Improving an Iterative Physics Solver Using a Direct Method

Maciej Mizerski
Technical Director, Roblox

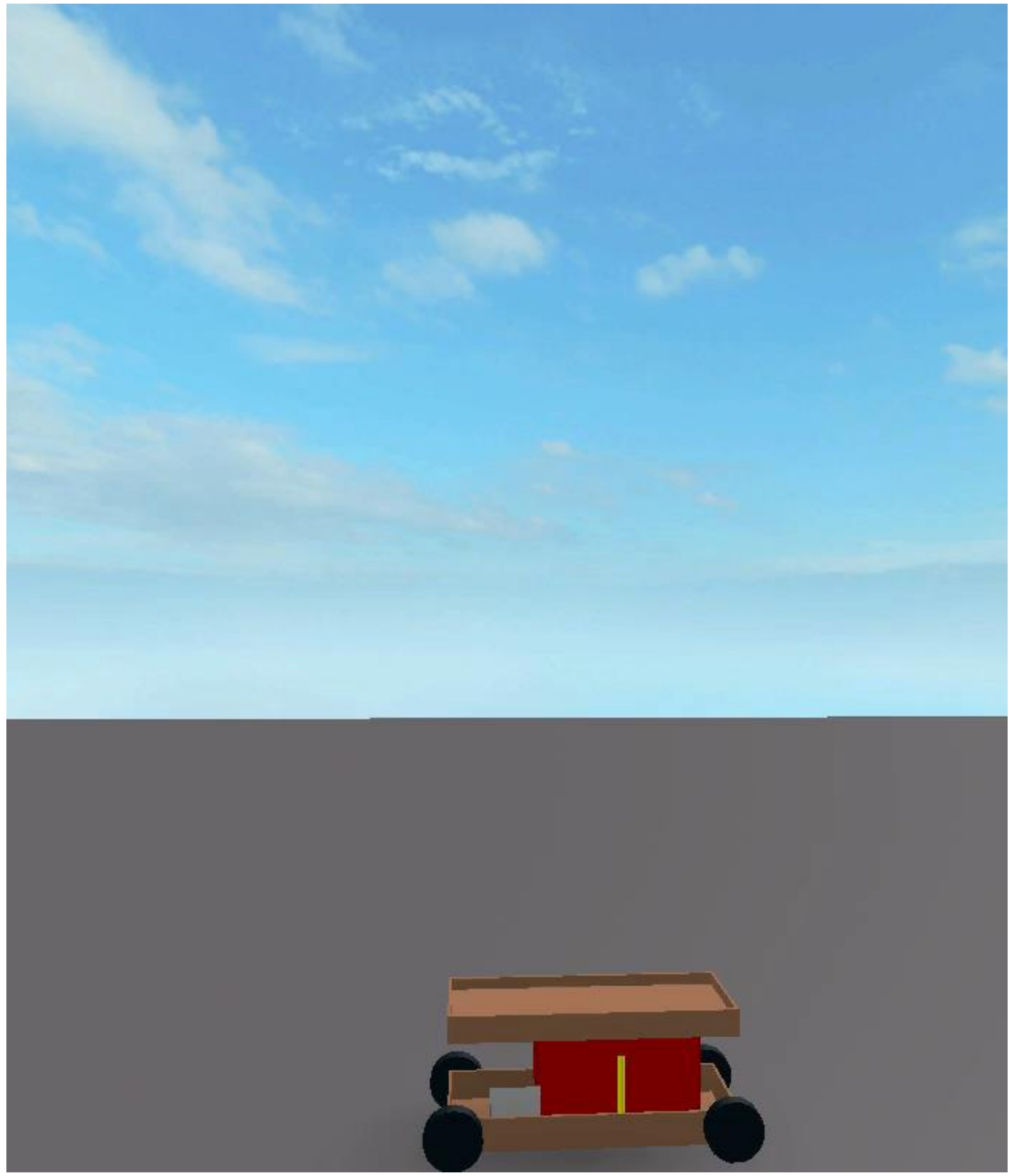
GAME DEVELOPERS CONFERENCE
MARCH 16–20, 2020 | #GDC20

ROBLOX

Physics Solver in Roblox

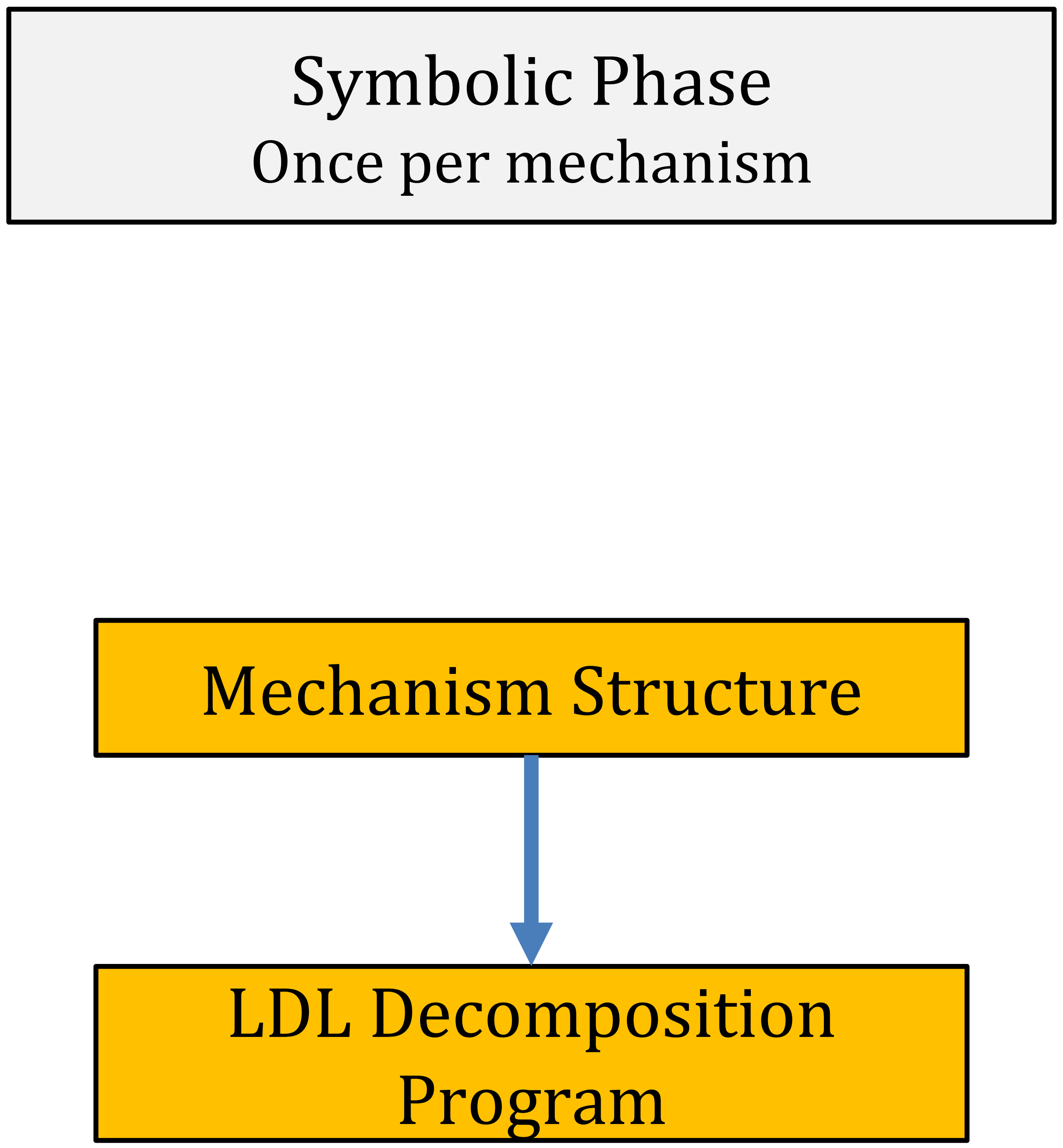
Computes the motion of <u>constrained</u> rigid bodies.

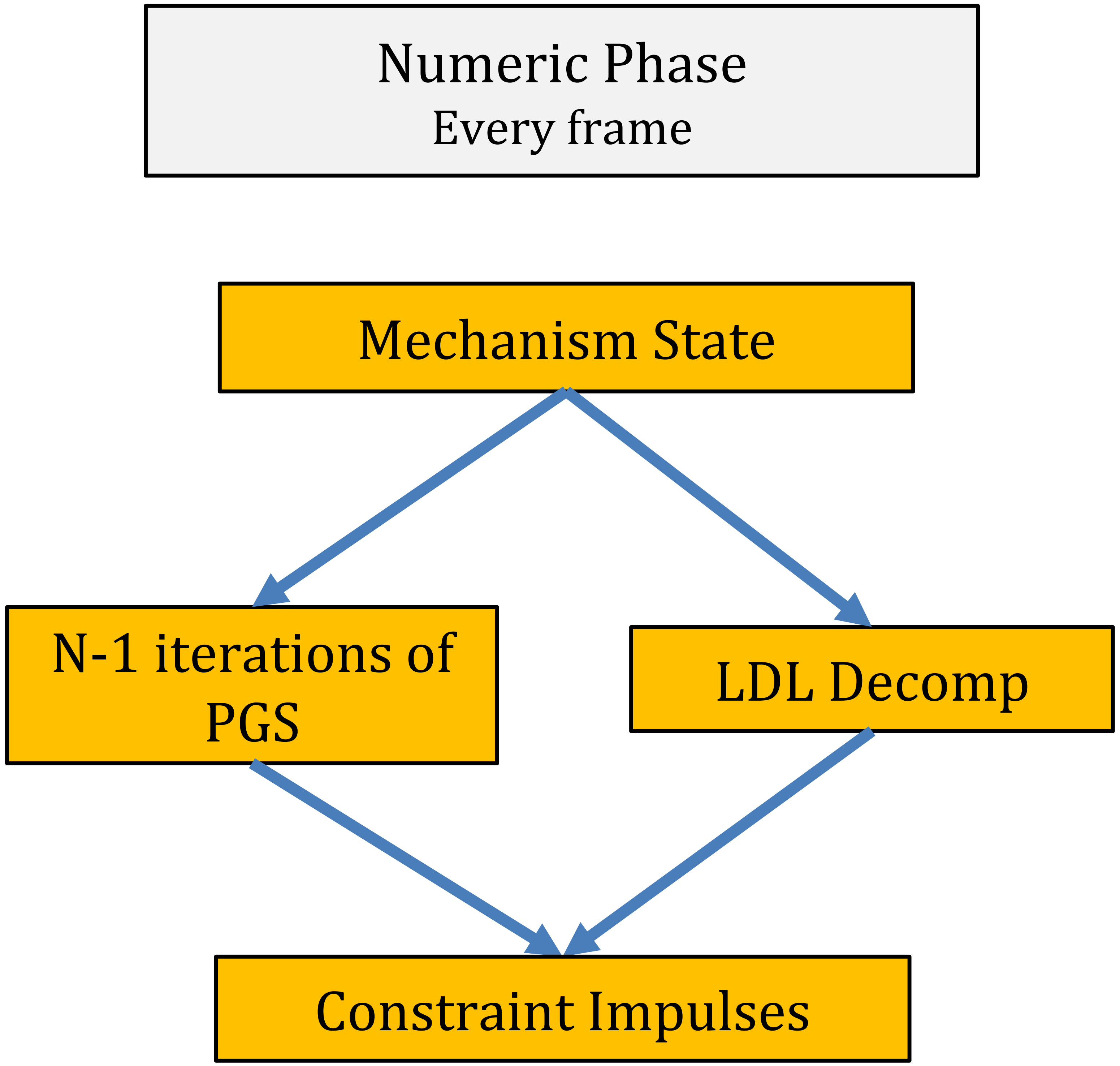
Support for Complex Mechanisms



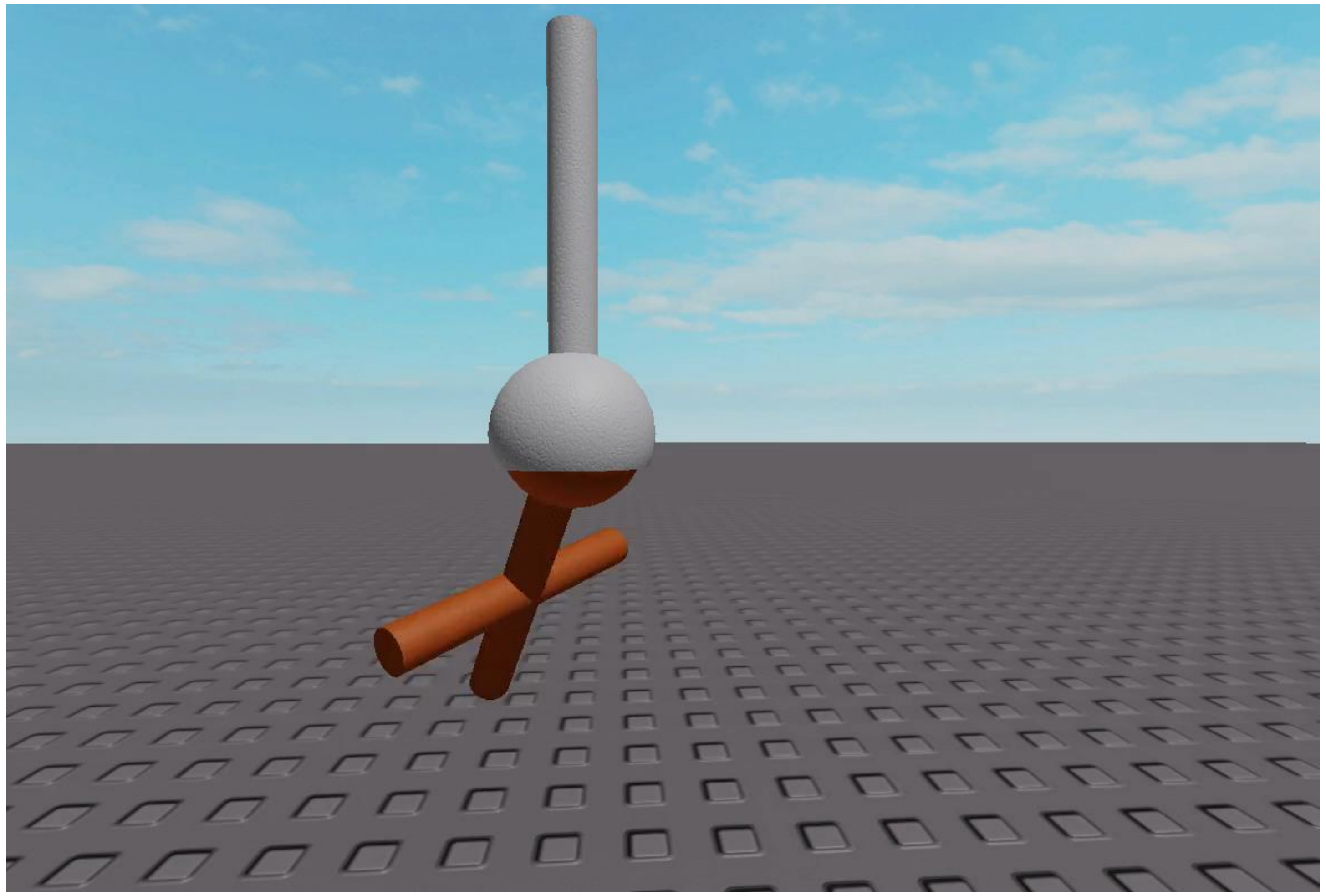
Typical Physics: PGS Solver

Roblox Physics: LDL-PGS Solver

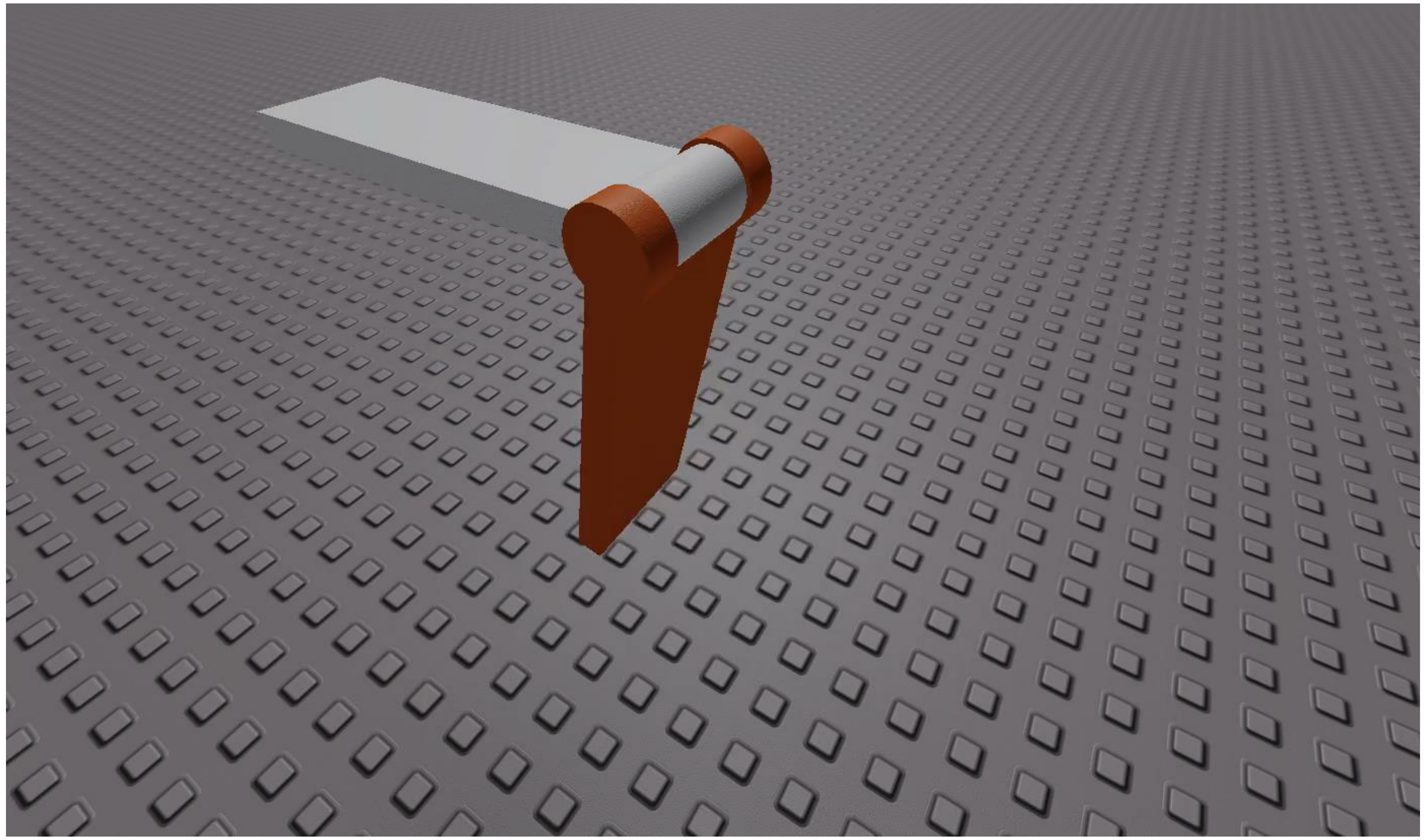

Symbolic Phase
Once per mechanism
Mechanism Structure
LDL Decomposition Program


Numeric Phase
Every frame
Mechanism State
N-1 iterations of PGS
LDL Decomp
Constraint Impulses

Constraint Examples



Ball In Socket



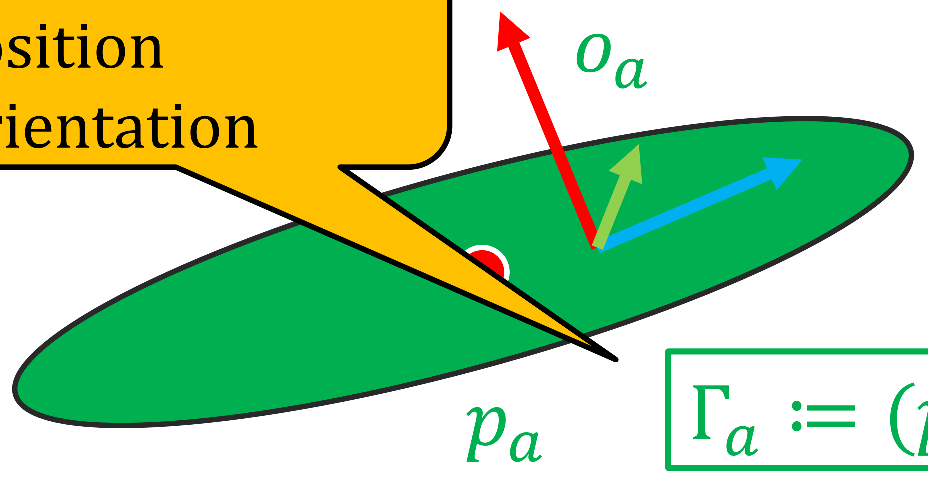
Hinge

Constraint Examples

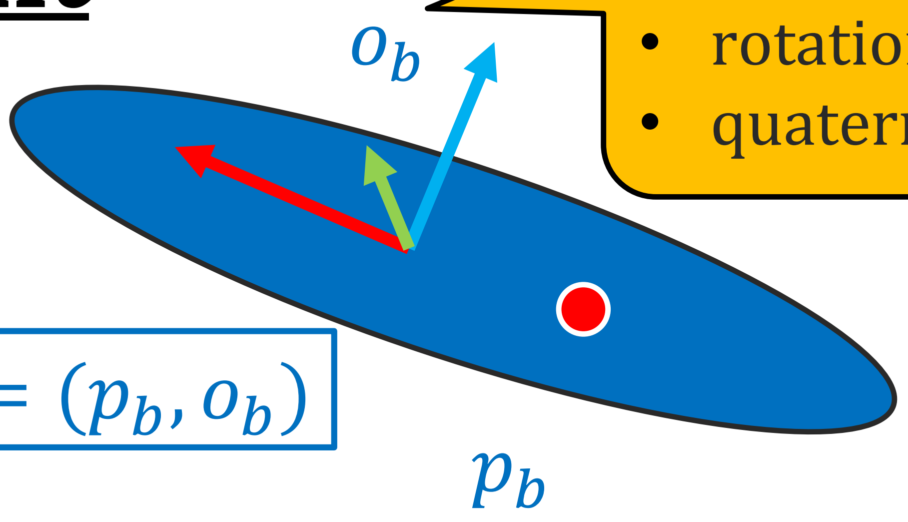
- Ball in Socket
- Hinge
- Rod (Distance Constraint)
- Prismatic
- Cylindrical
- Angular limit
- Positional limit
- Rope
- ...

