a sufficiently short vector for a 127-dimensional lattice

TABLE II
NEW SOLUTIONS FOR THE HALL OF FAME IN THE SVP CHALLENGE [3], FOUND BY CMAP-LAP

Dim.	Seed	Norm	App. factor	#Process	Total time
104	35	2516	0.97173	120	551 seconds
	85	2520	0.97010	120	214 seconds
	82	2529	0.97719	120	432 seconds
111	29	2597	0.96979	2000	792 seconds
	30	2635	0.98382	2000	541 seconds
	8	2660	0.99467	2000	611 seconds
121	4	2780	0.99706	2304	682 minutes
	2	2809	1.00820	2304	481 minutes
127*	3*	2790	0.97573	91,200	147 hours
	1†	2890	1.01429	9,980	31 hours
	0†	2898	1.01626	49,152	25 hours

*† We executed the CMAP-LAP several times on multiple computers, as described in paragraph V-D0a. We list the maximum number of processes and total approximate wall time among these executions in the table.

† These solutions are not new records, but they are the same solution as the previous record or very nearly close to it.

App. factor := approximation factor of the incumbent lattice vector

Numerical Experiments

New solution for SVP Challenge

- CMAP-LAP had succeeded in finding shorter lattice vectors in **104**, **111**, **121**, and **127** dimensions of the SVP Challenge.
- CMAP-LAP finds a sufficiently short vector in a reasonably short time.
  - The G6K, a famous SVP solver, reported taking 14 days = 336 hours to find a sufficiently short vector for a 127-dimensional lattice

TABLE II
NEW SOLUTIONS FOR THE HALL OF FAME IN THE SVP CHALLENGE [3], FOUND BY CMAP-LAP

Dim.	Seed	Norm	App. factor	#Process	Total time
104	35	2516	0.97173	120	551 seconds
	85	2520	0.97010	120	214 seconds
	82	2529	0.97719	120	432 seconds
111	29	2597	0.96979	2000	792 seconds
	30	2635	0.98382	2000	541 seconds
	8	2660	0.99467	2000	611 seconds
121	4	2780	0.99706	2304	682 minutes
	2	2809	1.00820	2304	481 minutes
127*	3*	2790	0.97573	91,200	147 hours
	1†	2890	1.01429	9,980	31 hours
	0†	2898	1.01626	49,152	25 hours

*† We executed the CMAP-LAP several times on multiple computers, as described in paragraph V-D0a. We list the maximum number of processes and total approximate wall time among these executions in the table.

† These solutions are not new records, but they are the same solution as the previous record or very nearly close to it.

Numerical Experiments

New solution for SVP Challenge

- CMAP-LAP had succeeded in finding shorter lattice vectors in **104**, **111**, **121**, and **127** dimensions of the SVP Challenge.
- CMAP-LAP finds a sufficiently short vector in a reasonably short time.
  - The G6K, a famous SVP solver, reported taking 14 days = 336 hours to find a sufficiently short vector for a 127-dimensional lattice

TABLE II
NEW SOLUTIONS FOR THE HALL OF FAME IN THE SVP CHALLENGE [3], FOUND BY CMAP-LAP

Dim.	Seed	Norm	App. factor	#Process	Total time
104	35	2516	0.97173	120	551 seconds
	85	2520	0.97010	120	214 seconds
	82	2529	0.97719	120	432 seconds
111	29	2597	0.96979	2000	792 seconds
	30	2635	0.98382	2000	541 seconds
	8	2660	0.99467	2000	611 seconds
121	4	2780	0.99706	2304	682 minutes
	2	2809	1.00820	2304	481 minutes
127*	3*	2790	0.97573	91,200	147 hours
	1†	2890	1.01429	9,980	31 hours
	0†	2898	1.01626	49,152	25 hours

*† We executed the CMAP-LAP several times … multiple computers, as described in paragraph V-D0a. We li… m number of p… i…

† T… a…

max #process is 91,200 in HLRN

Numerical Experiments

New solution for SVP Challenge

- CMAP-LAP had succeeded in finding shorter lattice vectors in **104**, **111**, **121**, and **127** dimensions of the SVP Challenge.
- CMAP-LAP finds a sufficiently short vector in a reasonably short time.
  - The G6K, a famous SVP solver, reported taking 14 days = 336 hours to find a sufficiently short vector for a 127-dimensional lattice