Constraint

Coordinates:
1. Position
2. Orientation



$\Gamma_a := (p_a, o_a)$



$\Gamma_b := (p_b, o_b)$

…

o_a, o_b:
- rotation matrices
- quaternions

Differentiable function of coordinates:

$$c(\Gamma_a, \Gamma_b, \ldots) \in \mathbb{R}^d$$

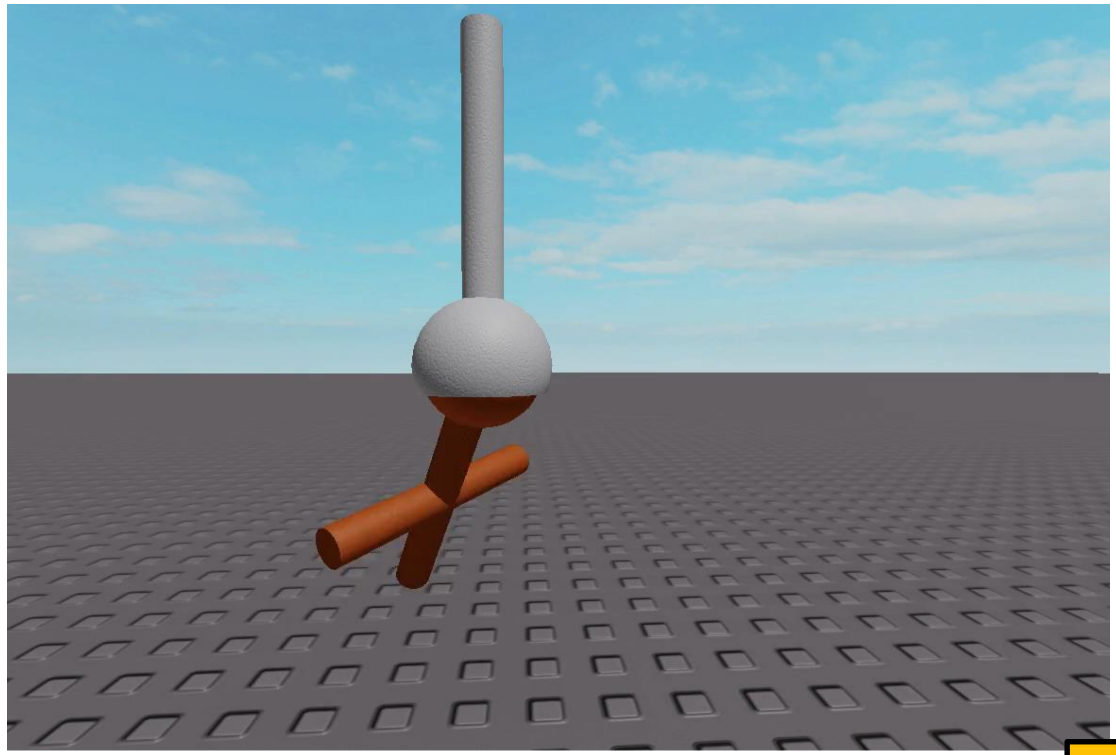
$\mathbb{R}^d$ = Constraint Space
d = Degree

Bodies respect the constraint if:

$$c(\Gamma_a, \Gamma_b, \ldots) = 0$$

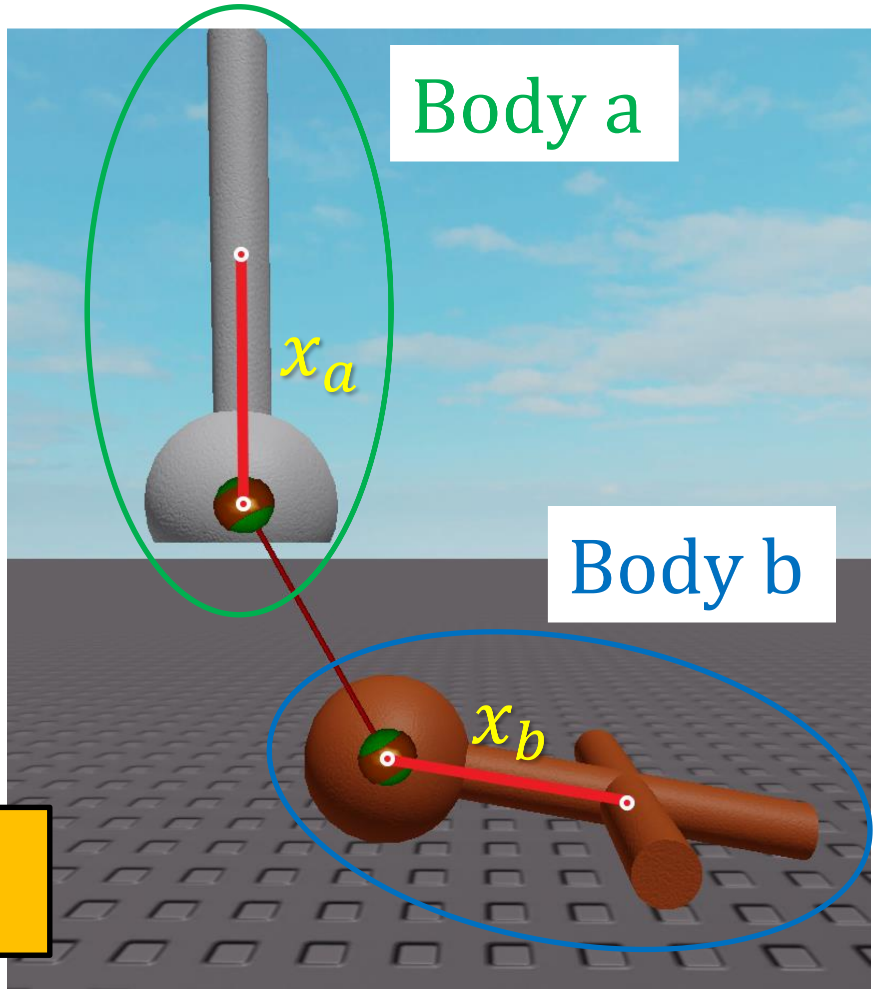
Must be **regular** on the zero locus:

Jacobian has full rank ⇔ d = number of DOF removed



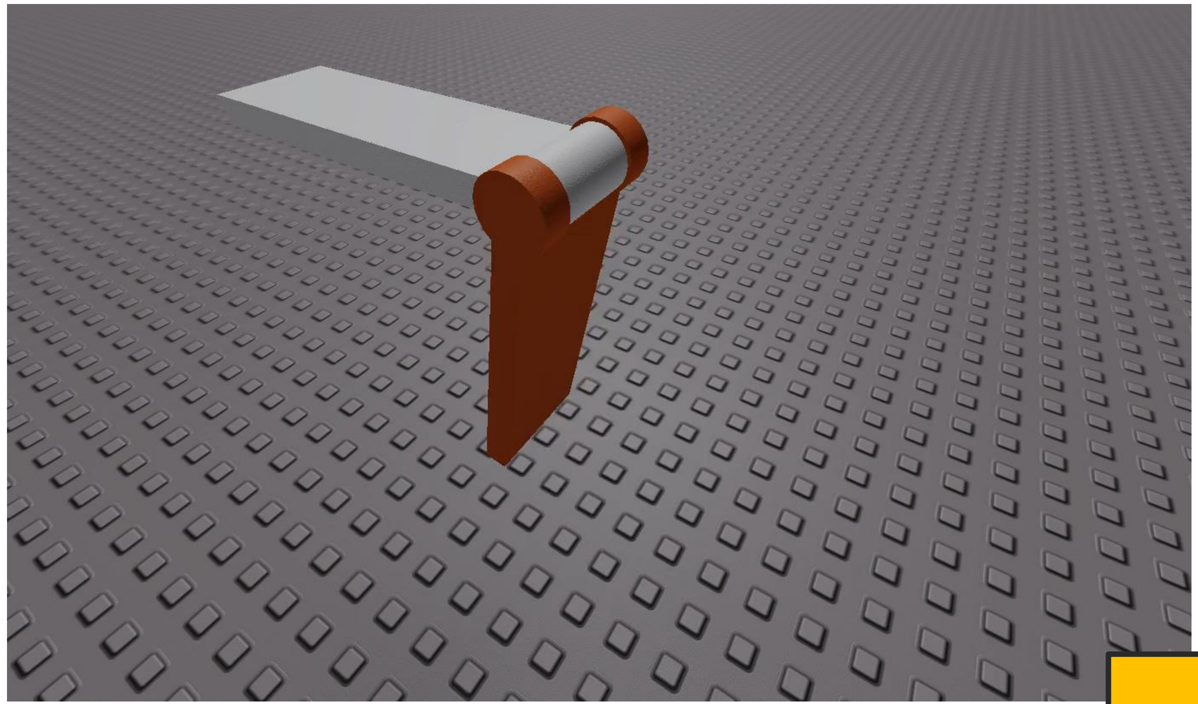
Ball in Socket

x_a - pivot in local space of a
x_b - pivot in local space of b

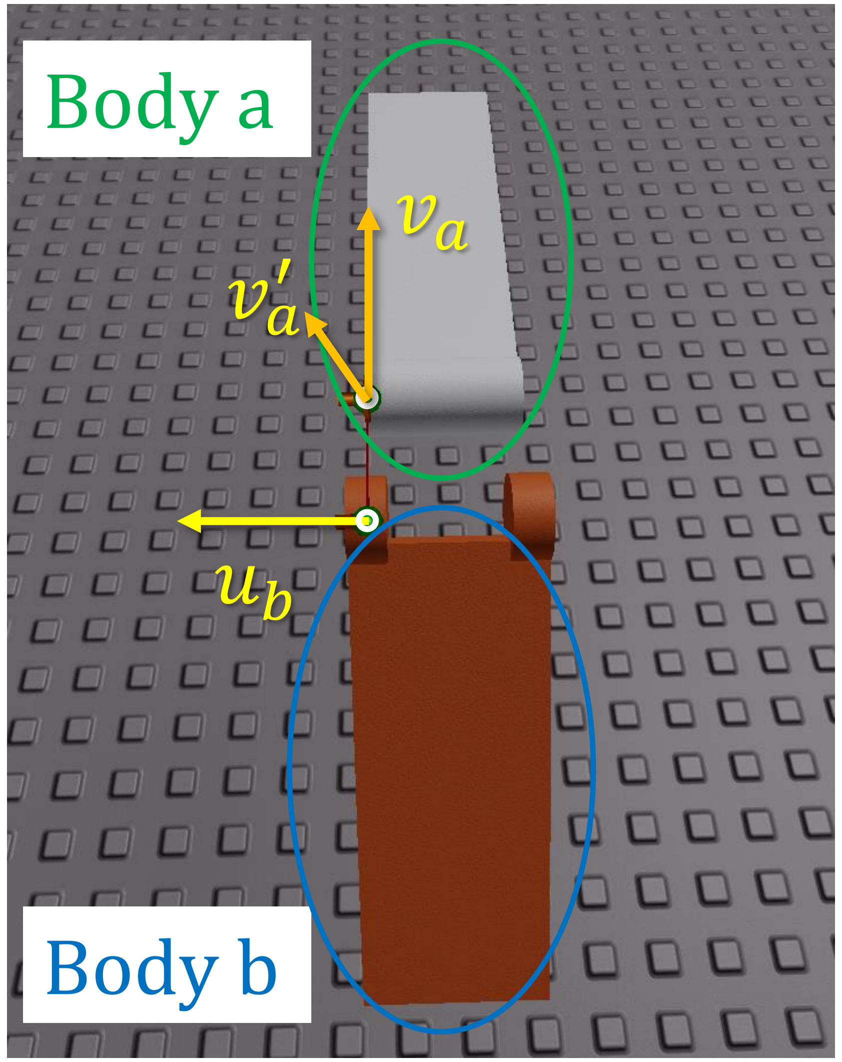


Constraint Function:

$$c(\Gamma_a, \Gamma_b) = \underbrace{\overbrace{(p_a + o_a x_a)}^{x_a \text{ in world}} - \overbrace{(p_b + o_b x_b)}^{x_b \text{ in world}}}_{\text{Same Location in World Space}} \in \mathbb{R}^3$$



Hinge



Ball in Socket

$u_b \perp v_a, v'_a$
in world space

Constraint Function:

$$c(\Gamma_a, \Gamma_b) = \begin{bmatrix} (p_a + o_a x_a) - (p_b + o_b x_b) \\ (o_a v_a) \cdot (o_b u_b) \\ (o_a v'_a) \cdot (o_b u_b) \end{bmatrix} \in \mathbb{R}^5$$

Velocity Space Constraint: Jacobian

Take the derivative of the constraint:

$$\frac{d}{dt}c(\Gamma_a, \Gamma_b, \dots) = \mathcal{J}_c \cdot \begin{bmatrix} \mathcal{V}_a \\ \mathcal{V}_b \\ \vdots \end{bmatrix}$$

Jacobian: $d \times 6n$ matrix

$$\mathcal{V}_\beta = \begin{bmatrix} v_\beta \\ \omega_\beta \end{bmatrix}$$

Linear Velocity

Angular Velocity

Jacobian:

Body Space Velocities

Jacobian

Constraint Space Velocities

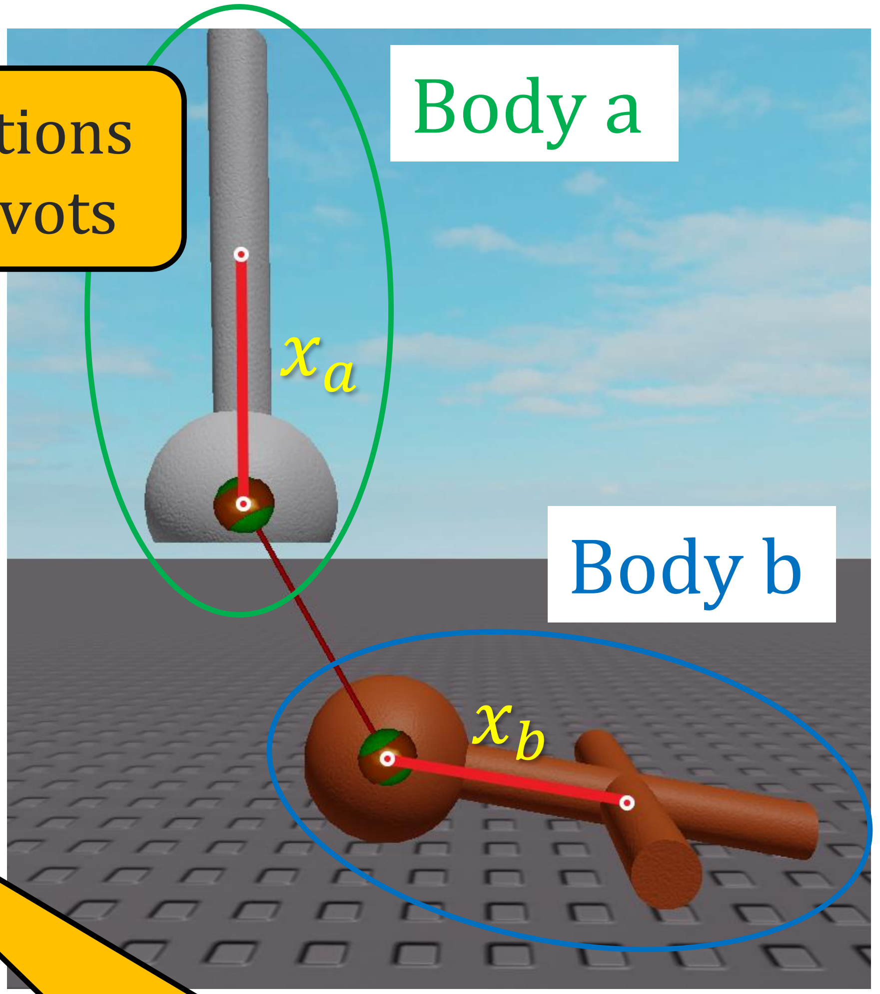
$$\mathcal{C}(\Gamma_a, \Gamma_b, \ldots) = 0 \quad \Rightarrow \quad \mathcal{J}_{\mathcal{C}} \begin{bmatrix} \mathcal{V}_a \\ \mathcal{V}_b \\ \vdots \end{bmatrix} = 0$$

No Motion In Constraint Space

Jacobian: Ball in Socket

$$c(\Gamma_a, \Gamma_b) = (p_a + o_a x_a) - (p_b + o_b x_b)$$

Relative positions of the two pivots



$$\frac{d}{dt}c = (v_a - o_a x_a \times \omega_a) - (v_b - o_b x_b \times \omega_b)$$

$$= \underbrace{[\underbrace{Id \quad -(o_a x_a)^\times}_{\mathcal{J}_c^{(a)}} \quad \underbrace{-Id \quad (o_b x_b)^\times}_{\mathcal{J}_c^{(b)}}]}_{\mathcal{J}_c} \begin{bmatrix} v_a \\ \omega_a \\ v_b \\ \omega_b \end{bmatrix}$$

Relative velocity of the two pivots

$v^\times$: cross-product matrix

Mechanism

It's a collection of Constraints and Bodies:

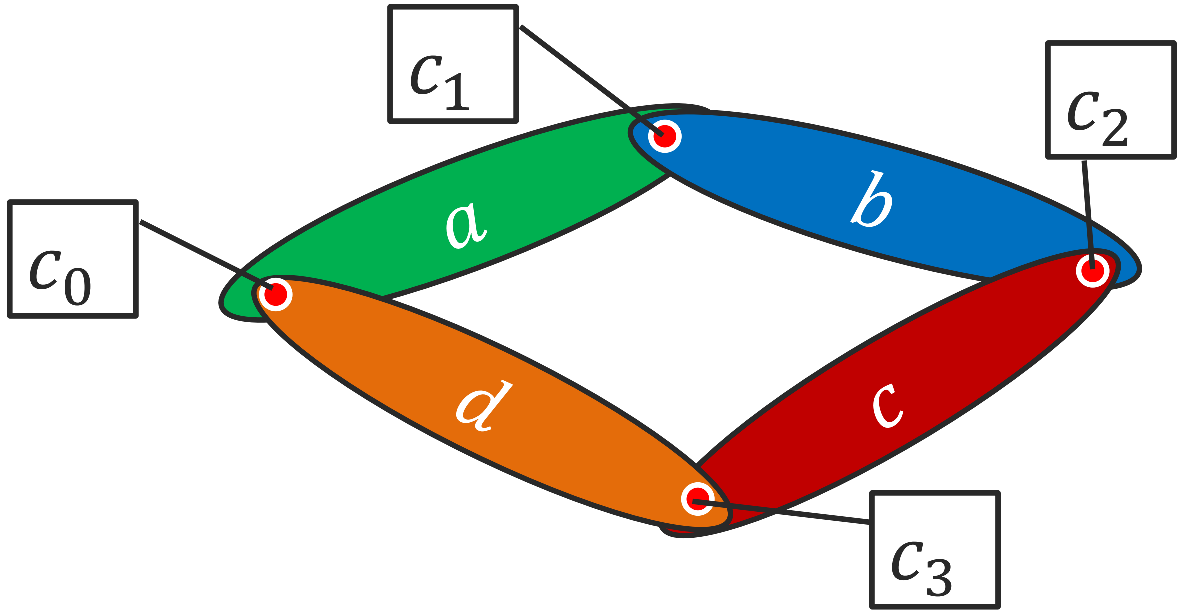
Global Constraint

$$\mathcal{C} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \end{bmatrix}, \qquad \mathcal{B} = [a, b, \dots]$$

Each constraint:
2 or 3 bodies

$$\frac{d}{dt}\mathcal{C}(\Gamma_a, \Gamma_b, \dots) = \begin{bmatrix} \mathcal{J}_0 \\ \mathcal{J}_1 \\ \vdots \end{bmatrix} \begin{bmatrix} \mathcal{V}_a \\ \mathcal{V}_b \\ \vdots \end{bmatrix} = \mathcal{J}_{\mathcal{C}} \mathcal{V}_{\mathcal{B}}$$

Each row: 12 or 18
non-zero entries



$$
\mathcal{J}_{\mathcal{C}} = \begin{array}{cc}
\begin{bmatrix}
\mathcal{J}_0^{(a)} & 0 & 0 & \mathcal{J}_0^{(d)} \\
\mathcal{J}_1^{(a)} & \mathcal{J}_1^{(b)} & 0 & 0 \\
0 & \mathcal{J}_2^{(b)} & \mathcal{J}_2^{(c)} & 0 \\
0 & 0 & \mathcal{J}_3^{(c)} & \mathcal{J}_3^{(d)}
\end{bmatrix} & \begin{array}{l} \} \, c_0 \\ \} \, c_1 \\ \} \, c_2 \\ \} \, c_3 \end{array} \\
\begin{array}{cccc} \underbrace{\quad}_{a} & \underbrace{\quad}_{b} & \underbrace{\quad}_{c} & \underbrace{\quad}_{d} \end{array} &
\end{array}
$$

Euler Integration Step

$$\mathcal{V}_0 := \text{Velocities at } t = 0$$

$$\mathcal{W} = \begin{bmatrix} m_0^{-1} Id & & & & \\ & I_0^{-1} & & & \\ & & \ddots & & \\ & & & m_{n-1}^{-1} Id & \\ & & & & I_{n-1}^{-1} \end{bmatrix}$$

Velocities next time step:

$$\mathcal{V}_1 = \mathcal{V}_0$$
$$+\mathcal{W} \cdot \text{External Forces} \cdot dt$$
$$+\mathcal{W}\mathcal{J}^t\Lambda$$

D'Alembert's Principle

Lagrangian Multipliers
aka Impulses

Solver: finds Λ such that:

$$\mathcal{J}\mathcal{V}_1 = 0$$

Linearized Constraint Equation

Valid only for equality constraints

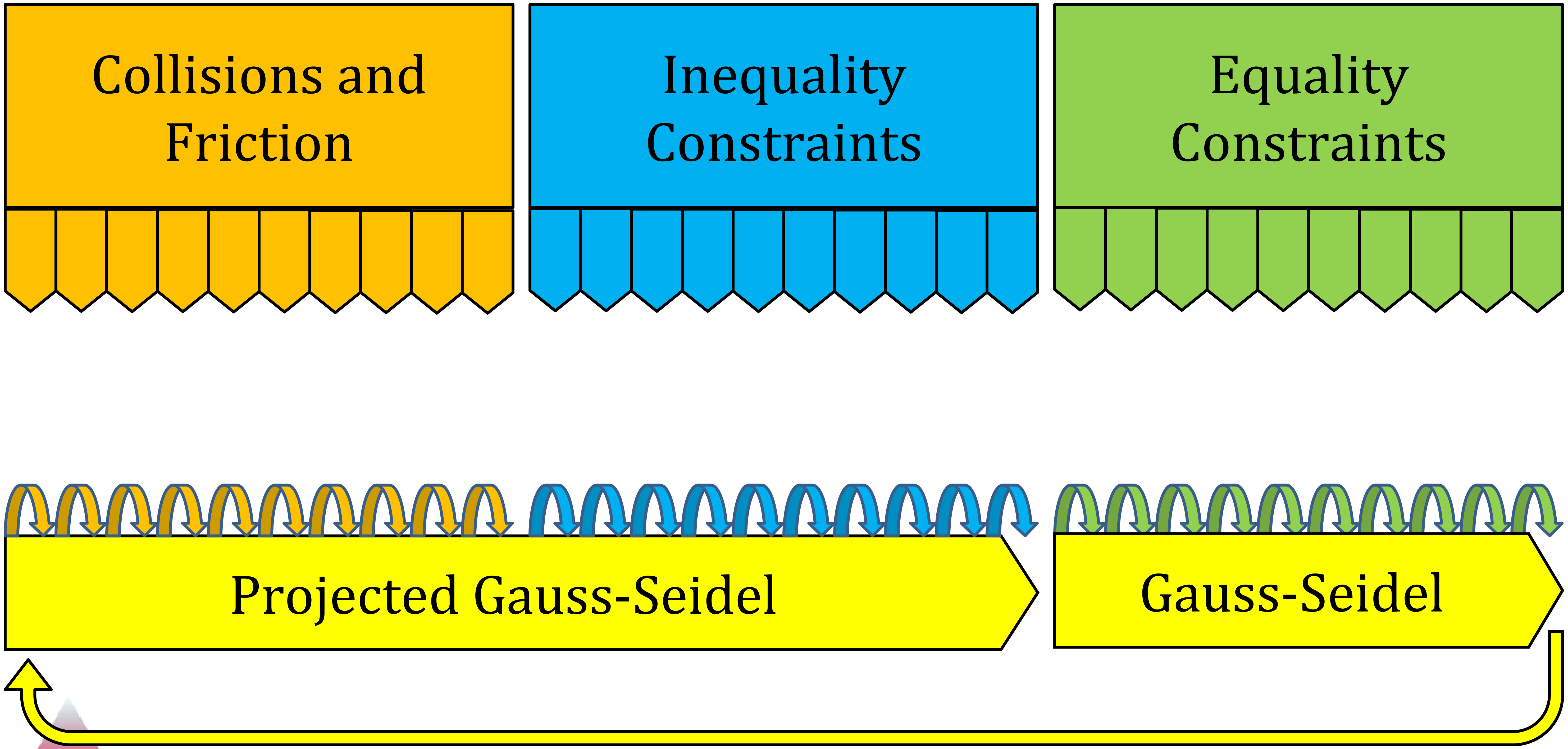
$$\mathcal{K}\Lambda = \mathcal{R}$$

$$\mathcal{K} := \mathcal{J}\mathcal{W}\mathcal{J}^t$$

$$\mathcal{R} := -\mathcal{J}(\ldots)$$

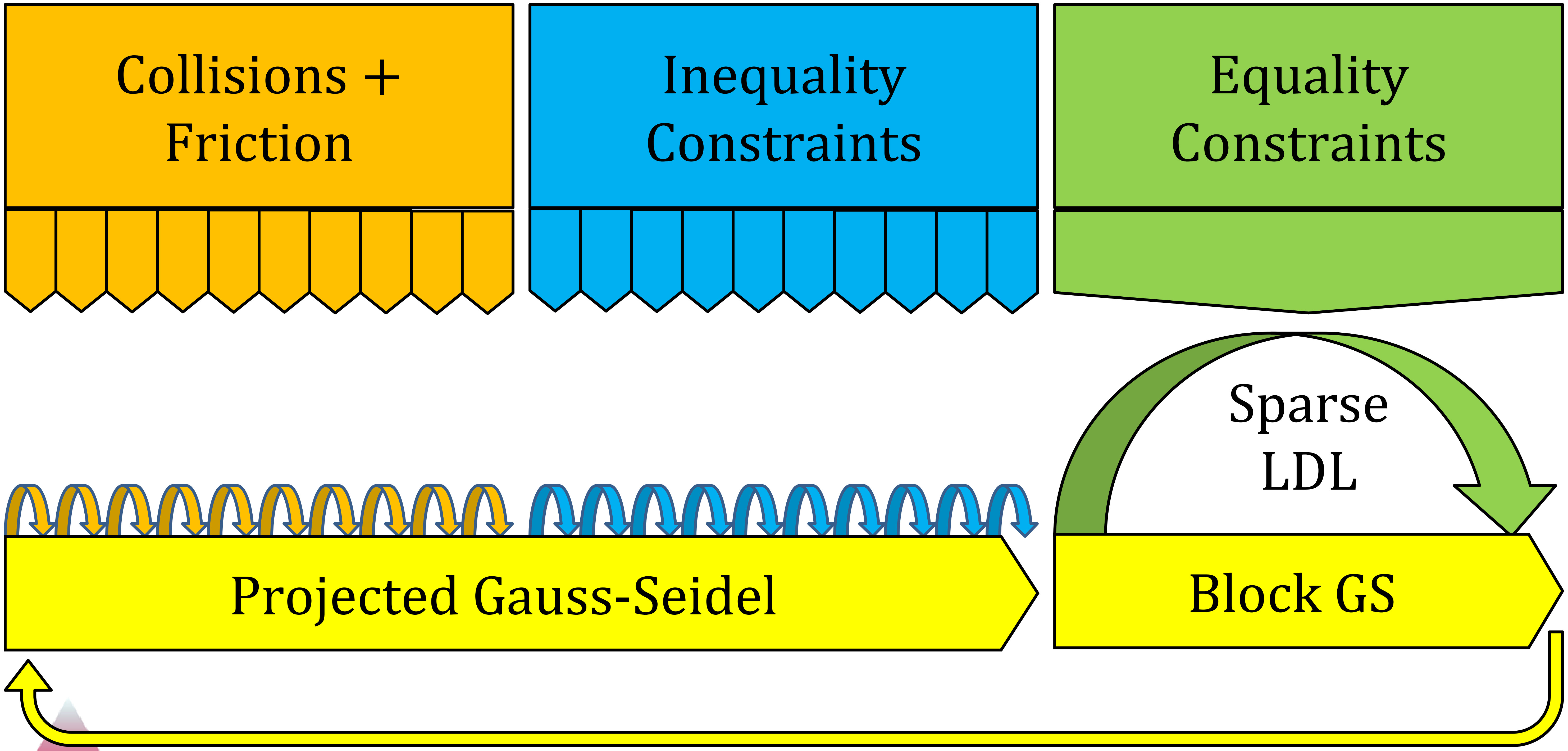
- Symmetric
- Positive semidefinite

PGS Solver

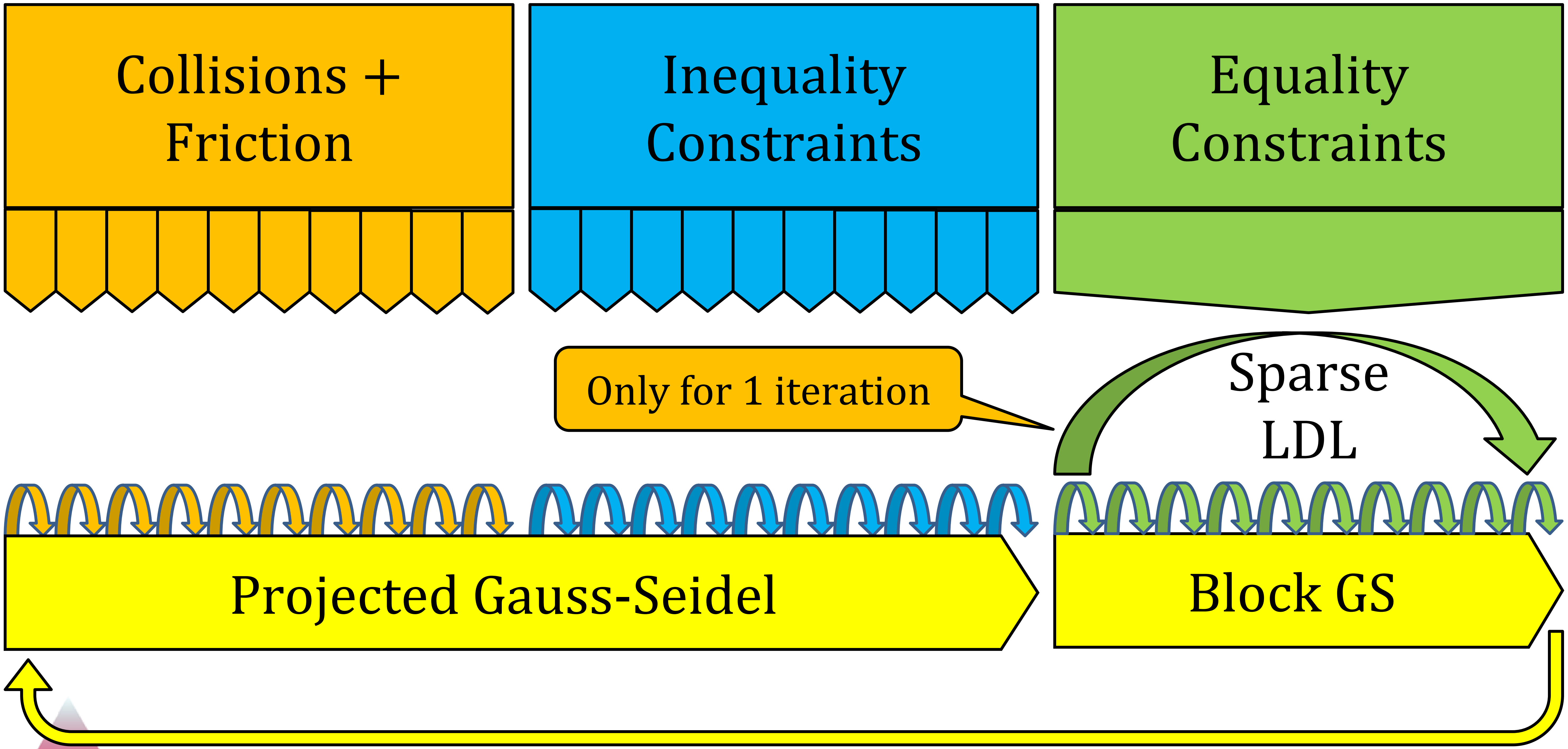


Collisions and Friction
Inequality Constraints
Equality Constraints
Projected Gauss-Seidel
Gauss-Seidel

LDL-PGS Solver



LDL-PGS Solver



Solving $\mathcal{K}\Lambda = \mathcal{R}$: (Projected) Gauss-Seidel

$$\Lambda = \text{Estimate}$$

Residual

$$\delta\mathcal{R}_i = \mathcal{R}_i - \mathcal{K}_i\Lambda$$

Bottleneck: many non-zero terms!

$i = \text{rows}$

$$\Lambda_\mathrm{i} \leftarrow \Lambda_\mathrm{i} + \mathcal{D}_i^{-1}\delta\mathcal{R}_i$$

Diagonal element

$$(\text{Project } \Lambda_\mathrm{i})$$

Faster Calculation of $\mathcal{K}_i\Lambda$

Constraint Equation:

$$\mathcal{K}\Lambda = \mathcal{R}$$

$$(\mathcal{J}\mathcal{W}\mathcal{J}^t)\Lambda = \mathcal{R}$$

Associativity of matrix multiplication

$$\mathcal{J}\underbrace{(\mathcal{W}\mathcal{J}^t\Lambda)}_{\delta\mathcal{V}} = \mathcal{R}$$

Velocity Changes

In Gauss-Seidel replace:

$$\mathcal{K}_i\Lambda \leftrightarrow \mathcal{J}_i\delta\mathcal{V}$$

Only 12 (or 18) Terms!

Impulse Solver (Kaczmarz Method)

$$\Lambda = \text{Estimate}, \delta\mathcal{V} = \mathcal{W}\mathcal{J}^t\Lambda$$

$$\Lambda_i \leftarrow \Lambda_i + \mathcal{D}_i^{-1}(\mathcal{R}_i - \mathcal{J}_i\delta\mathcal{V})$$

$$(\Lambda_i = \text{Project}(\Lambda_i))$$

$$\delta\mathcal{V}_{a_i} += \mathcal{W}_a[\mathcal{J}_i^a]^t\delta\Lambda_i$$

$$\delta\mathcal{V}_{b_i} += \mathcal{W}_b[\mathcal{J}_i^b]^t\delta\Lambda_i$$

$i = \text{rows}$

Only 50 Float Operations

Update $\delta\mathcal{V}$ using solution changes $\delta\Lambda_i$

Block Structure

Row partition:

$$\pi = \{\pi_0, \pi_1, \dots\}$$
$$\pi_i = \{\pi_{i,0}, \pi_{i,1}, \dots\}$$

List of rows for each i

$$\mathcal{J}_\pi = \begin{bmatrix} \mathcal{J}_{\pi_0} \\ \mathcal{J}_{\pi_1} \\ \vdots \end{bmatrix}, \qquad \mathcal{J}_{\pi_i} = \begin{bmatrix} \mathcal{J}_{\pi_{i,0}} \\ \mathcal{J}_{\pi_{i,1}} \\ \vdots \end{bmatrix}$$

Partitioned Constraint Matrix

Symmetric Matrices

Transposed

$$\mathcal{K} = \begin{bmatrix} \mathcal{J}_{\pi_0} \mathcal{W} \mathcal{J}_{\pi_0}^t & \mathcal{J}_{\pi_0} \mathcal{W} \mathcal{J}_{\pi_1}^t & \cdots \\ \mathcal{J}_{\pi_1} \mathcal{W} \mathcal{J}_{\pi_0}^t & \mathcal{J}_{\pi_1} \mathcal{W} \mathcal{J}_{\pi_1}^t & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$\mathcal{N}$ for "Node" Matrix

$$= \begin{bmatrix} \mathcal{N}_0 & \mathcal{E}_{01} & \cdots \\ \mathcal{E}_{10} & \mathcal{N}_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$\mathcal{E}$ for "Edge" Matrix

Block Impulse Solver

$$\Lambda = \text{Estimate}, \delta\mathcal{V} = \mathcal{W}\mathcal{J}^t\Lambda$$

$$\Lambda_{\pi_i} \leftarrow \Lambda_{\pi_i} + \mathcal{N}_i^{-1}(\mathcal{R}_{\pi_i} - \mathcal{J}_{\pi_i}\delta\mathcal{V})$$

$$\delta\mathcal{V} \mathrel{+}= \mathcal{W}\mathcal{J}_{\pi_i}^t\delta\Lambda_{\pi_i}$$

$i =$ elements of partition

Inverse of a square matrix

Block solver doesn't support projection.

We need to solve:

$$\delta\Lambda_{\pi_i} = \mathcal{N}_i^{-1}\delta r$$

Small dimensions => easily invert $\mathcal{N}_i$

Ex: natural partition based on constraints:

- a ball-in-socket (dim = 3x3)
- a hinge (dim = 5x5)
- a cylindrical (dim = 4x4)
- …

$\mathcal{N}_i$ are invertible because constraints are **regular**

Idea: group all equality constraints together

π_0 : rows of all equality constraints

$\pi_1, \pi_2, \ldots$: other individual rows

H for holonomic

$$\mathcal{H} := \mathcal{N}_0 = J_{\pi_0} \mathcal{W} J_{\pi_0}^t$$

constraint matrix of equality constraints.

How to evaluate?

$$\mathcal{H}^{-1}(\ldots)$$

Potential issues with $\mathcal{H}$:

1. Not invertible
2. Large dimensions
3. Large dense submatrices

Solutions:

1. Regularize: add compliance
2. Use a sparse method
3. Increase sparsity: split bodies

Compliance and Regularization

$\mathcal{H}$ is **symmetric positive semidefinite.**

It may be not be invertible, but

$$\widetilde{\mathcal{H}} := \mathcal{H} + \begin{bmatrix} \epsilon_0 & & \\ & \epsilon_1 & \\ & & \ddots \end{bmatrix}$$

ϵ_i are not unit free, scale dependent

Is invertible for any $\epsilon_0, \epsilon_1, \ldots > 0$

Use a scaled diagonal:

$$\epsilon_i = \epsilon \cdot \mathcal{H}_{i,i} \text{ for small } \epsilon > 0$$

Adds **compliance** to constraints!