TABLE II
NEW SOLUTIONS FOR THE HALL OF FAME IN THE SVP CHALLENGE [3], FOUND BY CMAP-LAP

Dim.	Seed	Norm	App. factor	#Process	Total time
104	35	2516	0.97173	120	551 seconds
	85	2520	0.97010	120	214 seconds
	82	2529	0.97719	120	432 seconds
111	29	2597	0.96979	2000	792 seconds
	30	2635	0.98382	2000	541 seconds
	8	2660	0.99467	2000	611 seconds
121	4	2780	0.99706	2304	682 minutes
	2	2809	1.00820	2304	481 minutes
127*	3*	2790	0.97573	91,200	147 hours
	1†	2890	1.01429	9,980	31 hours
	0†	2898	1.01626	49,152	25 hours

*† We executed the CMA… P several times on multiple computers, as described i… D0a. We list the maximum number of p… i…

† T… a…

App. factor := estimated approximation factor of the incumbent solution

Numerical Experiments

Scalability

- Scalability keeps even in the larger-scale, e.g., 100,032 processes

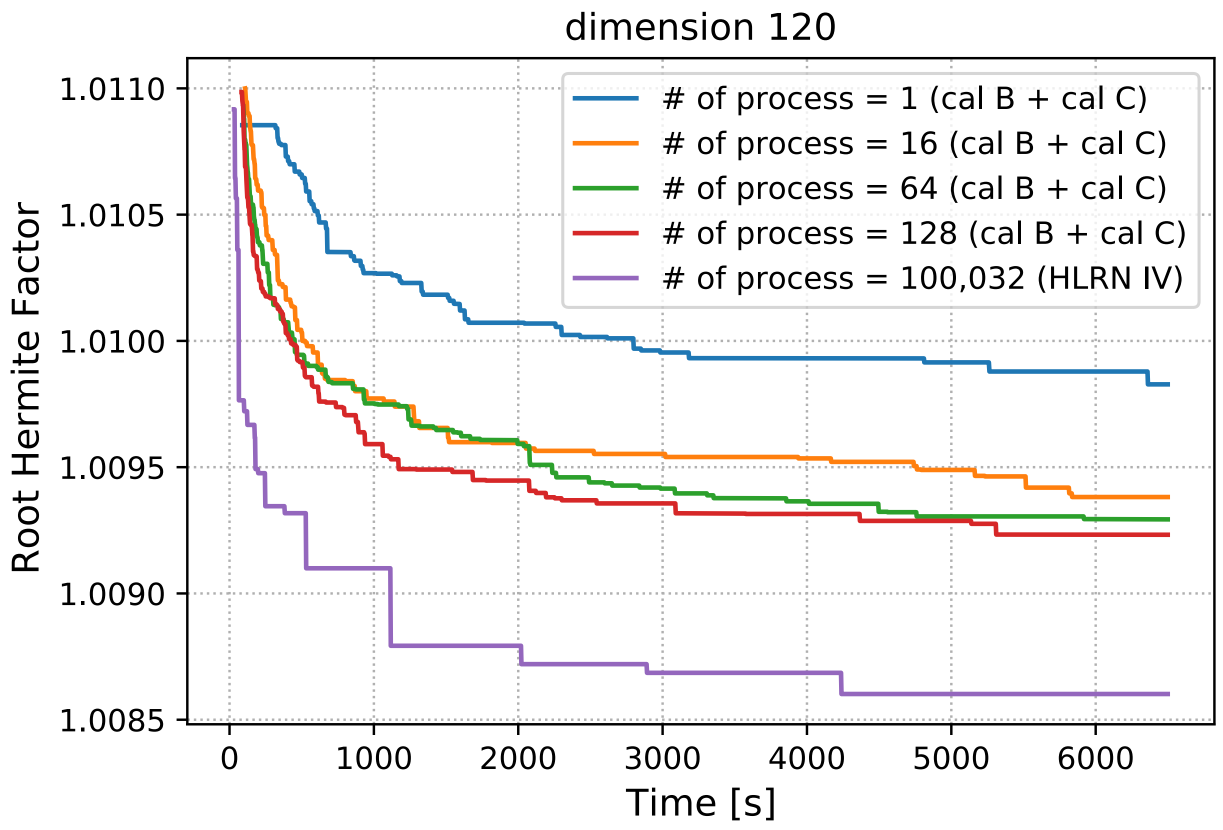
- Metric is **Root Hermite Factor $\gamma^{1/n}$**, which is an index to measure the output quality of a reduction algorithm

$$\gamma^{1/n} := \left(\frac{\|\mathbf{b}\|}{\mathrm{vol}(L)^{1/n}}\right)^{1/n}$$

  where $\mathbf{b}$ is the shortest basis vector output by reduction algorithm.