Compliance and Regularization

Traditional PGS engines also use compliance to stabilize solutions

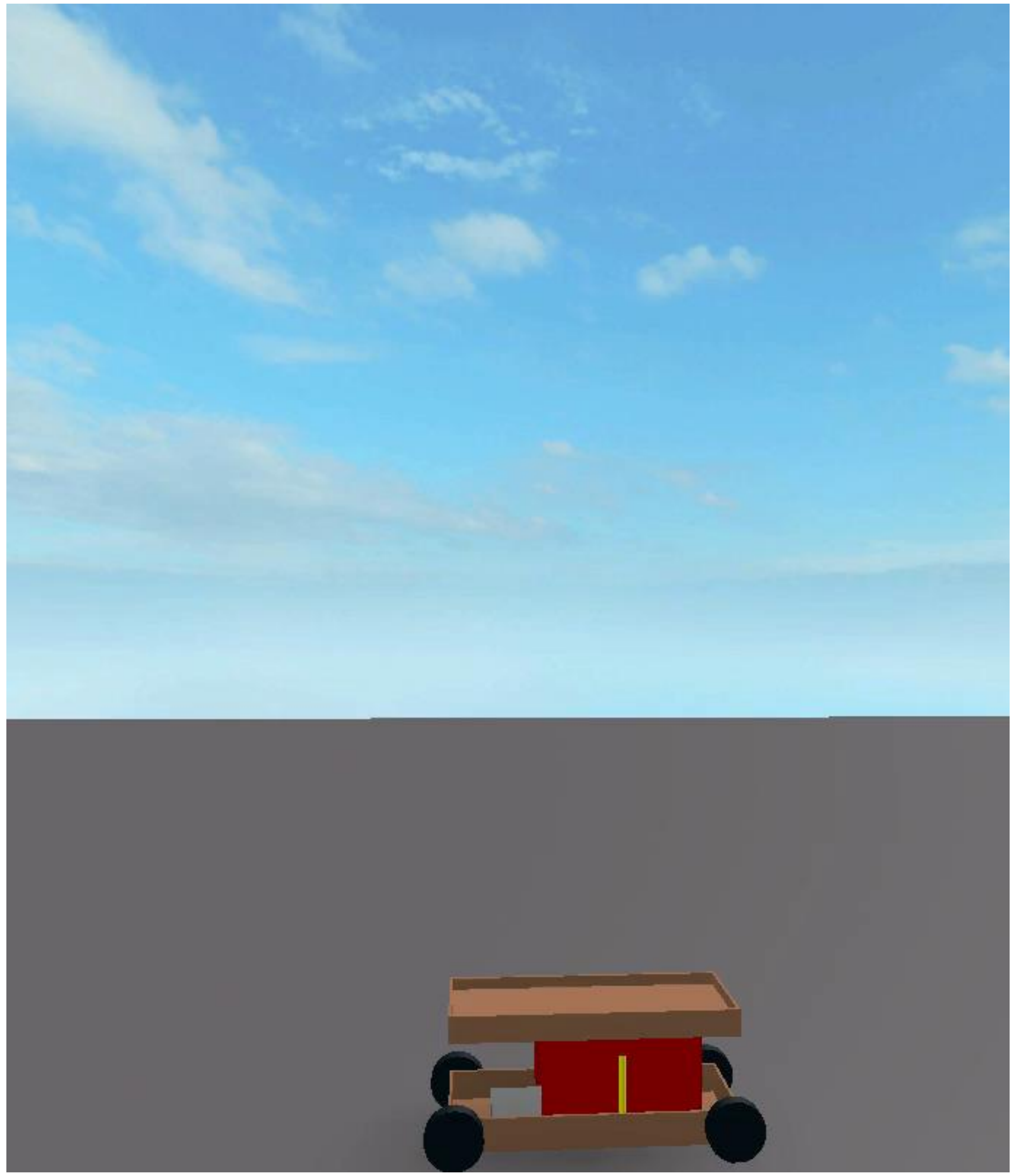
$$\text{PGS for } \tilde{\mathcal{H}}\Lambda = \mathcal{R}$$

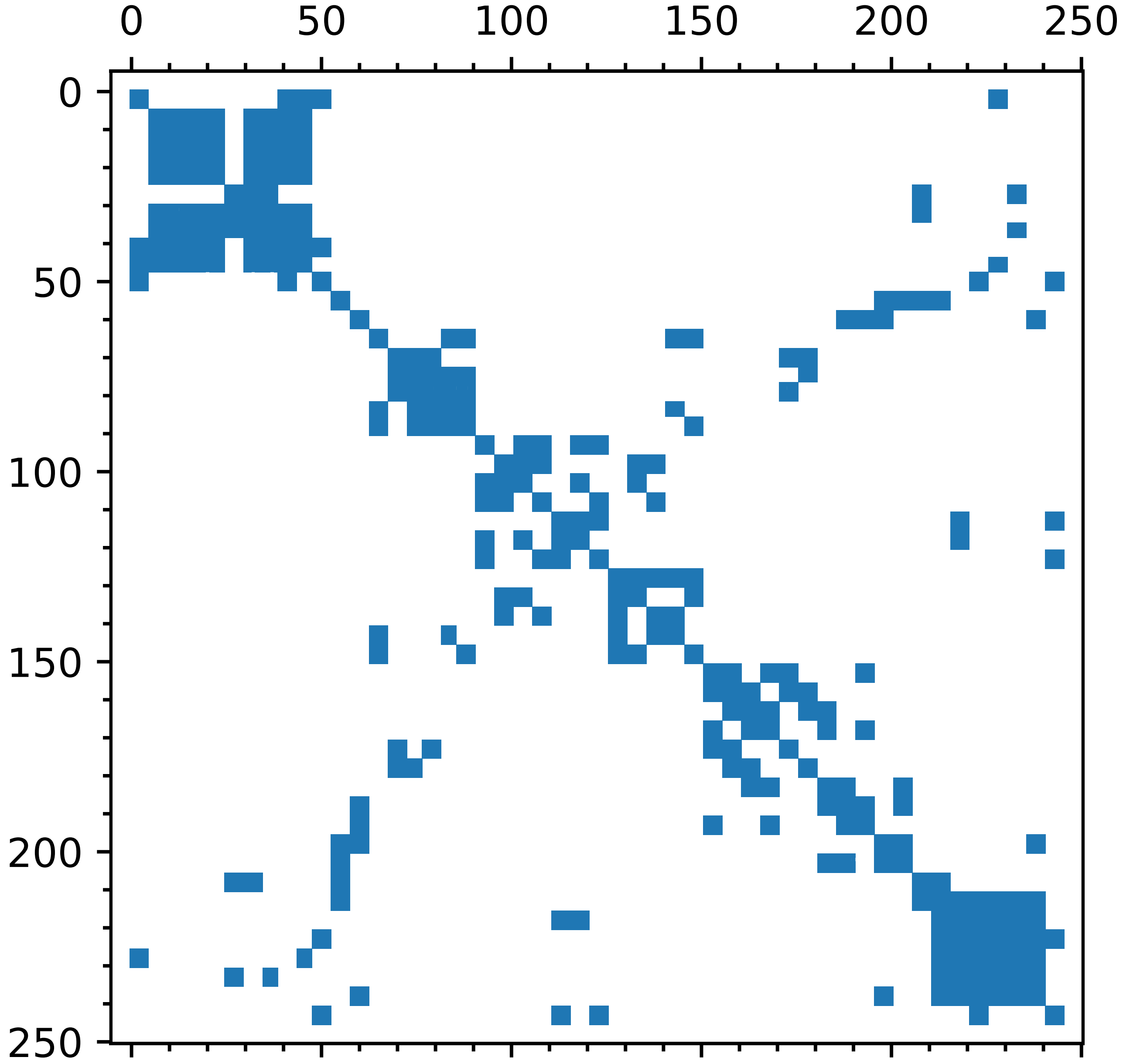
$$\Updownarrow$$

$$\text{PGS for } \mathcal{H}\Lambda = \mathcal{R}$$
$$\text{with } \Lambda_i \leftarrow (1+\epsilon_i)^{-1}\Lambda_i$$
$$\text{for each row}$$

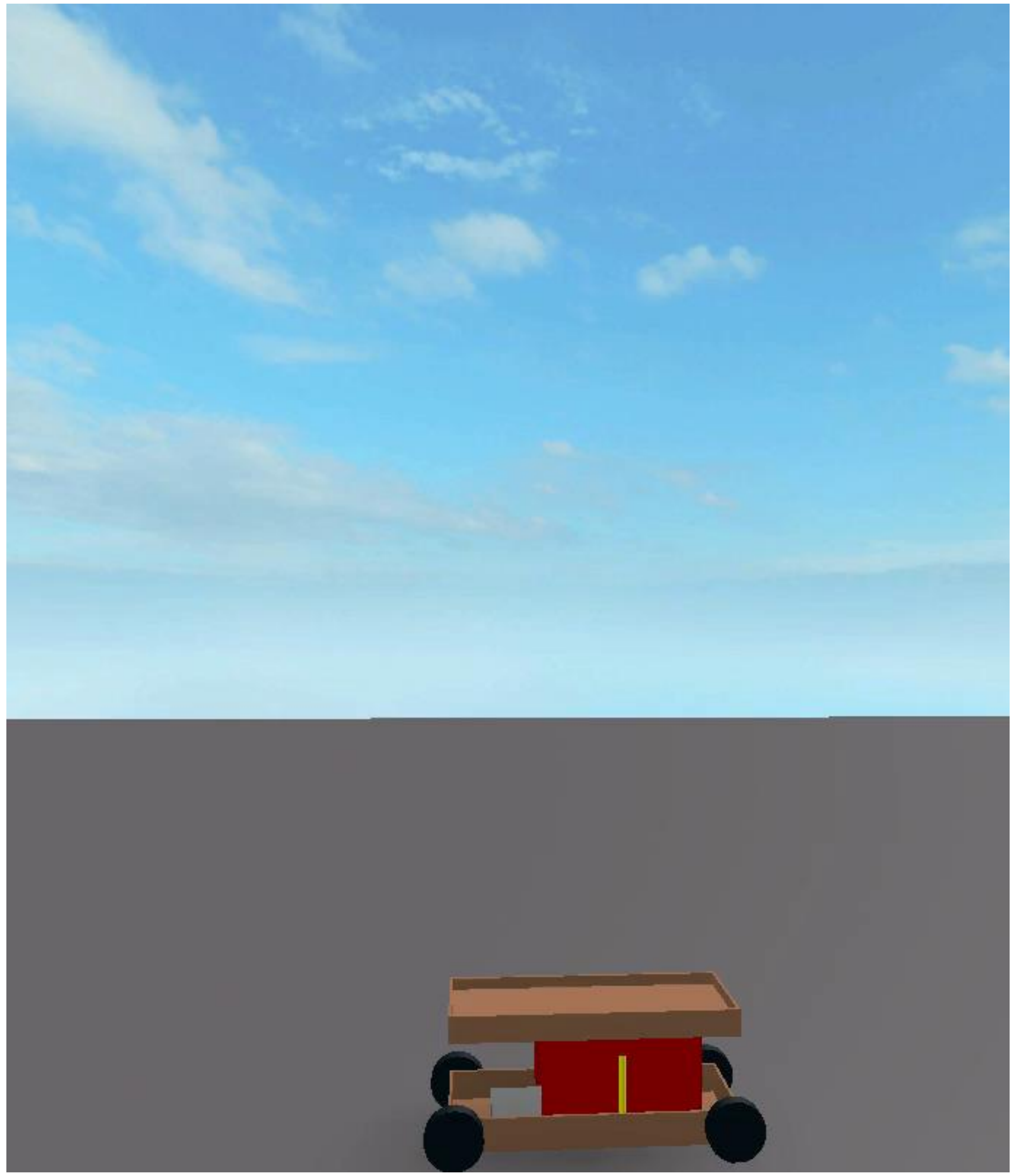
CFM in Bullet/ODE (Constraint Force Mixing)

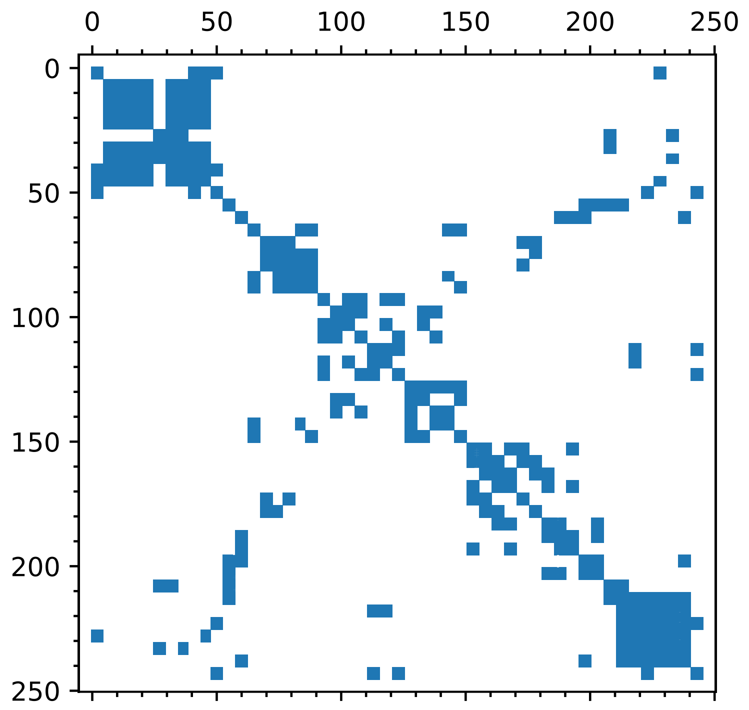
Sparse Methods





Sparse Methods





$$\#Constraints = 50$$
$$Dim(\mathcal{H}) = 246$$
$$Density(\mathcal{H}) = 12\%$$

Sparse Cholesky LDL:

$$Flops = 90k$$
$$Performance\ (Core\ i9) = 70\mu s$$

Cholesky LDL Decomposition

Any Symmetric Positive Definite matrix can be decomposed as

$$\mathcal{A} = \mathcal{L}\mathcal{D}\mathcal{L}^t$$

Where

$\mathcal{L}$ – lower triangular with 1s on diagonal

$\mathcal{D}$ – diagonal

This is useful because both $\mathcal{L}^{-1}$ and $\mathcal{D}^{-1}$ can be efficiently evaluated.

Useful to compute $\mathcal{A}^{-1}v$:

$$\begin{aligned}\mathcal{A}^{-1}v &= (\mathcal{L}^{-t}\mathcal{D}^{-1}\mathcal{L}^{-1})v \\ &= \mathcal{L}^{-t}(\mathcal{D}^{-1}(\mathcal{L}^{-1}v))\end{aligned}$$

Where

$$\mathcal{L}^{-1}(\cdot), \mathcal{L}^{-t}(\cdot)$$

Computed by back-substitution and

$$\mathcal{D}^{-1}(\cdot)$$

Are scalar products.

Block LDL

Suppose $\mathcal{A}$ is SPD and has a block structure:

$$\mathcal{A} = \begin{bmatrix} \begin{bmatrix} * & * \\ * & * \end{bmatrix} & \begin{bmatrix} * & * & * \\ * & * & * \end{bmatrix} & \begin{bmatrix} * \\ * \end{bmatrix} \\ \begin{bmatrix} * & * \\ * & * \\ * & * \end{bmatrix} & \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} & \begin{bmatrix} * \\ * \\ * \end{bmatrix} \\ \begin{bmatrix} * & * \end{bmatrix} & \begin{bmatrix} * & * & * \end{bmatrix} & \begin{bmatrix} * \end{bmatrix} \end{bmatrix}$$

Block LDL

It can decomposed as

$$\mathcal{A} = \mathcal{L}\mathcal{D}\mathcal{L}^t$$

$$\mathcal{D} = \begin{bmatrix} \begin{bmatrix} * & * \\ * & * \end{bmatrix} & & \\ & \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} & \\ & & [*] \end{bmatrix}, \qquad \mathcal{L} = \begin{bmatrix} \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} & & \\ \begin{bmatrix} * & * \\ * & * \\ * & * \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} & \\ \begin{bmatrix} * & * \end{bmatrix} & \begin{bmatrix} * & * & * \end{bmatrix} & [1] \end{bmatrix}$$

$\mathcal{D}$ and $\mathcal{L}$ inherit the block structure from $\mathcal{A}$.

Block LDL vs LDL?

Performance: block operations **faster than** scalar!

LDL: Algorithms

There are at least 2 algorithms:

- Gaussian Elimination ← we'll use this one
- Doolittle Algorithm

Schur Complement

Multiply first row by $\mathcal{E}\mathcal{N}^{-1}$ on the left

Subtract from second row

Eliminated

$$\begin{bmatrix} \mathcal{N} & \mathcal{E}^t \\ \mathcal{E} & \mathcal{S} \end{bmatrix}$$

Multiply first column by $\mathcal{N}^{-1}\mathcal{E}^t$ on the right

Subtract from second column

$$\begin{bmatrix} \mathcal{N} & \mathcal{E}^t \\ 0 & \mathcal{S} - \mathcal{E}\mathcal{N}^{-1}\mathcal{E}^t \end{bmatrix}$$

Eliminated

$$\begin{bmatrix} \mathcal{N} & 0 \\ 0 & \mathcal{S} - \mathcal{E}\mathcal{N}^{-1}\mathcal{E}^t \end{bmatrix}$$

Schur Complement

$$\begin{bmatrix} \mathcal{N} & \mathcal{E}^t \\ \mathcal{E} & \mathcal{S} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} Id & \\ \mathcal{E}\mathcal{N}^{-1} & Id \end{bmatrix}}_{\text{Lower Triangular}} \underbrace{\begin{bmatrix} \mathcal{N} & 0 \\ 0 & \mathcal{S} - \mathcal{E}\mathcal{N}^{-1}\mathcal{E}^t \end{bmatrix}}_{\text{Block Diagonal}} \underbrace{\begin{bmatrix} Id & \mathcal{N}^{-1}\mathcal{E}^t \\ & Id \end{bmatrix}}_{\text{Upper Triangular}}$$

Block Gaussian Elimination = Recursive Schur complements

Block Gaussian Elimination

$$\left[\begin{array}{c|ccc} \mathcal{N}_0 & \mathcal{E}_{10}^t & \mathcal{E}_{20}^t & \cdots \\ \hline \mathcal{E}_{10} & \mathcal{N}_1 & \mathcal{E}_{21}^t & \cdots \\ \mathcal{E}_{20} & \mathcal{E}_{21} & \mathcal{N}_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}\right]$$

$\boxed{\mathcal{E}_{*,0}^t}$ $\boxed{\mathcal{S}_0}$ $\boxed{\mathcal{E}_{*,0}}$

$$= \begin{bmatrix} \mathcal{N}_0 & \mathcal{E}_{*,0}^t \\ \mathcal{E}_{*,0} & \mathcal{S}_0 \end{bmatrix}$$

$$= \begin{bmatrix} Id & \\ \mathcal{E}_{*,0}\mathcal{N}_0^{-1} & Id \end{bmatrix} \begin{bmatrix} \mathcal{N}_0 & 0 \\ 0 & \mathcal{S}_0 - \mathcal{E}_{*,0}\mathcal{N}_0^{-1}\mathcal{E}_{*,0}{}^t \end{bmatrix} \begin{bmatrix} Id & \mathcal{N}_0^{-1}\mathcal{E}_{*,0} \\ & Id \end{bmatrix}$$

Block Gaussian Elimination

Block Diagonals

Already Eliminated

$\mathcal{S}_{i-1}$

$$\begin{bmatrix} \ddots & 0 & 0 \\ 0 & \mathcal{N}_i & \mathcal{E}_{*,i}^t \\ 0 & \mathcal{E}_{*,i} & \mathcal{S}_i \end{bmatrix} = \underbrace{\begin{bmatrix} Id & & \\ & Id & \\ & \mathcal{L}_{*,i} & Id \end{bmatrix}}_{[\mathcal{L}_{*,i}]} \begin{bmatrix} \ddots & & \\ & \mathcal{N}_i & 0 \\ & 0 & \mathcal{S}_i - \mathcal{E}_{*,i}\mathcal{L}_{*,i}^t \end{bmatrix} \underbrace{\begin{bmatrix} Id & & \\ & Id & \mathcal{L}_{*,i}^t \\ & & Id \end{bmatrix}}_{[\mathcal{L}_{*,i}]^t}$$

$[\mathcal{L}_{*,i}] \leftarrow$ Elementary Matrices $\rightarrow [\mathcal{L}_{*,i}]^t$

Schur Complement:

1. Invert $\mathcal{N}_i$
2. Compute:
   $$\mathcal{L}_{*,i} = \mathcal{E}_{*,i}\mathcal{N}_i^{-1}$$
3. Reduce:
   $$\mathcal{S}_i \leftarrow \mathcal{S}_i - \mathcal{E}_{*,i}\mathcal{L}_{*,i}^t$$

Block LDL

$$\mathcal{D} = \begin{bmatrix} \mathcal{N}_0 & & \\ & \ddots & \\ & & \mathcal{N}_{n-1} \end{bmatrix}$$

$$\mathcal{L} = [\mathcal{L}_{*,0}][\mathcal{L}_{*,1}]\cdots[\mathcal{L}_{*,n-2}] = \begin{bmatrix} 1 & & & & \\ \uparrow & 1 & & & \\ & \uparrow & \ddots & & \\ \mathcal{L}_{*,0} & \mathcal{L}_{*,1} & & \ddots & \\ \downarrow & \downarrow & & \mathcal{L}_{*,n-2} & 1 \end{bmatrix}$$

$$\mathcal{H} = \mathcal{L}\mathcal{D}\mathcal{L}^t$$

Inverse Operator

$$\mathcal{H}^{-1}v = \mathcal{L}^{-t}\left(\mathcal{D}^{-1}\left(\mathcal{L}^{-1}v\right)\right)$$

$$\mathcal{L}^{-1} = \left[-\mathcal{L}_{*,n-2}\right]\cdots\left[-\mathcal{L}_{*,1}\right]\left[-\mathcal{L}_{*,0}\right]$$
$$\mathcal{L}^{-t} = \left[-\mathcal{L}_{*,0}\right]^{t}\left[-\mathcal{L}_{*,1}\right]^{t}\cdots\left[-\mathcal{L}_{*,n-2}\right]^{t}$$

Don't evaluate these products. Use as is.

$$\mathcal{D}^{-1} = \begin{bmatrix} \mathcal{N}_0^{-1} & & \\ & \ddots & \\ & & \mathcal{N}_{n-1}^{-1} \end{bmatrix}$$

Computed during elimination

Sparse Block LDL

Constraints

$$\mathcal{H} = \begin{bmatrix} \mathcal{N}_0 & \mathcal{E}_{10}^t & \mathcal{E}_{20}^t & \cdots & 0 \\ \mathcal{E}_{10} & \mathcal{N}_1 & 0 & \cdots & \mathcal{E}_{n-1,1}^t \\ \mathcal{E}_{20} & 0 & \mathcal{N}_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \mathcal{E}_{n-1,1} & 0 & \cdots & \mathcal{N}_{n-1} \end{bmatrix}$$

Where some $\mathcal{E}_{ij}$'s are 0.

Can we reduce the operation count in the Gaussian Elimination?

Yes absolutely! Here is how:

- Sparse matrices ⇔ Graphs
- Gaussian elimination ⇔ Process on graphs
- Pack the sparse matrix data in memory

Body Graph

- Nodes: Rigid Bodies
- Edges: Constraints

Constraint Graph

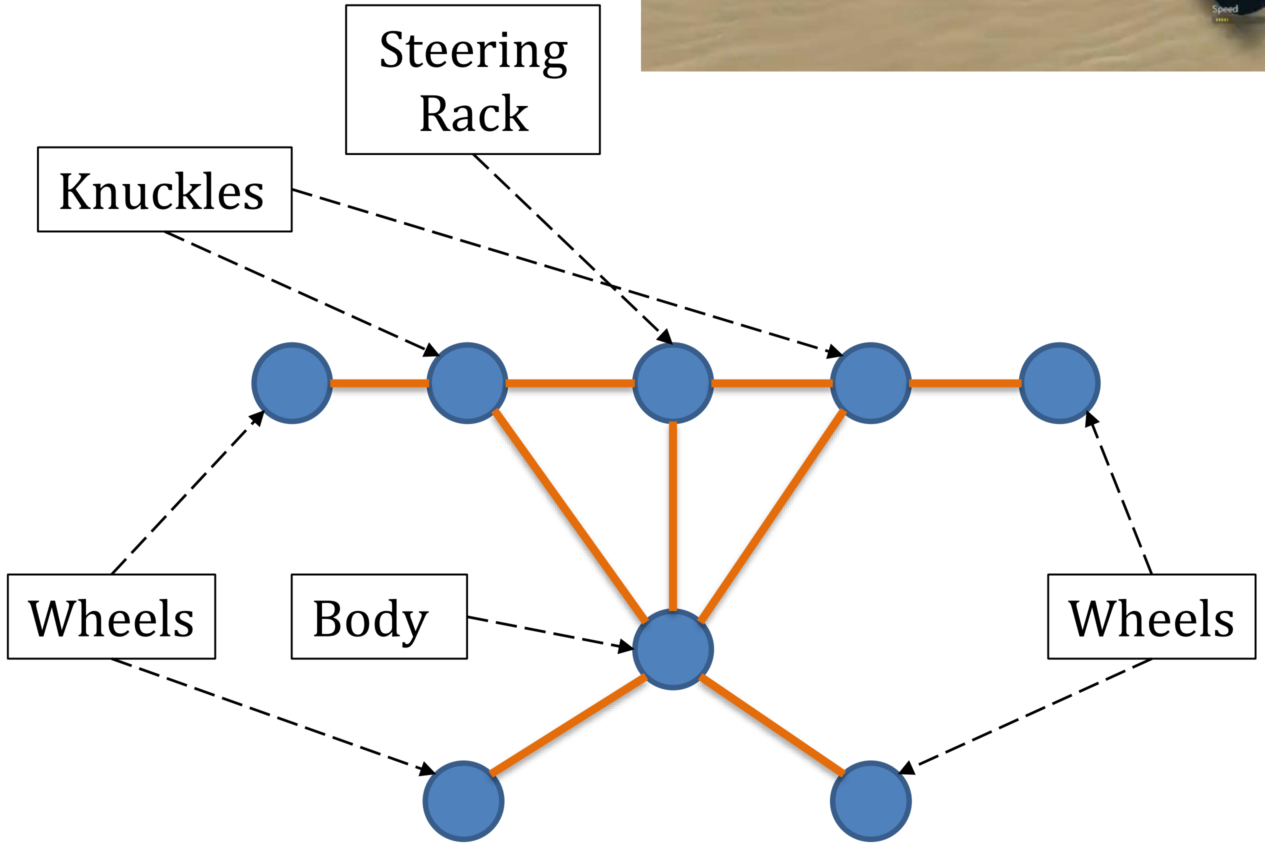
It is the **Edge Graph** of the Body Graph:

- Nodes: Constraints
- Edges: Common bodies between constraints

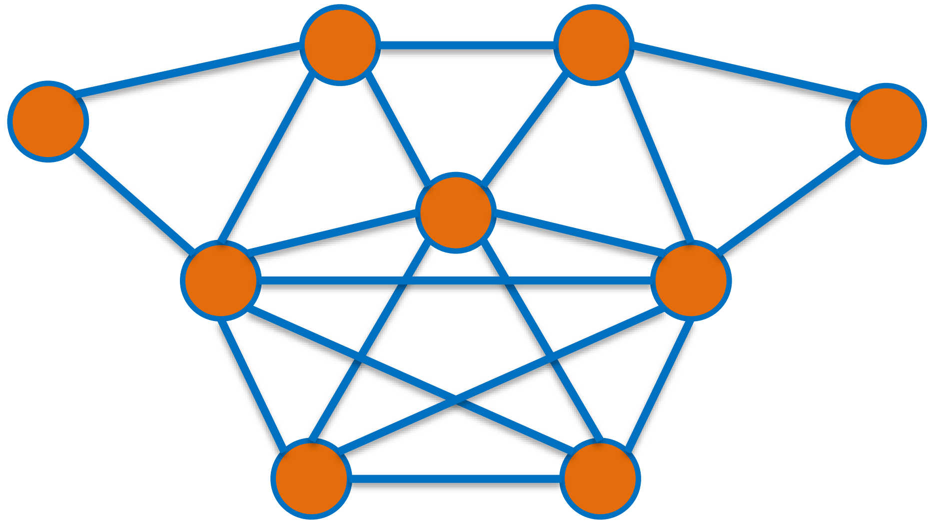
Constraint Graph = Graph of the Constraint Matrix

Constraint Graph

Constraint Matrix	Constraint Graph
Diagonal block $\mathcal{N}_i$	Node n_i
Off-diagonal block $\mathcal{E}_{ji}$	Edge e_{ji} between n_i and n_j
Gaussian Elimination	Graph Elimination

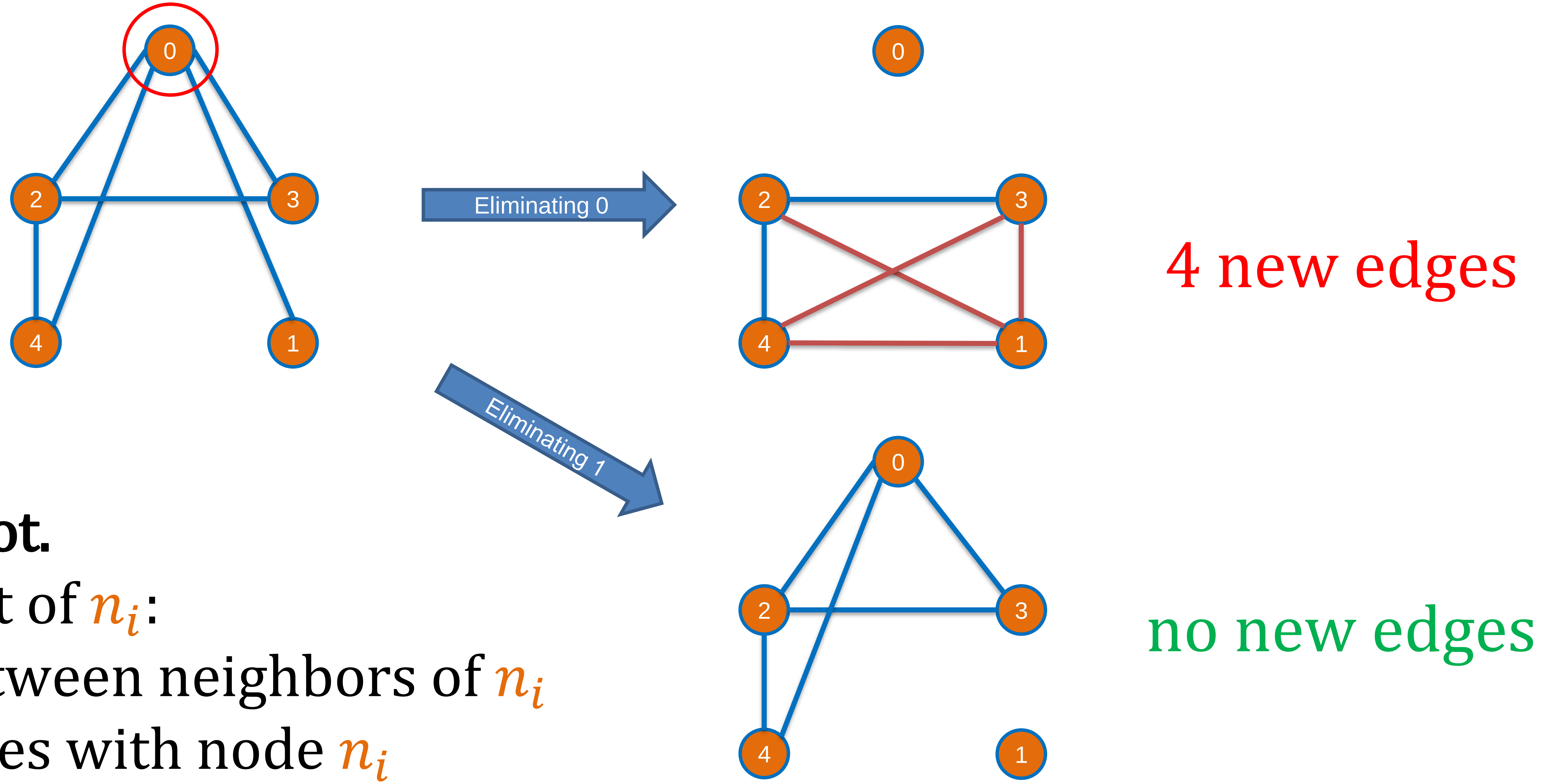


Body Graph



Constraint Graph

Gaussian Elimination on Graphs

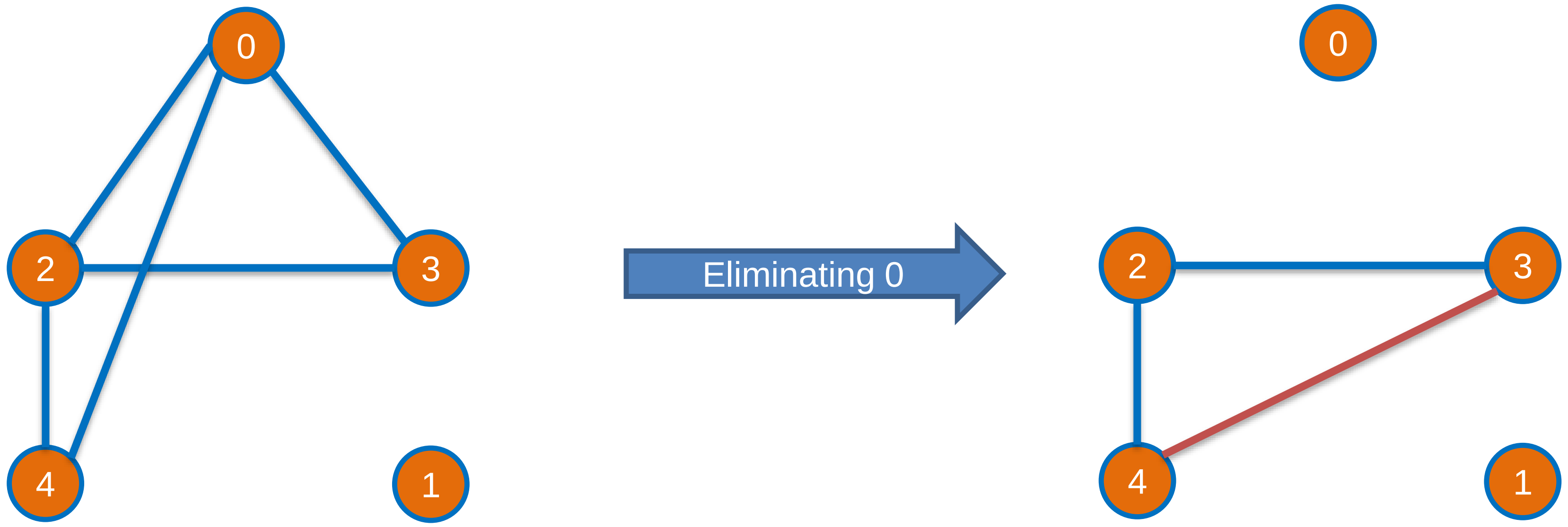


n_i is called the **Pivot.**

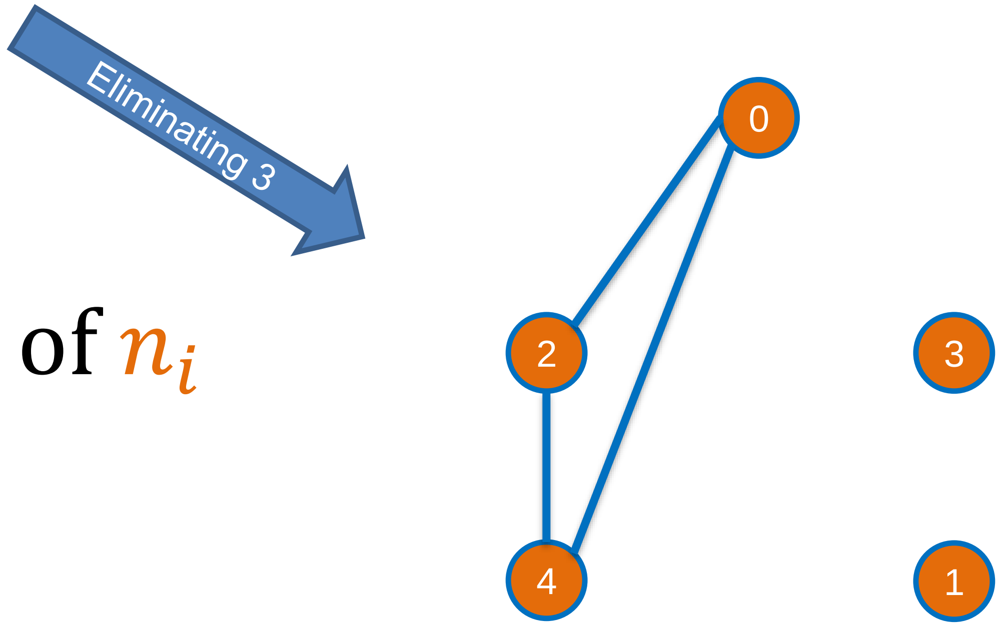
Schur Complement of n_i:

1. Add edges between neighbors of n_i
2. Eliminate edges with node n_i

Gaussian Elimination on Graphs



1 new edge

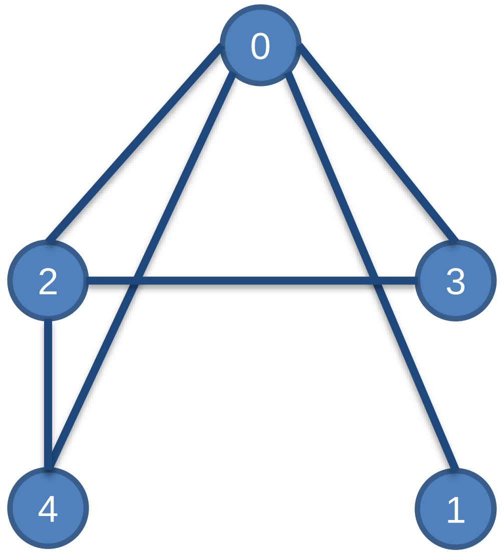


no new edges

Schur Complement of $\mathcal{N}_i$ on Graph:

1. Add edges between neighbors of n_i
2. Eliminate edges with node n_i

Gaussian Elimination on Graphs



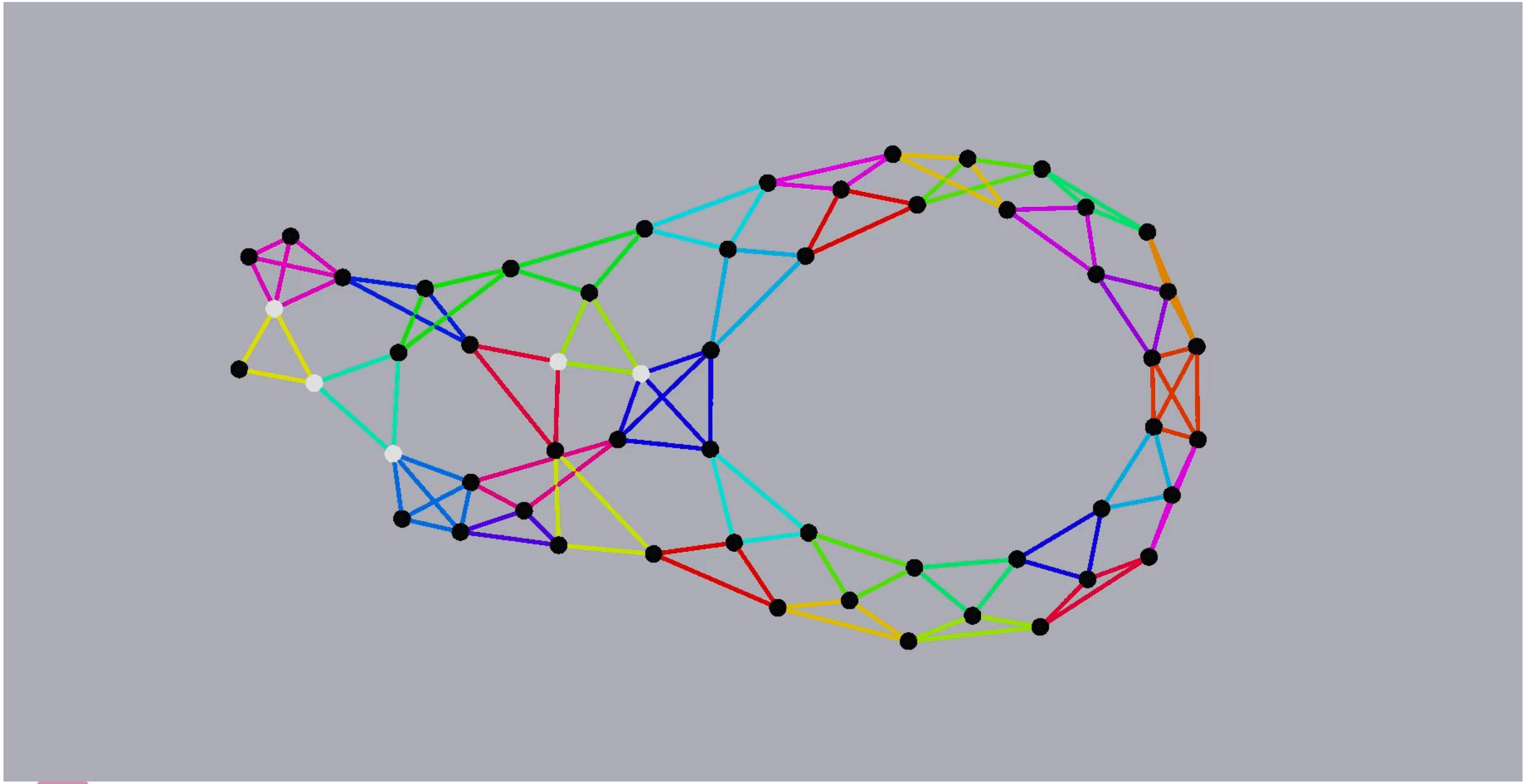
A **perfect elimination order** for this graph is:

[1, 3, 0, 2, 4]

Does not always exist!