- Smaller $\gamma^{1/n}$ means that output quality is good and find shorter vector

- All solver run Reduction (DeepBKZ) algorithm



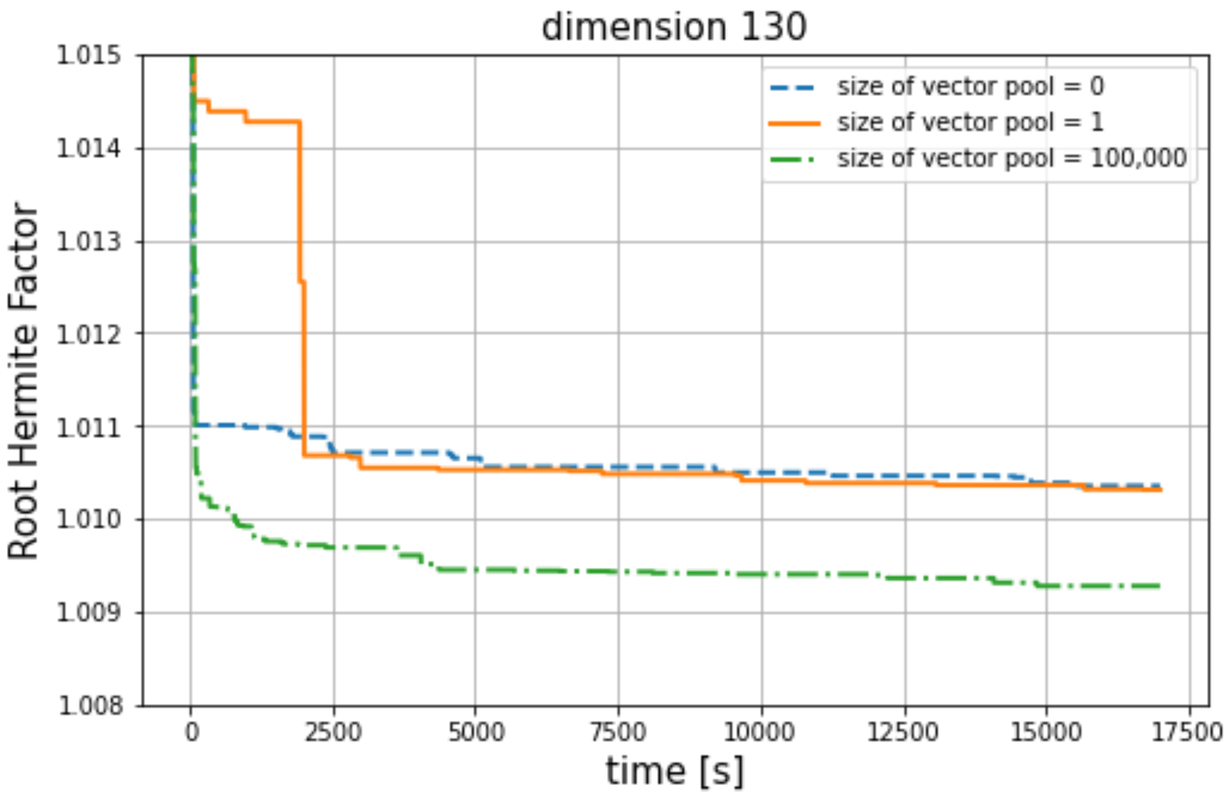
Numerical Experiments

Sharing efficiency

- Execute with different vector pool size
  - Size of pool = 0 (blue)
    ⇔ no sharing
  - Size of pool = 1 (orange)
    ⇔ sharing only incumbent vector
  - Size of pool = 100,000 (green)
    ⇔ sharing almost all short vectors