Elimination Game: minimize the number of new edges

Finding such ordering is NP-Complete.
There are good heuristics!



Gaussian Elimination on Graphs

Heuristics for Graph Elimination:

- **Minimum Degree Algorithm (MDA)**
  - Fast but generates mediocre ordering
- **Minimum Edge Creation Algorithm (MECA)**
  - More expensive but better ordering
  - We only need to compute this once!

Gaussian Elimination on Graphs

Minimum Edge Creation Algorithm (MECA):
At each step, eliminate the pivot that creates a minimum number of new edges…

Gaussian Elimination on Graphs

Graph Elimination gives us:

1. An ordered sequence of nodes (**pivot sequence**):

$$Pivots = [n_0, n_1, \dots]$$

2. For each pivot n_i a sequence of eliminated edges (**edge sequences**):

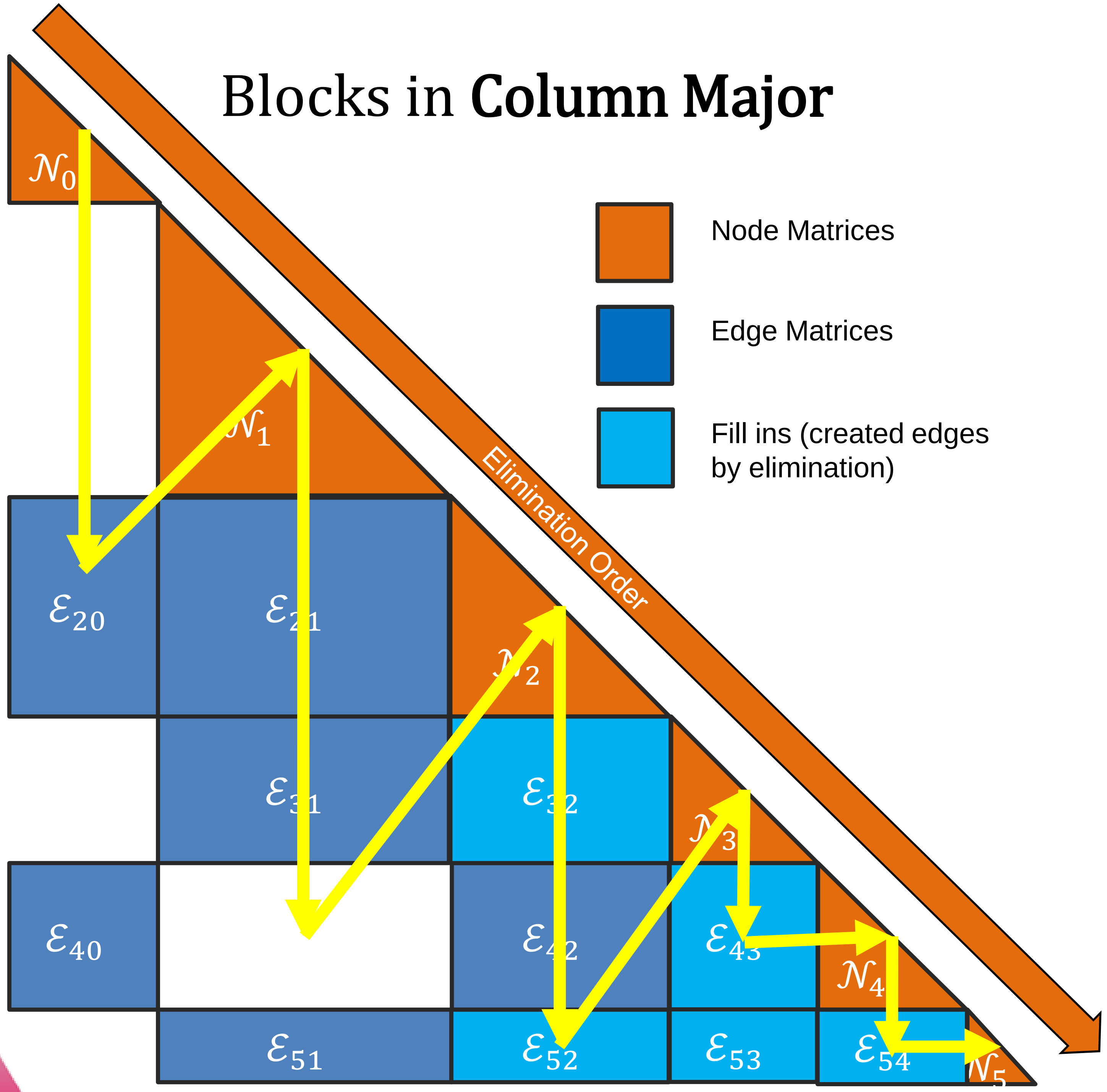
$$Elim(n_i) = [e_{j_0,i}, e_{j_1,i}, \dots]$$

These sequences should be sorted in pivot order:

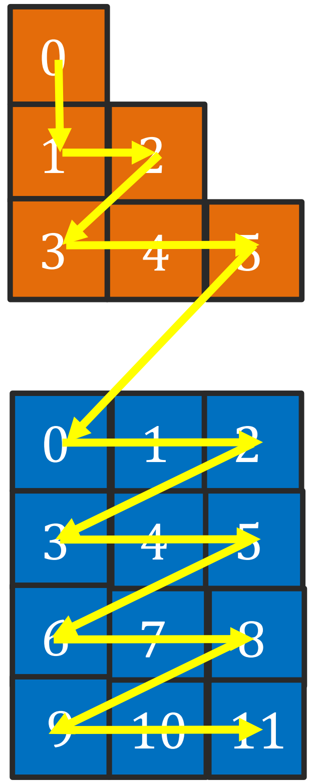
$$n_{j_0} < n_{j_1} < \cdots$$

Sparse Block Matrix Memory Layout

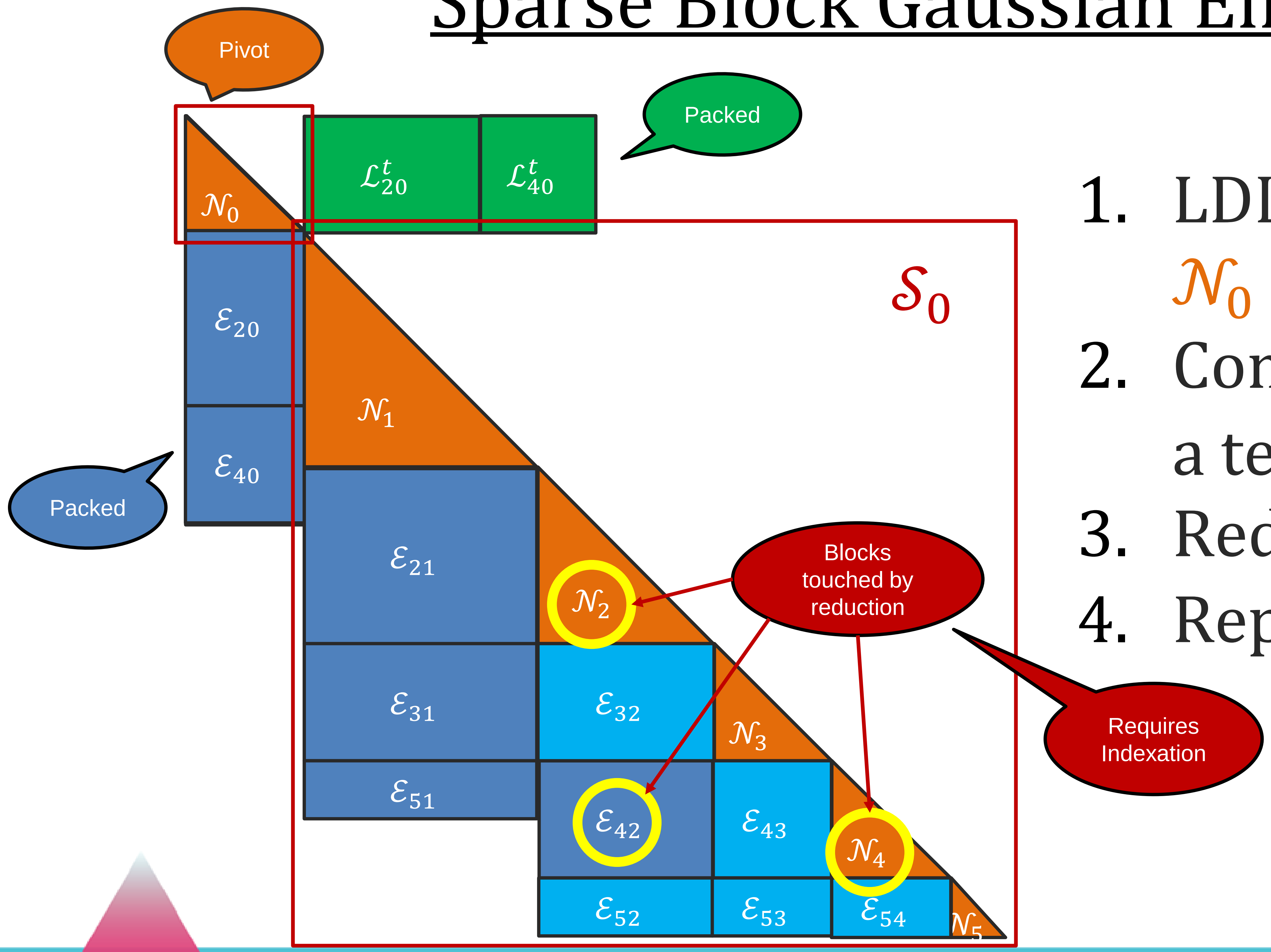
Blocks in **Column Major**



Elements of blocks in **Row Major**

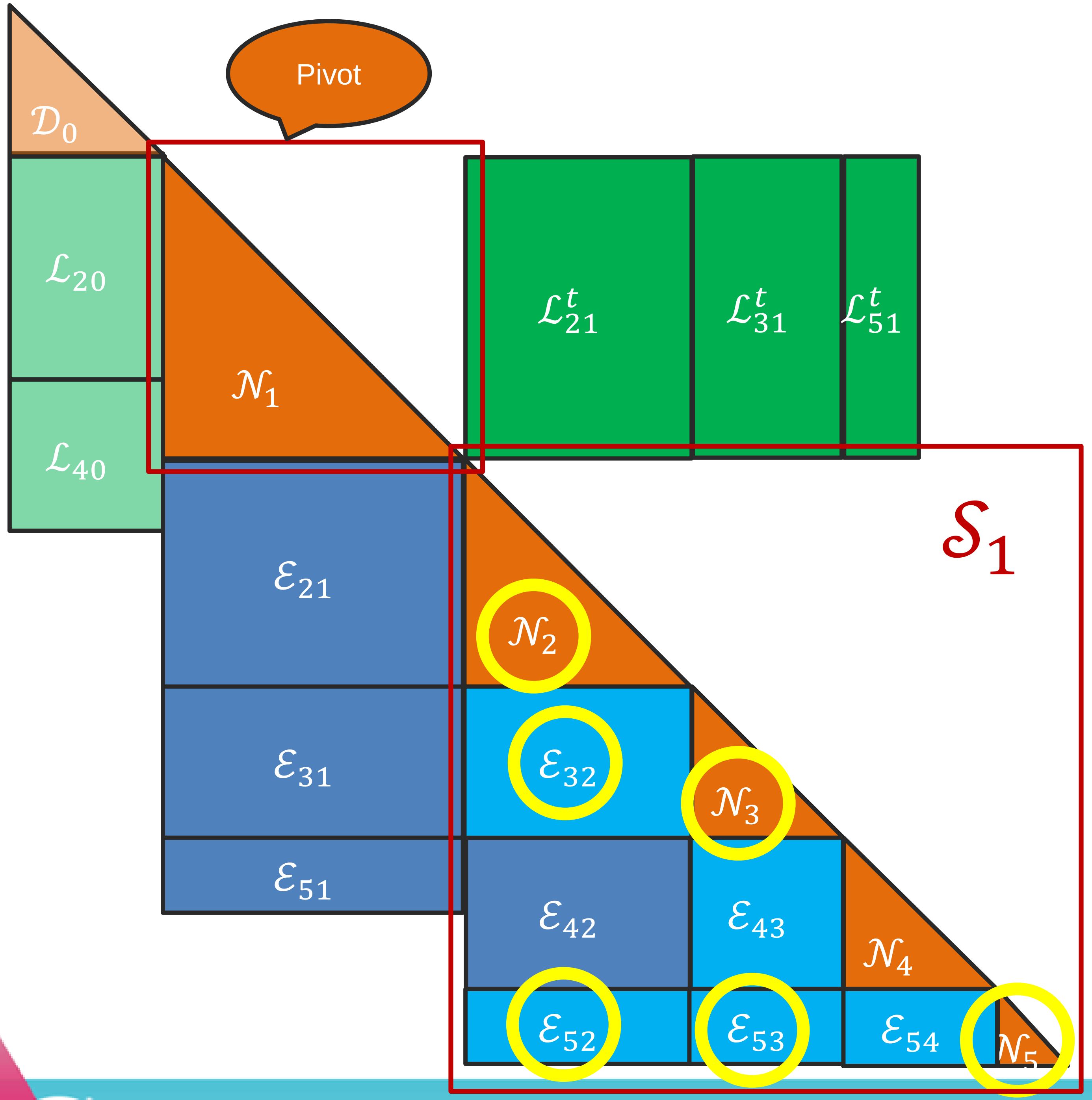


Sparse Block Gaussian Elimination



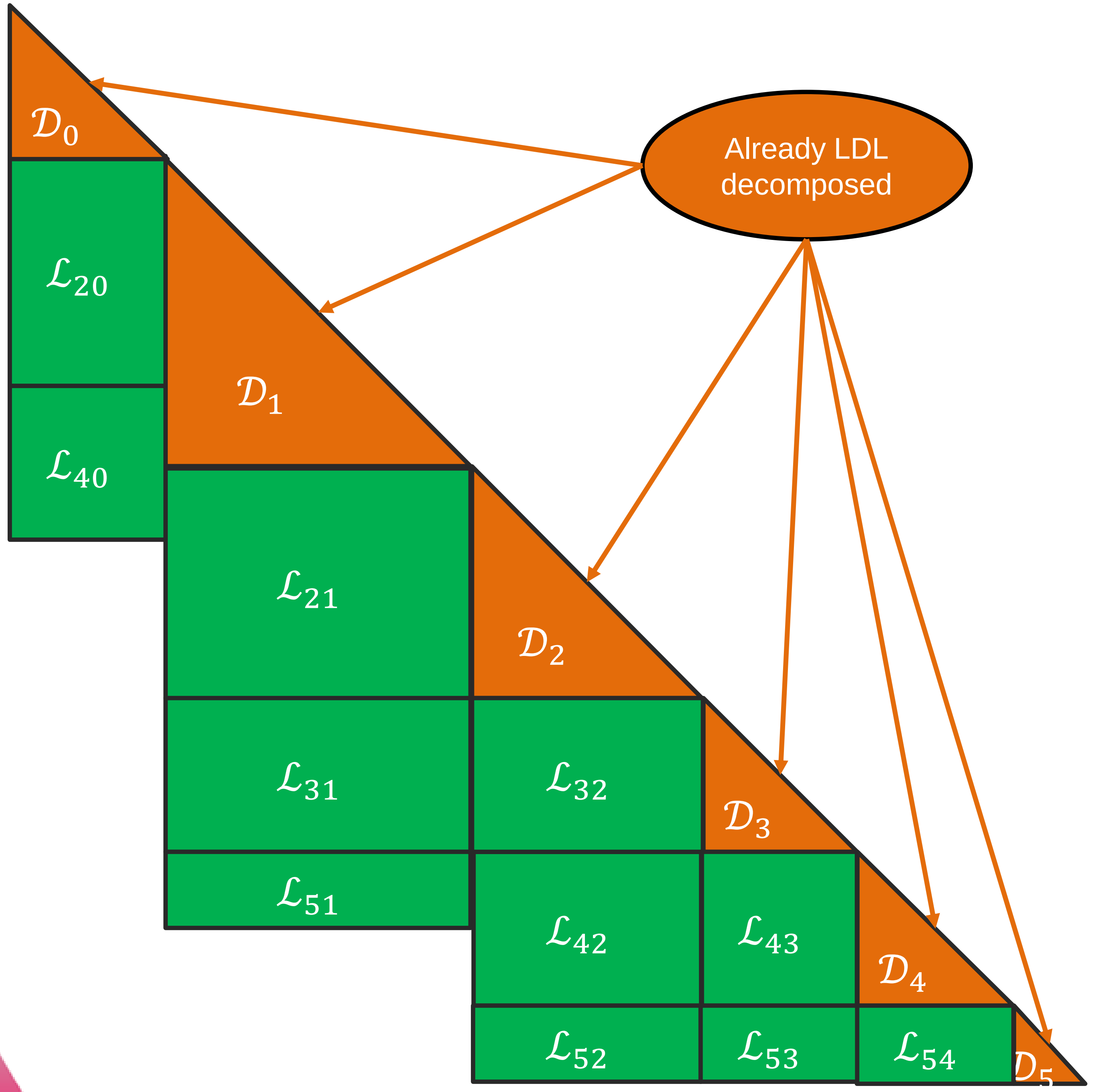
1. LDL decompose the pivot $\mathcal{N}_0$ in-place, $\mathcal{D}_0 := LDL(\mathcal{N}_0)$
2. Compute $\mathcal{L}_{*,0} = \mathcal{E}_{*,0}\mathcal{N}_0^{-1}$ in a temp buffer
3. Reduce $\mathcal{S}_0 \leftarrow \mathcal{S}_0 - \mathcal{E}_{*,0}\mathcal{L}_{*,a}^t$
4. Replace $\mathcal{E}_{*,0} \leftarrow \mathcal{L}_{*,0}$

Sparse Block Gaussian Elimination



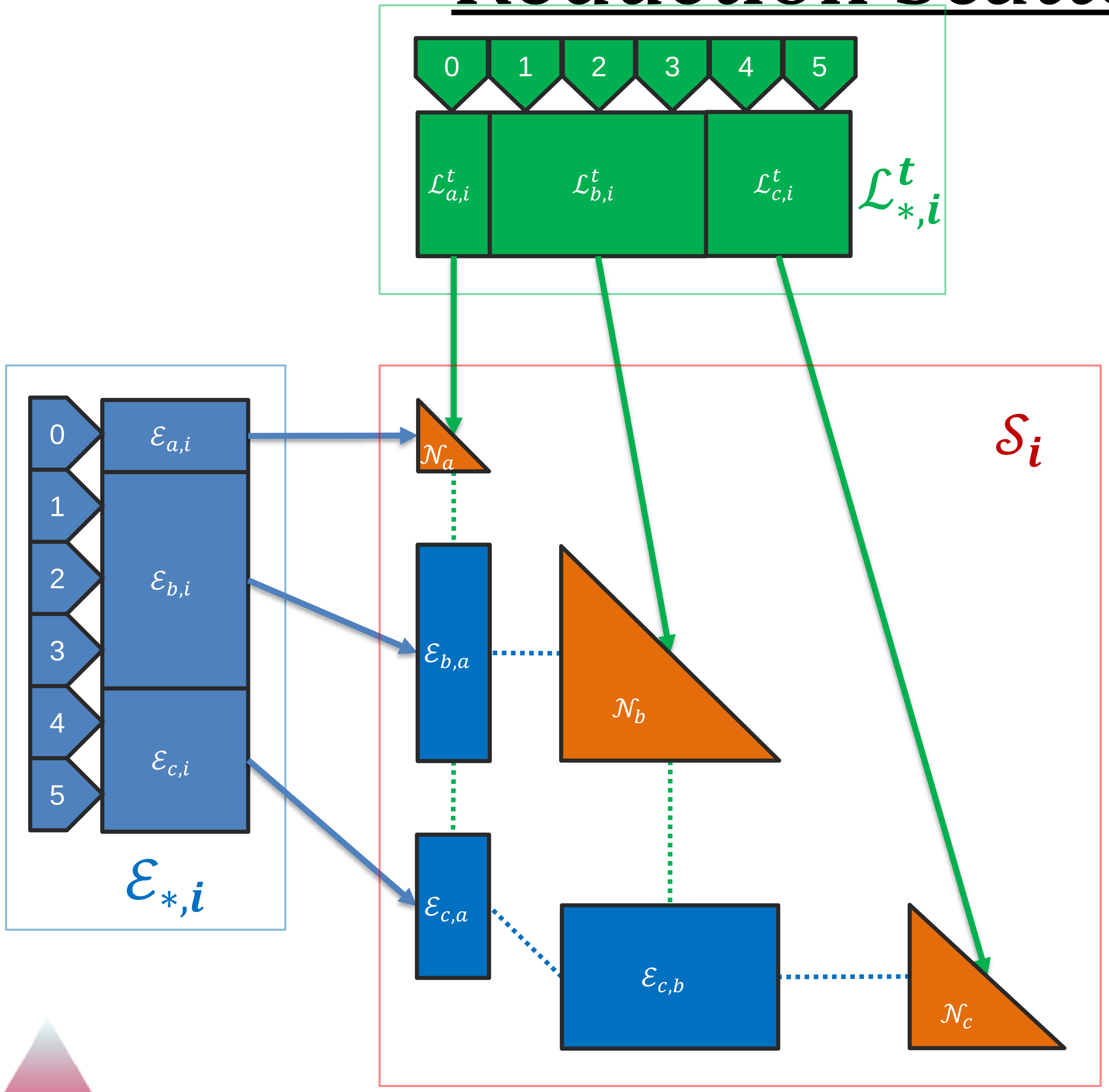
1. LDL decompose the pivot $\mathcal{N}_1$ in-place, $\mathcal{D}_1 \coloneqq LDL(\mathcal{N}_1)$
2. Compute $\mathcal{L}_{*,1} = \mathcal{E}_{*,1}\mathcal{N}_1^{-1}$ in a temp buffer
3. Reduce $\mathcal{S}_1 \leftarrow \mathcal{S}_1 - \mathcal{E}_{*,1}\mathcal{L}^t_{*,1}$
4. Replace $\mathcal{E}_{*,1} \leftarrow \mathcal{L}_{*,1}$

Sparse Block LDL Decomposition



d

Reduction Scattering Indexation



k	Row of E	Column of $\mathcal{L}^t$	$RSI\,[k]$
0	0	0	$\&\mathcal{N}_a[0,0]$
1	1	0	$\&\mathcal{E}_{b,a}[0,0]$
2	1	1	$\&\mathcal{N}_b[0,0]$
3	2	0	$\&\mathcal{E}_{b,a}[1,0]$
4	2	1	$\&\mathcal{N}_b[1,0]$
5	2	2	$\&\mathcal{N}_b[1,1]$
6	3	0	$\&\mathcal{E}_{b,a}[2,0]$
7	3	1	$\&\mathcal{N}_b[2,0]$
8	3	2	$\&\mathcal{N}_b[2,1]$
9	3	3	$\&\mathcal{N}_b[2,2]$
10	4	0	$\&\mathcal{E}_{c,a}[0,0]$
⋮	⋮	⋮	⋮
14	5	5	$\&\mathcal{N}_c[1,1]$

Size of RSI :

$$\frac{h(h+1)}{2}$$

Here:

$h = 5,$

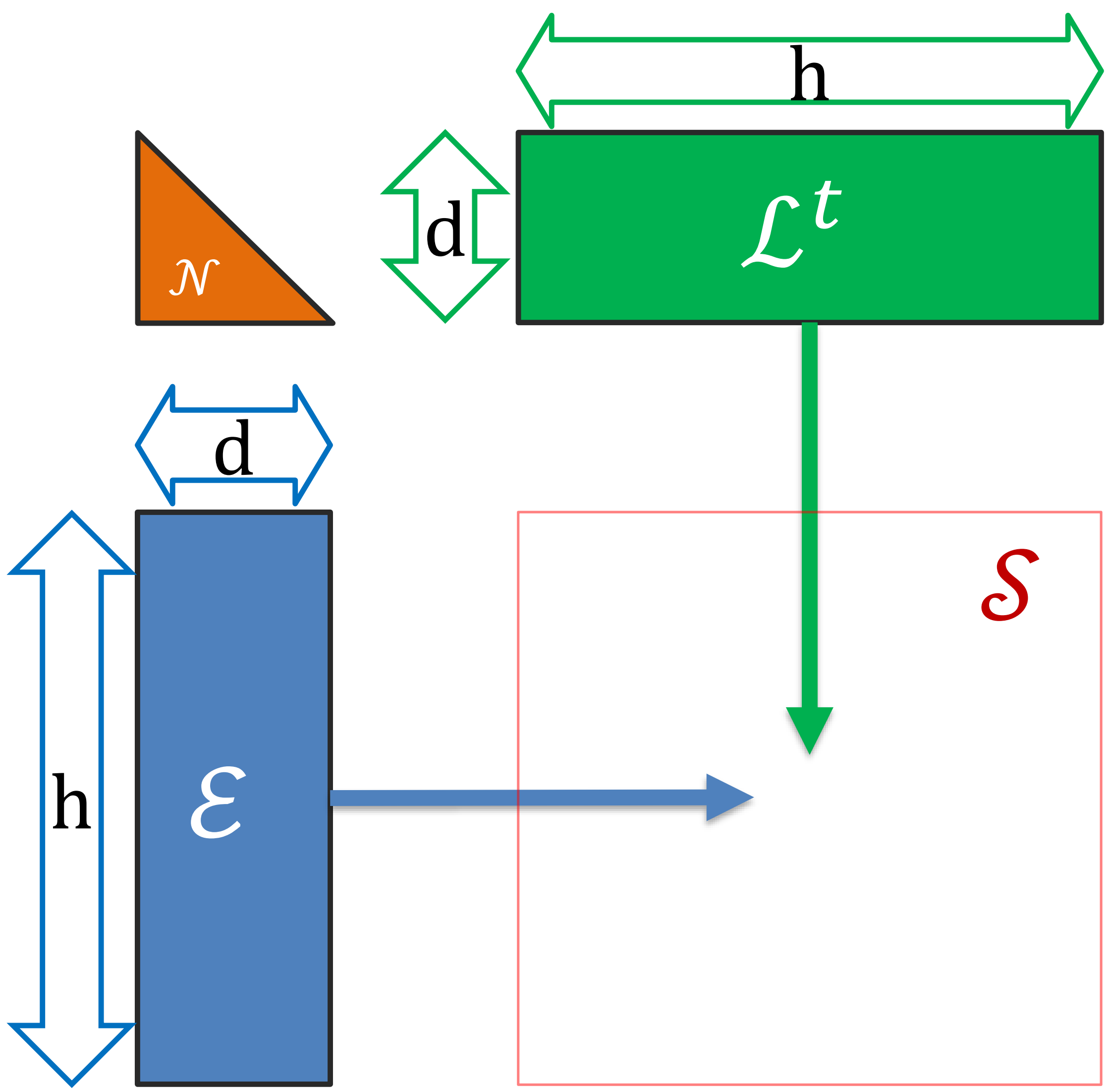
$size = 15$

RSI better use indices => less memory!

Precompute during symbolic phase.

Performance of Block LDL

$\boxed{h \gg d}$



Operation count:

1. $LDL(\mathcal{N}) : \frac{d^3}{6}$
2. $\mathcal{L} = \mathcal{E}\mathcal{N}^{-1} : hd^2$
3. $\mathcal{S} = \mathcal{S} - \mathcal{E}\mathcal{L}^t : \boxed{d\,\frac{h(h+1)}{2}}$

Memory Access:

1. $LDL(\mathcal{N}) : \frac{d(d+1)}{2}$
2. $\mathcal{L} = \mathcal{E}\mathcal{N}^{-1} : 2hd$
3. $\mathcal{S} = \mathcal{S} - \mathcal{E}\mathcal{L}^t : 2\text{hd} \boxed{+ \frac{h(h+1)}{2}}$

Dominant

Modern processors:

- Good at floating point operations
- Bad at memory access

$$\frac{\text{Op Count}}{\text{Mem Access}} \sim d$$

$\Rightarrow$ **Fastest for large d, as long as $d \ll h$**

Reduction Step: 70% CPU Time

Edge Matrix Width

Schur Complement

Edge Matrix

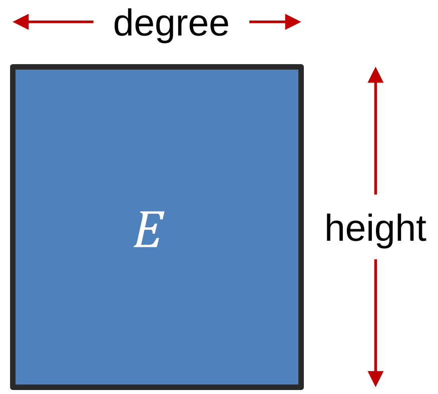
L Matrix

Reduction Scattering Indexation

Edge Matrix Height

```
template<uint degree>
void reduce(float* S, const float* E, const float* L, const uint* rsi, uint height)
{
    for (uint i = 0; i < height; i++)
    {
        for (uint j = 0; j <= i; j++)
        {
            S[ *rsi ] -= dotProduct<degree>(E + i*Dim, L + j*Dim);
            rsi++;
        }
    }
}
```

Loop Unroll and Inline!



Implementation

```
template<uint deg> void eliminatePivot (float* S, float* LTemp, float* N, float* E,
    const uint* rsi, uint height)
{
    // Dense LDL decomposition of N
    ldlDecompose<deg>(N);

    // Compute L = E * N^-1
    computeL<deg>(LTemp, N, E, height);

    // Reduce S -= E * L^t
    reduce<deg>(S, E, LTemp, rsi, height);

    // Don’t need the Edge Matrix anymore
    // Overwrite with L Matrix
    copy(E, LTemp, height * deg);
}
```

~5% of CPU Time
No Unrolling Necessary

~25% of CPU
Time… Unroll!

~70% of CPU
Time

Implementation

Use switch statement:

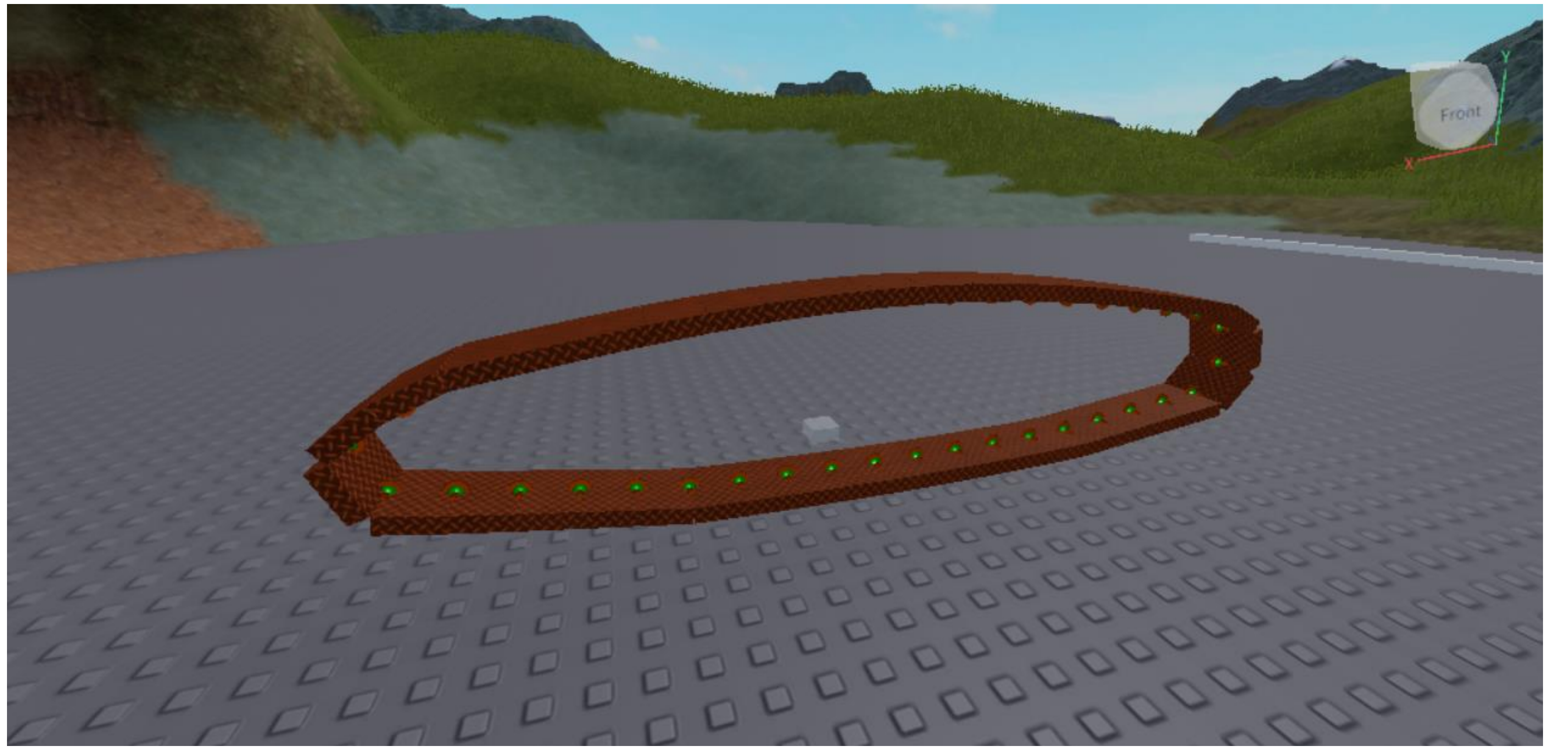
```
for (const Pivot& pivot : pivots)
{
    switch (pivot.degree)
    {
    case 0: break;
    case 1: eliminatePivot<1>(…); break;
    case 2: eliminatePivot<2>(…); break;
    case 3: eliminatePivot<3>(…); break;
    case 4: eliminatePivot<4>(…); break;
    case 5: eliminatePivot<5>(…); break;
    case 6: eliminatePivot<6>(…); break;
    default: eliminatePivot(…, pivot.dimension); break; // Not templated for Degree > 6
    }
}
```

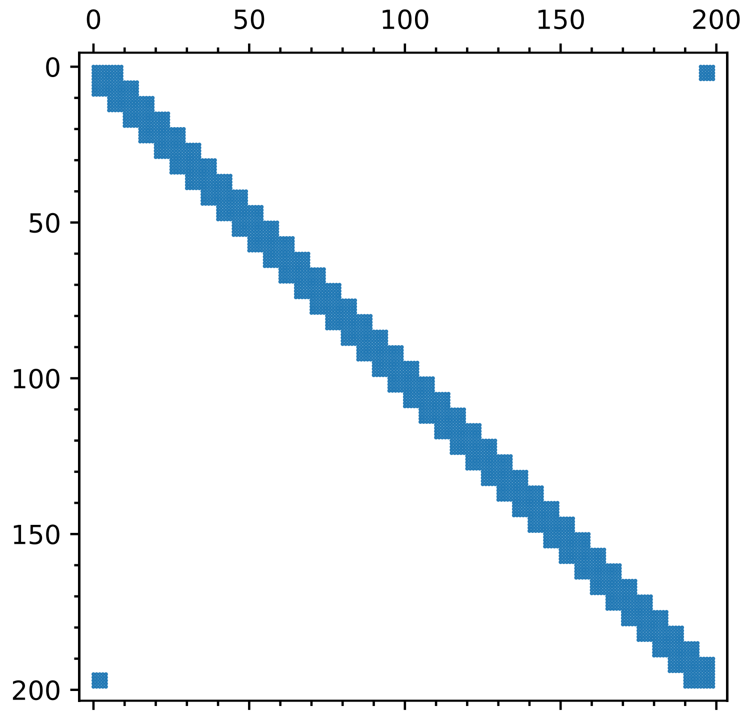
Tracked Vehicle

3 Components:

- Main body
- Left track
- Right track

Track



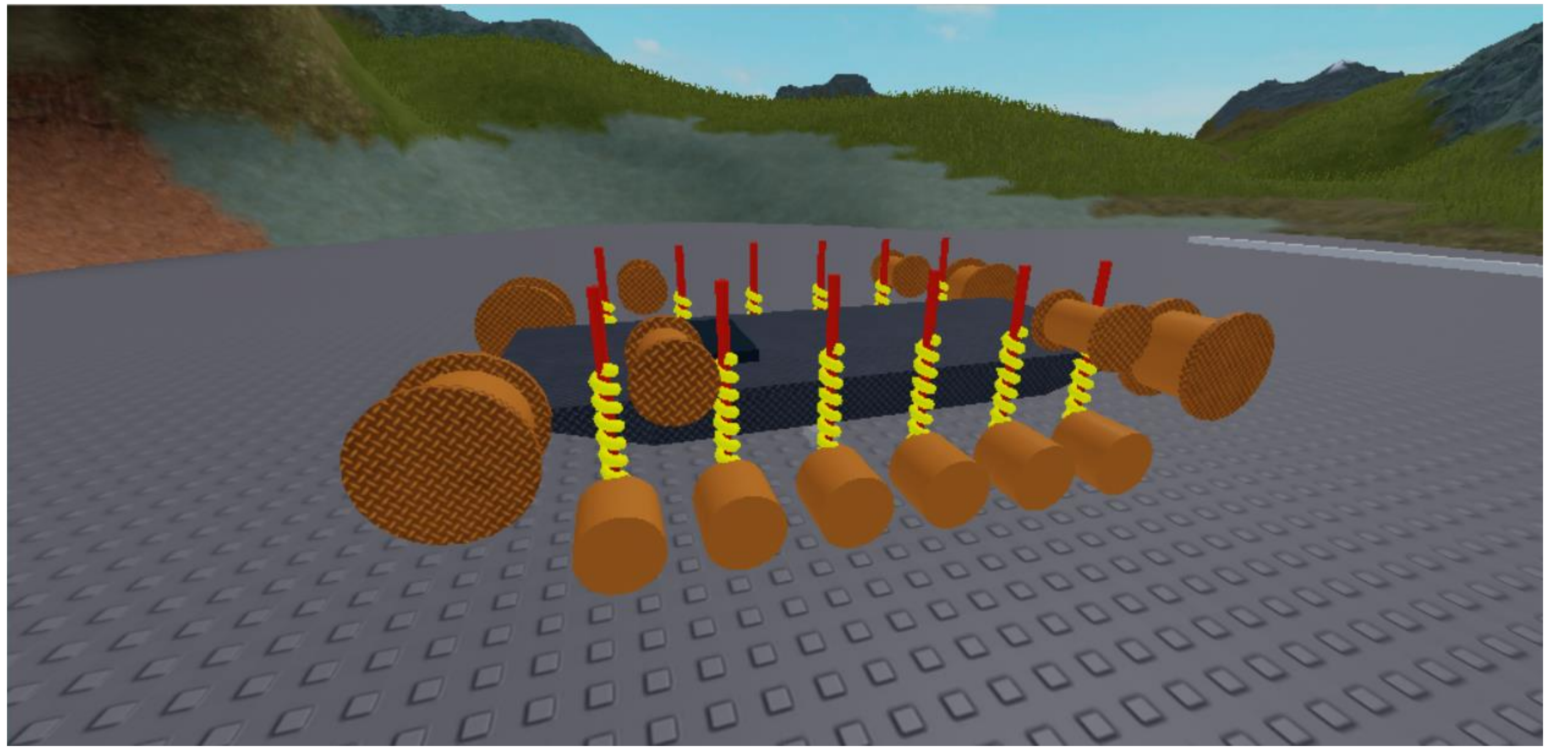


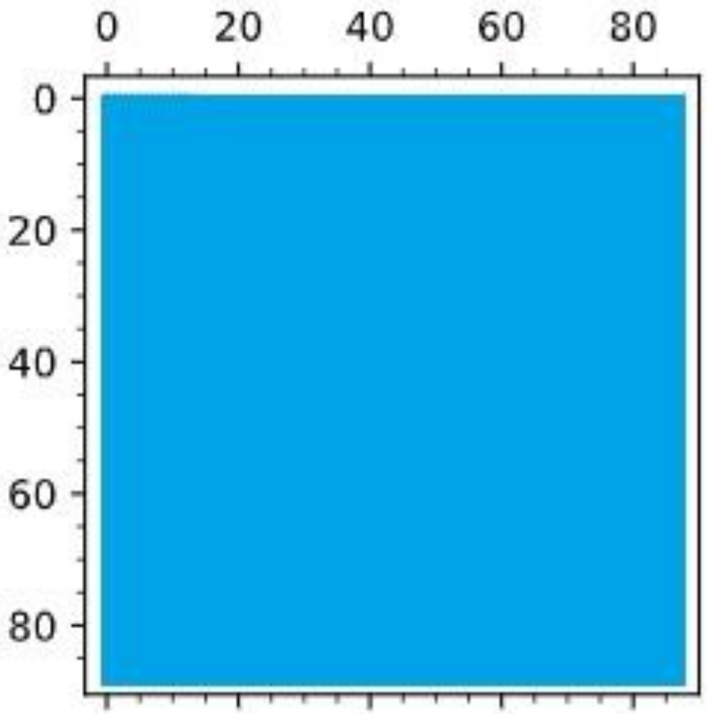
$$\#Constraints = 40$$
$$Dim(\mathcal{H}) = 200$$
$$Density(\mathcal{H}) = 7.5\%$$