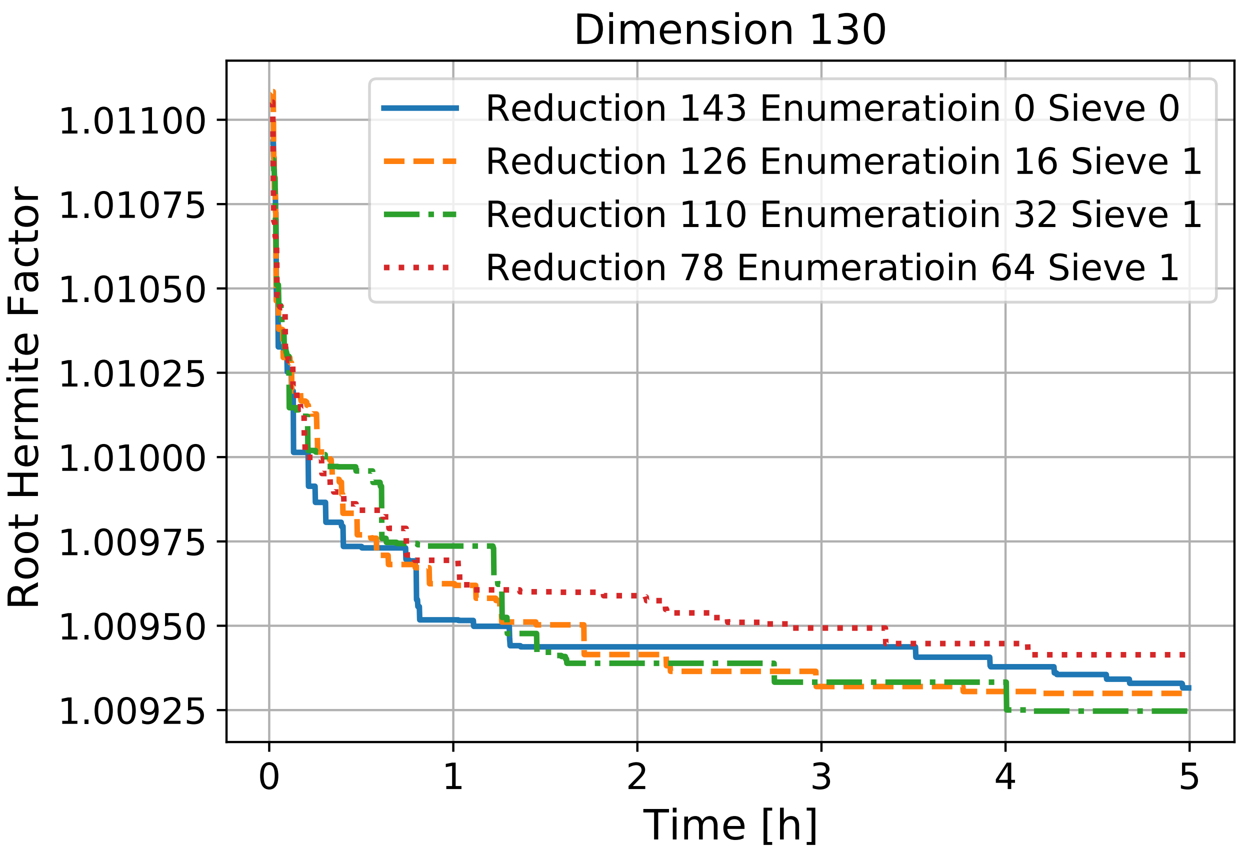
- All solver run Reduction (DeepBKZ) algorithm

- With the effect of sharing vectors,
  the Root Hermite Factor could get smaller



Heterogeneously efficiency

- Execute with different algorithm configuration
- #(Reduction, Enumeration, Sieve)
  - = (143, 0, 0) (blue)
  - = (126, 16, 1) (orange)
  - = (110, 32, 1) (green)
  - = (78, 64, 1) (red)



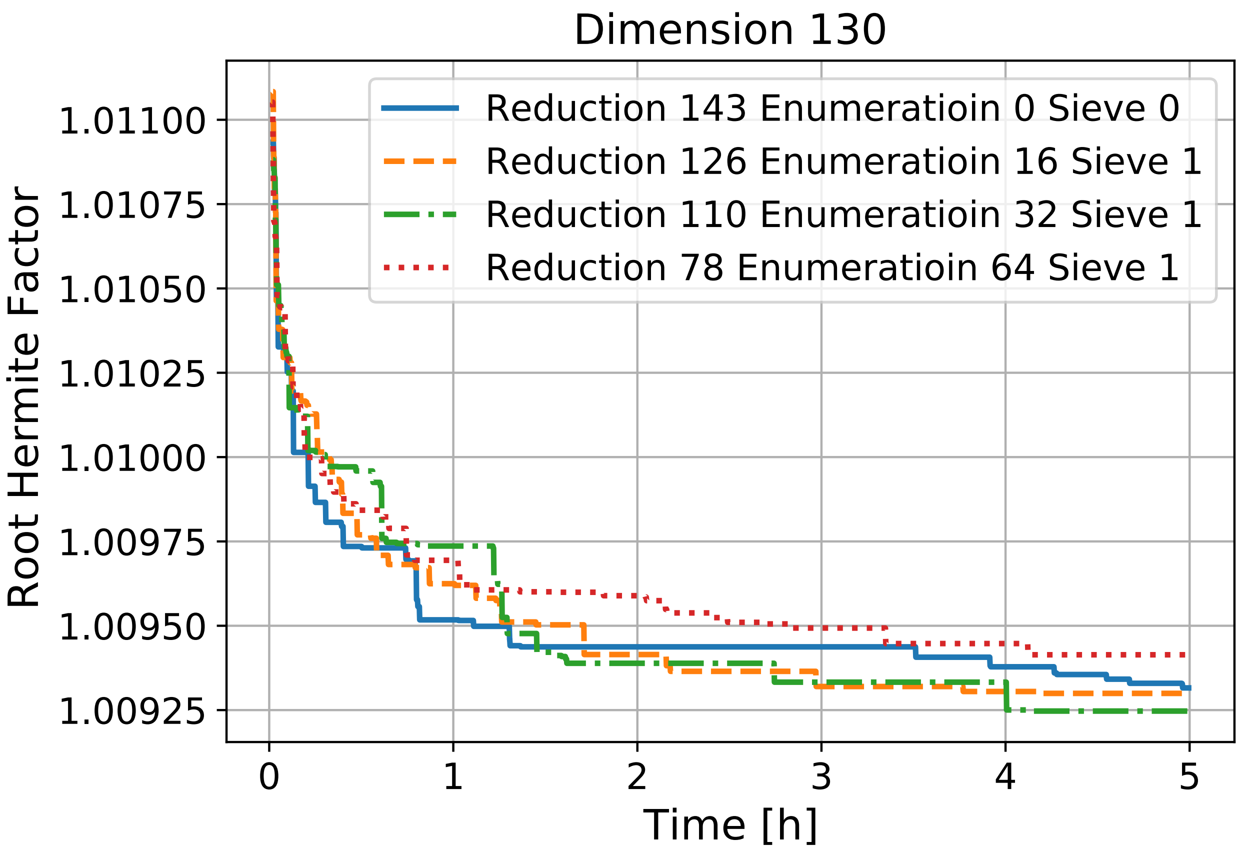
Numerical Experiments

Heterogeneously efficiency

- Execute with different algorithm configuration
- #(Reduction, Enumeration, Sieve)
  - = (143, 0, 0) (blue)
  - = (126, 16, 1) (orange)
  - = (110, 32, 1) (green)
  - = (78, 64, 1) (red)

- There was difference in results depending on the configuration
- For further improvement, it is necessary
  - tuning parameters
  - dynamic modification of the configuration

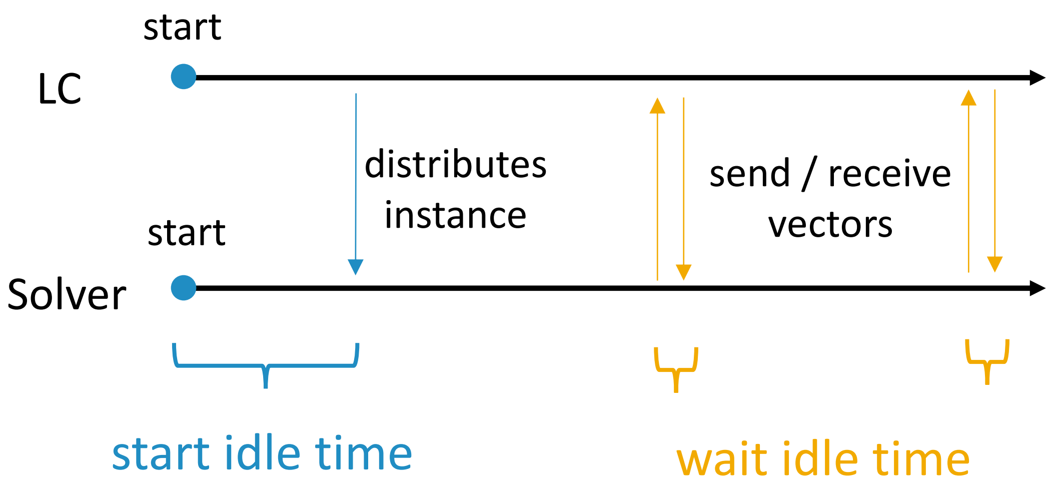
Future work

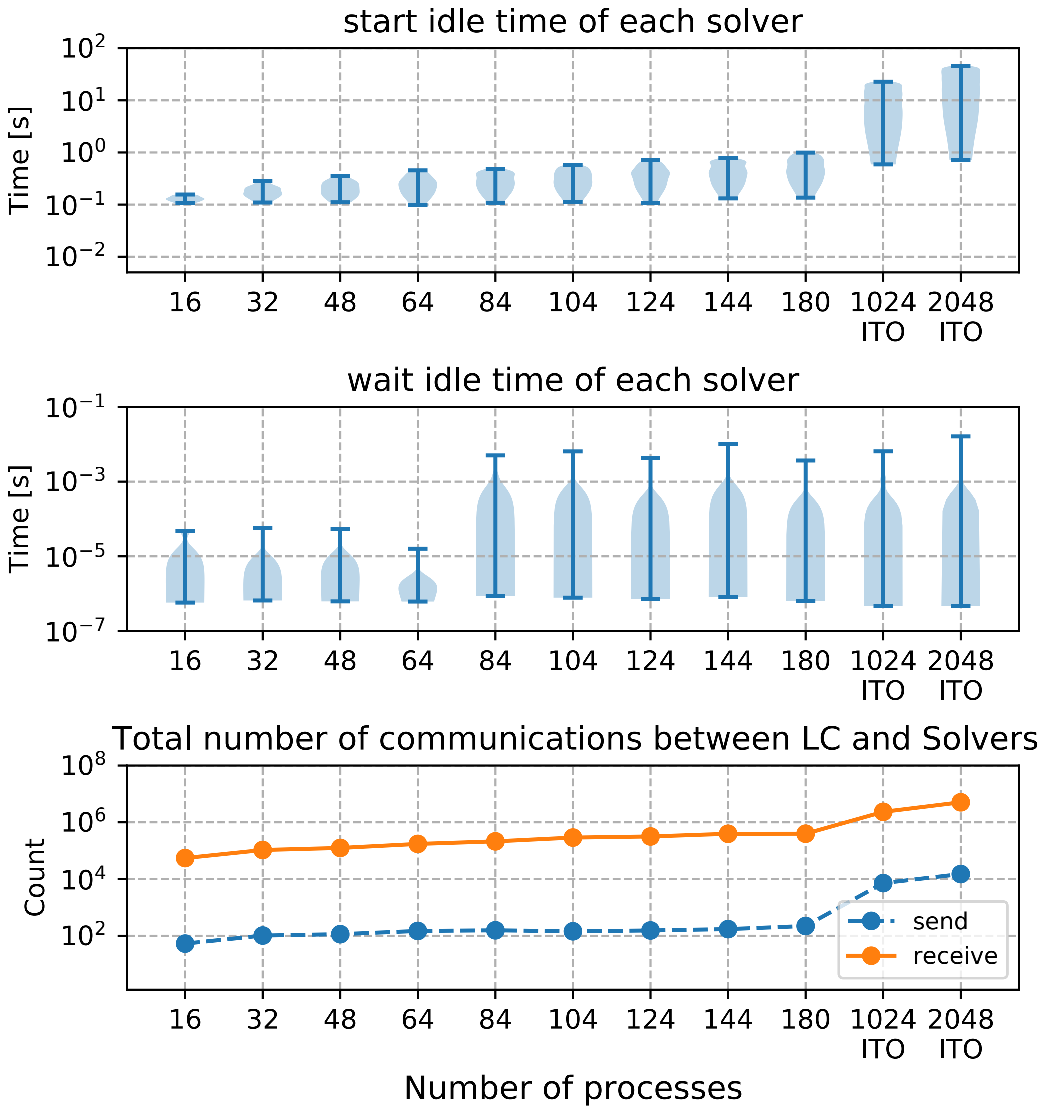


Numerical Experiments: communication overhead

Parallelization performance of UG

- Definition of idle times

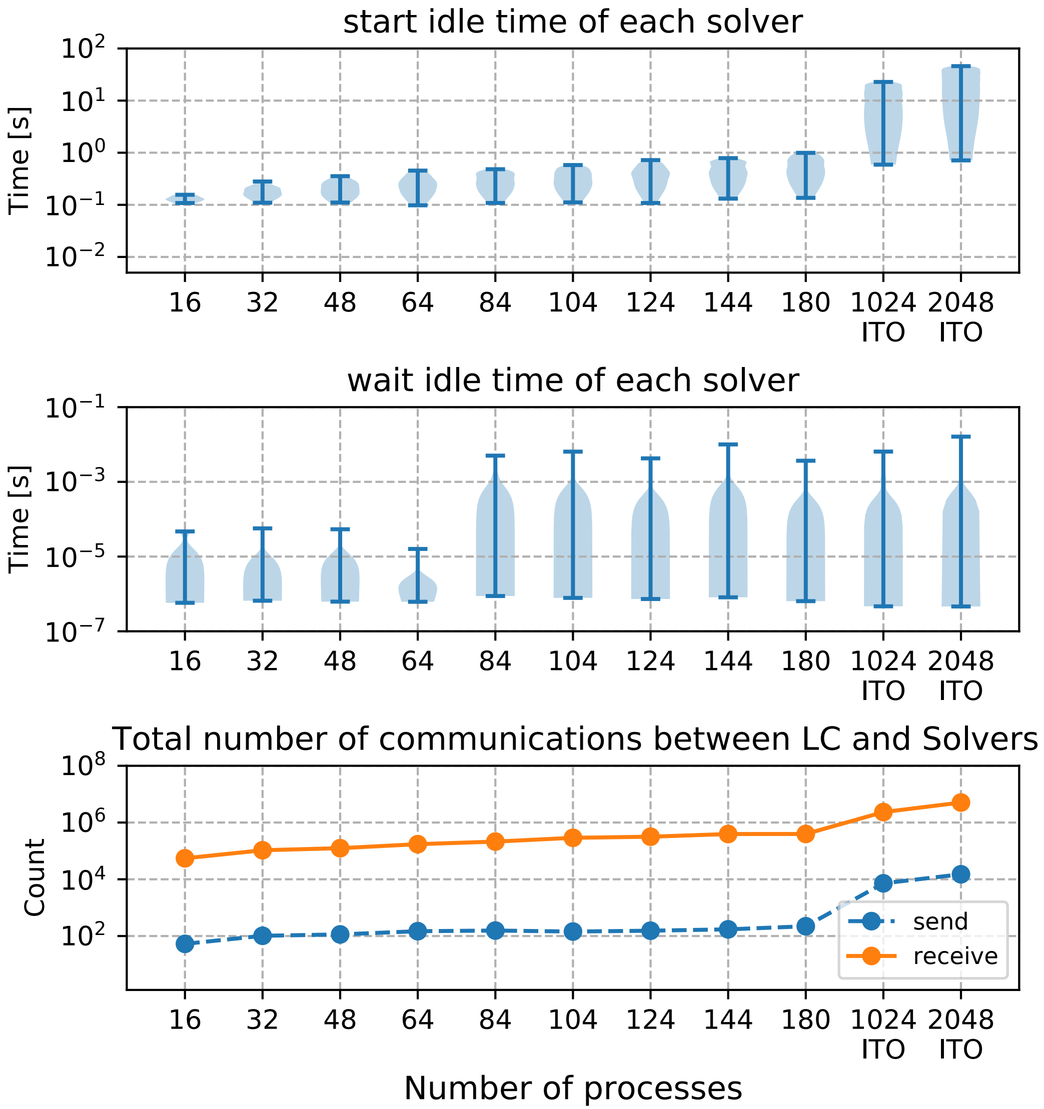




Numerical Experiments: communication overhead

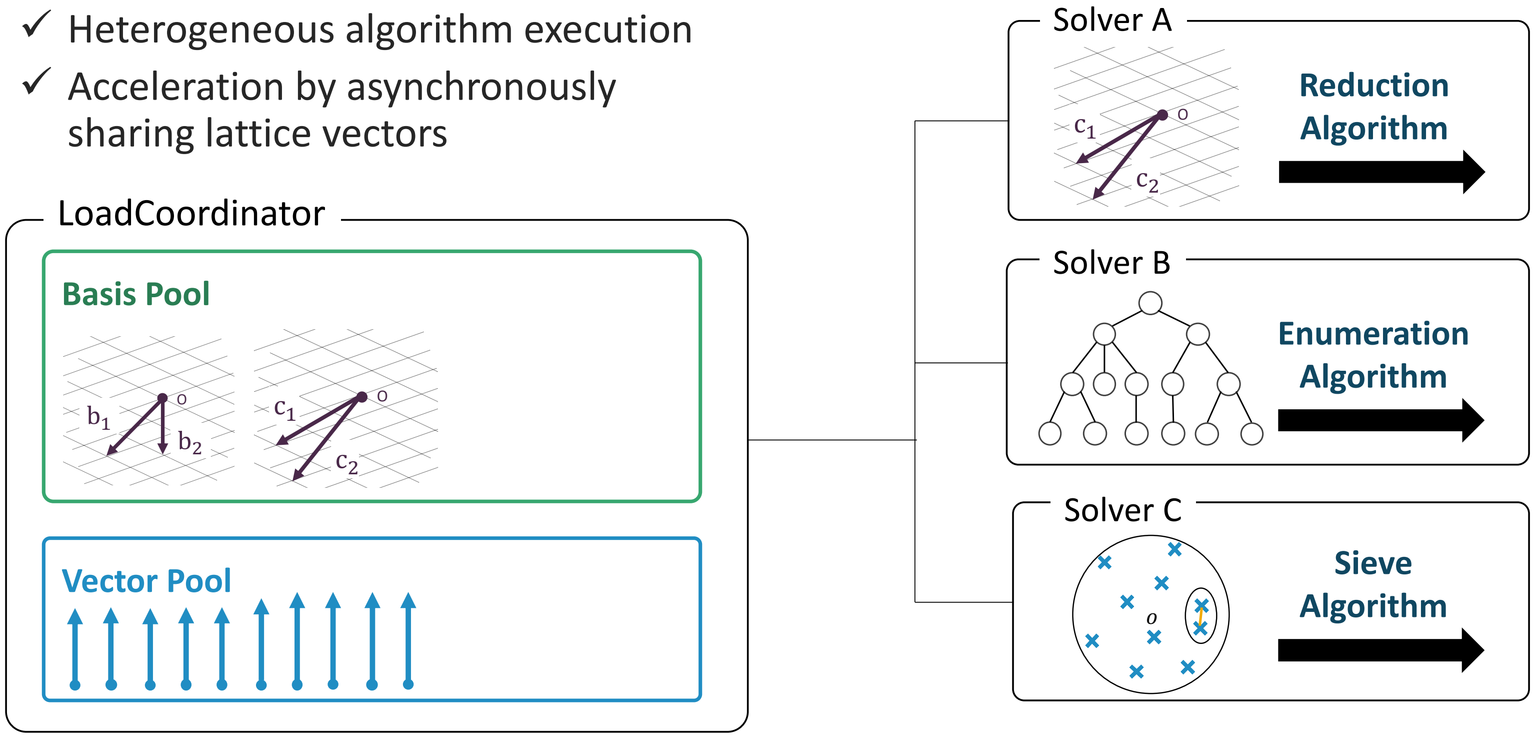
Parallelization performance of UG

- CMAP-LAP ran for two hours with 100-dimensional SVP as input
- As the number of processes increases, the LC’s load increases
- Although the LC's load increases, idle time is much less than the total running time per hour.



Summary

- ✓ We developed a parallel solver for SVP based on Generalized UG
  - ✓ Supervisor-Worker parallelization type
  - ✓ Heterogeneous algorithm execution
  - ✓ Acceleration by asynchronously sharing lattice vectors



- ✓ Update some SVP Challenge records
- ✓ Show scalability and low communication overhead sharing and heterogeneously efficiency of our solver