Sparse Cholesky LDL:

$$Flops = 43k$$
$$Performance = 35\mu s$$

Multi-Wheeled Vehicle

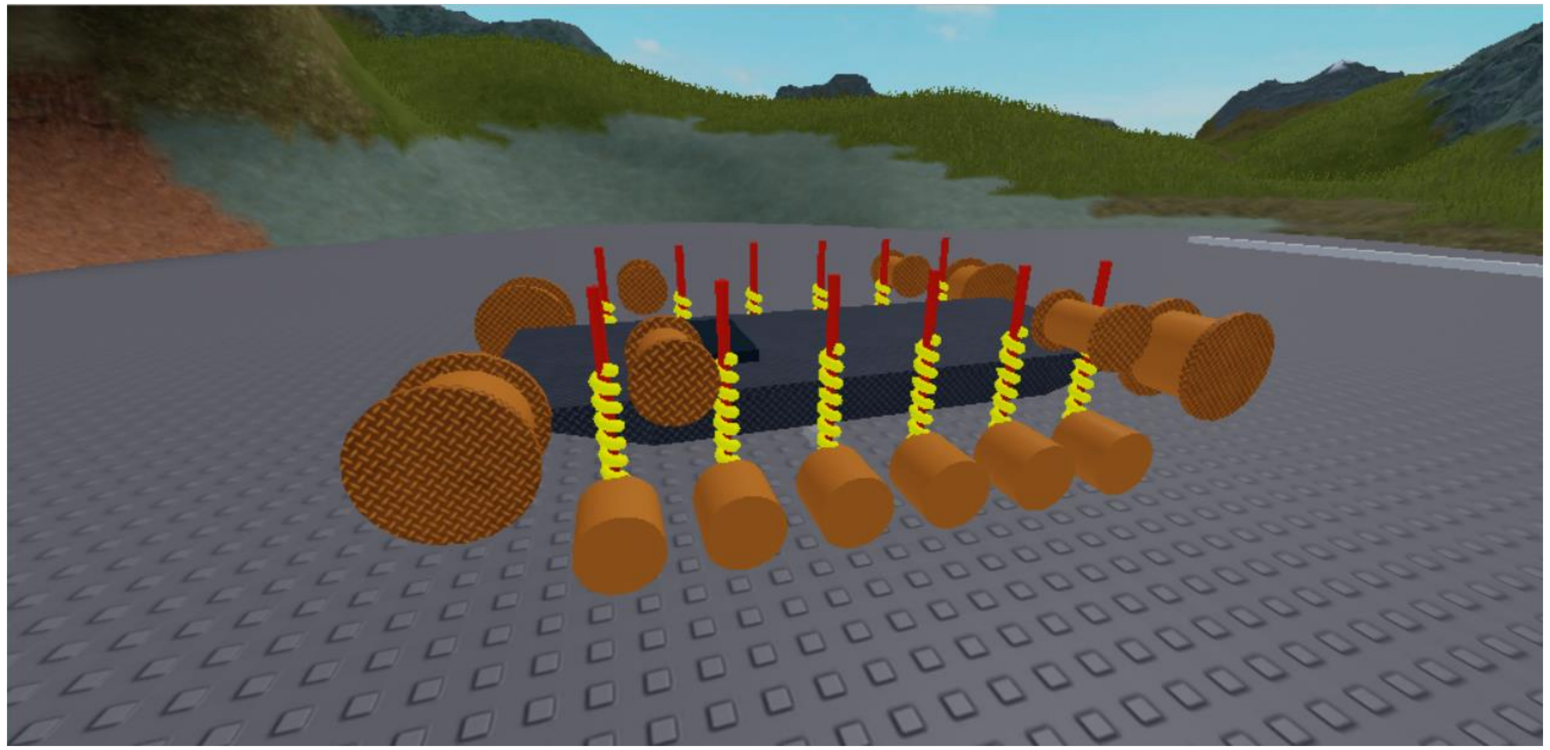




$$\#Constraints = 20\ (= n)$$
$$Dim(\mathcal{H}) = 88$$
$$Density(\mathcal{H}) = 100\%$$
$$Flops = 250k = O(n^3)$$

There is a solution: **Body Shattering**

Multi-Wheeled Vehicle

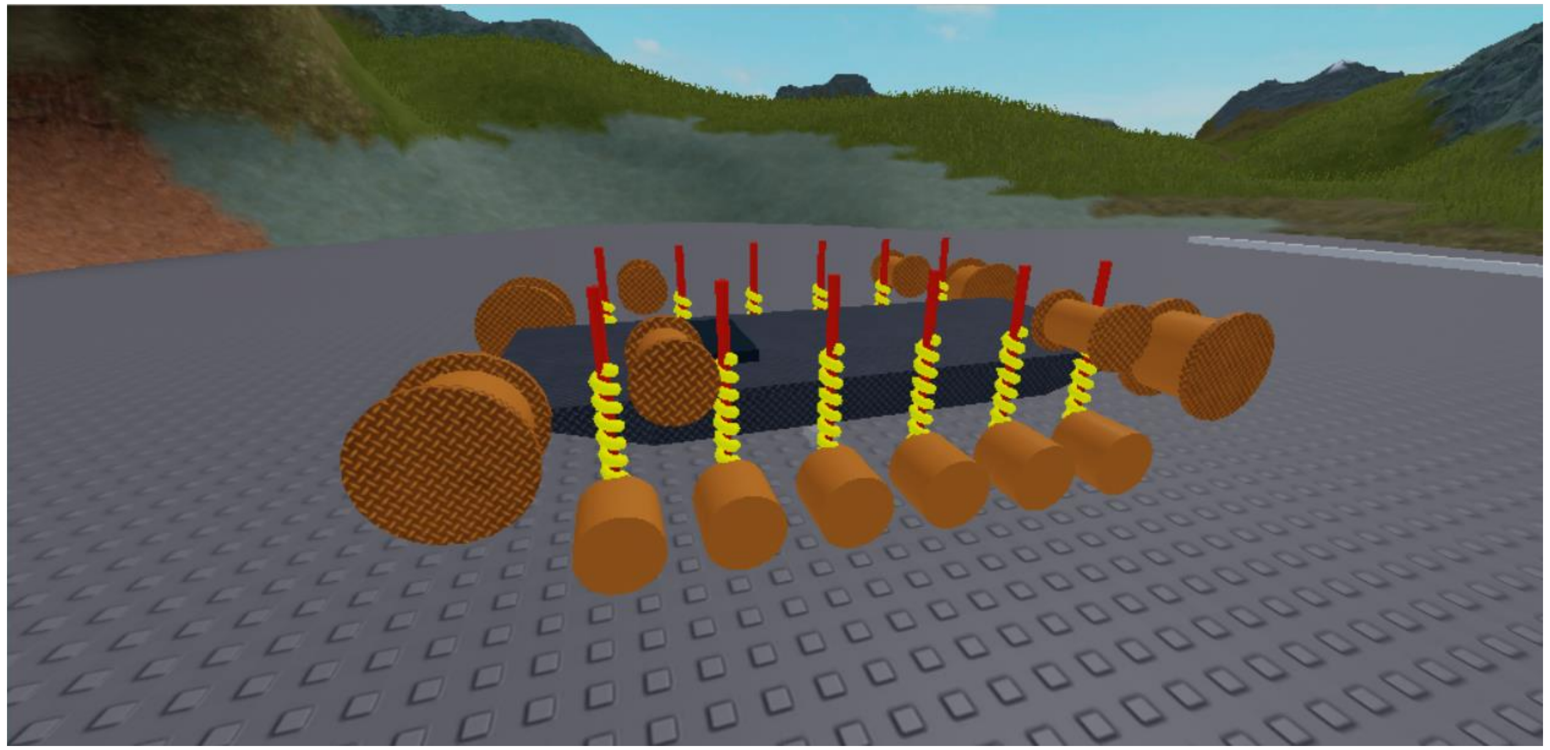


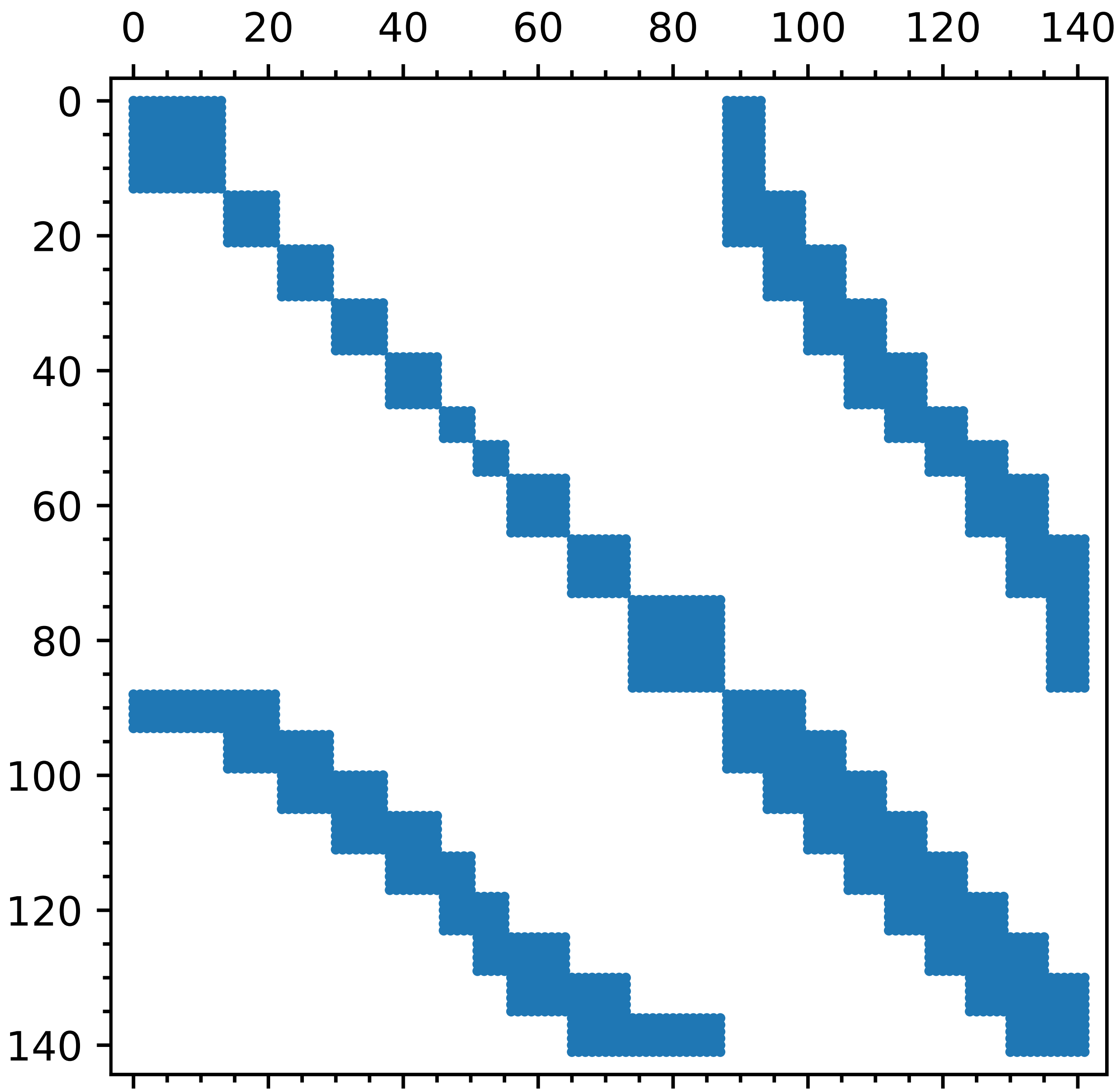
$$\#Constraints = 20\ (= n)$$
$$Dim(\mathcal{H}) = 88$$
$$Density(\mathcal{H}) = 100\%$$
$$Flops = 250k = O(n^3)$$

Body Shattering

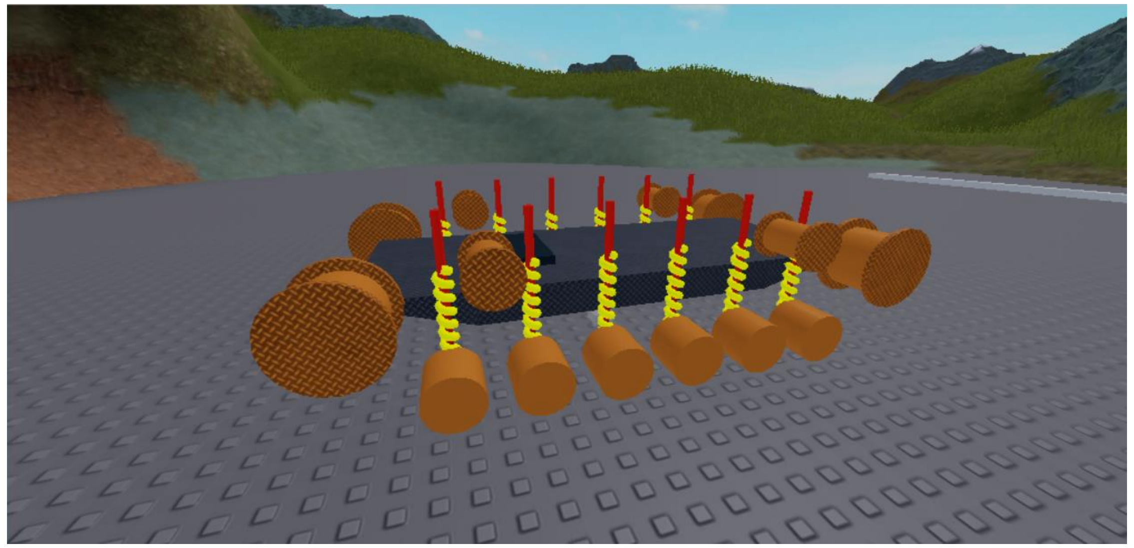
$$\#Constraints = 29$$
$$Dim(\mathcal{H}) = 142$$
$$Density(\mathcal{H}) = 17.5\%$$
$$Flops = 35k = O(n)$$
$$Performance = 34\ \mu s$$

Shattered Multi-Wheeled Vehicle





Dense Submatrices and Body Shattering



Large dense submatrix

↕

Body with many constraints

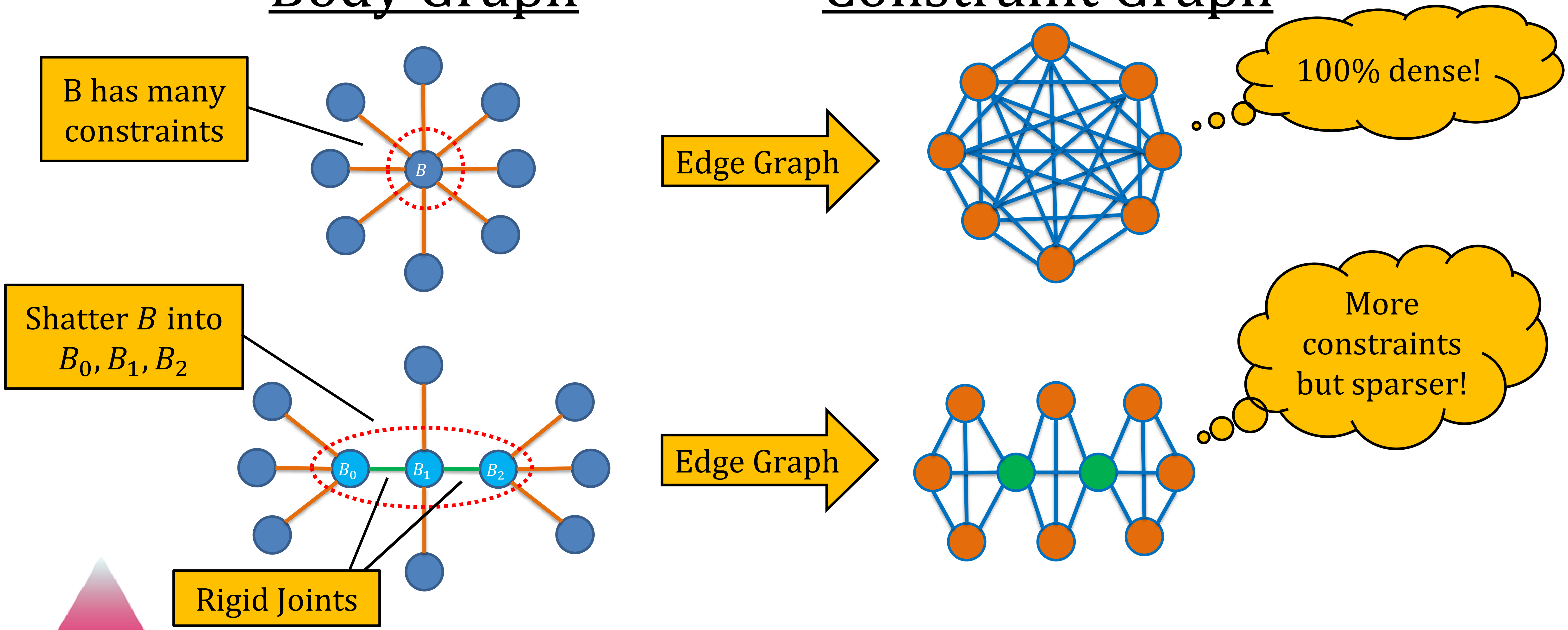
Body Shattering:

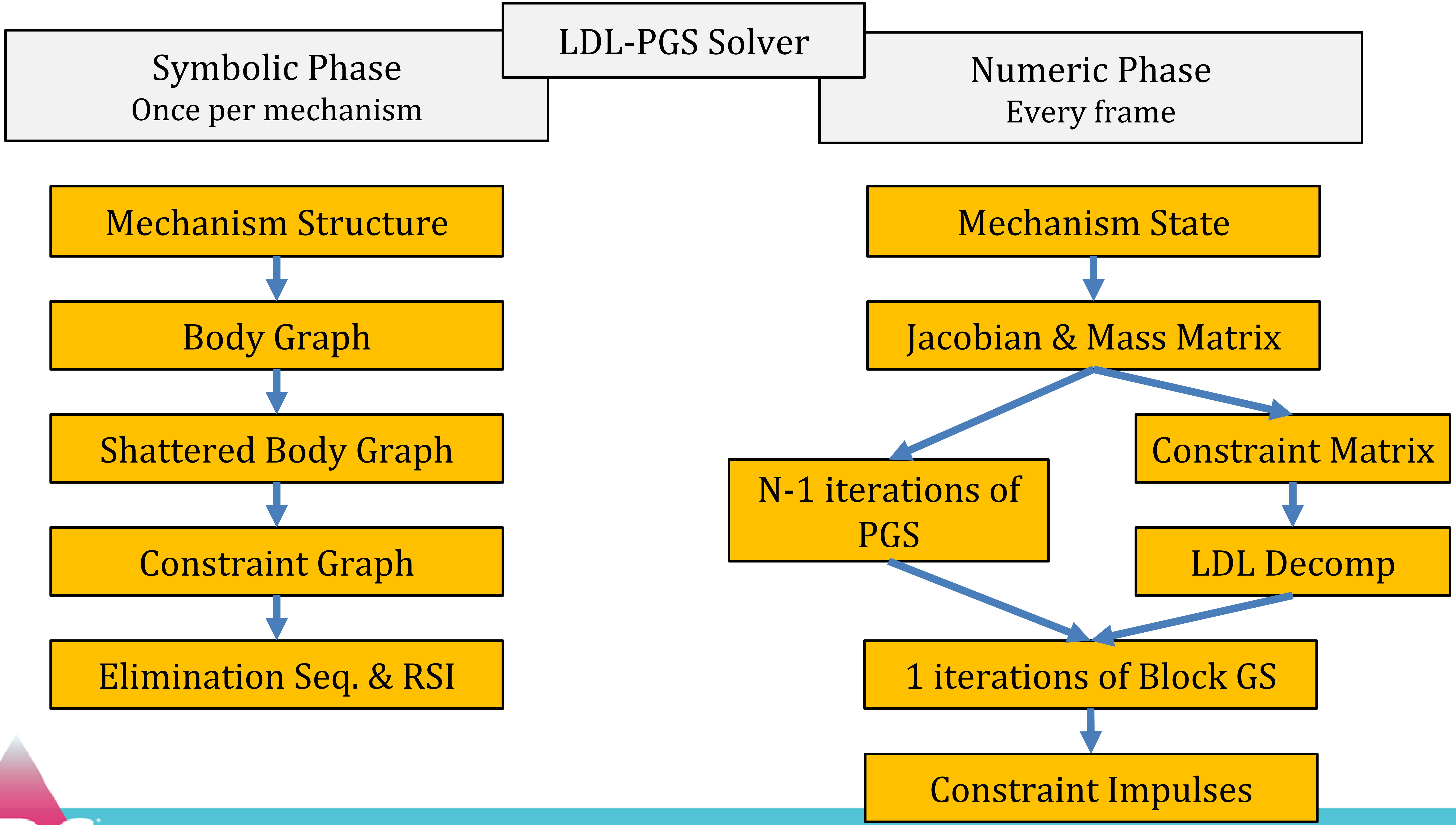
- Find bodies with many constraints (total degree > 20)
- Split into smaller equal shards
- Join using rigid joints
- Distribute constraints over the shards

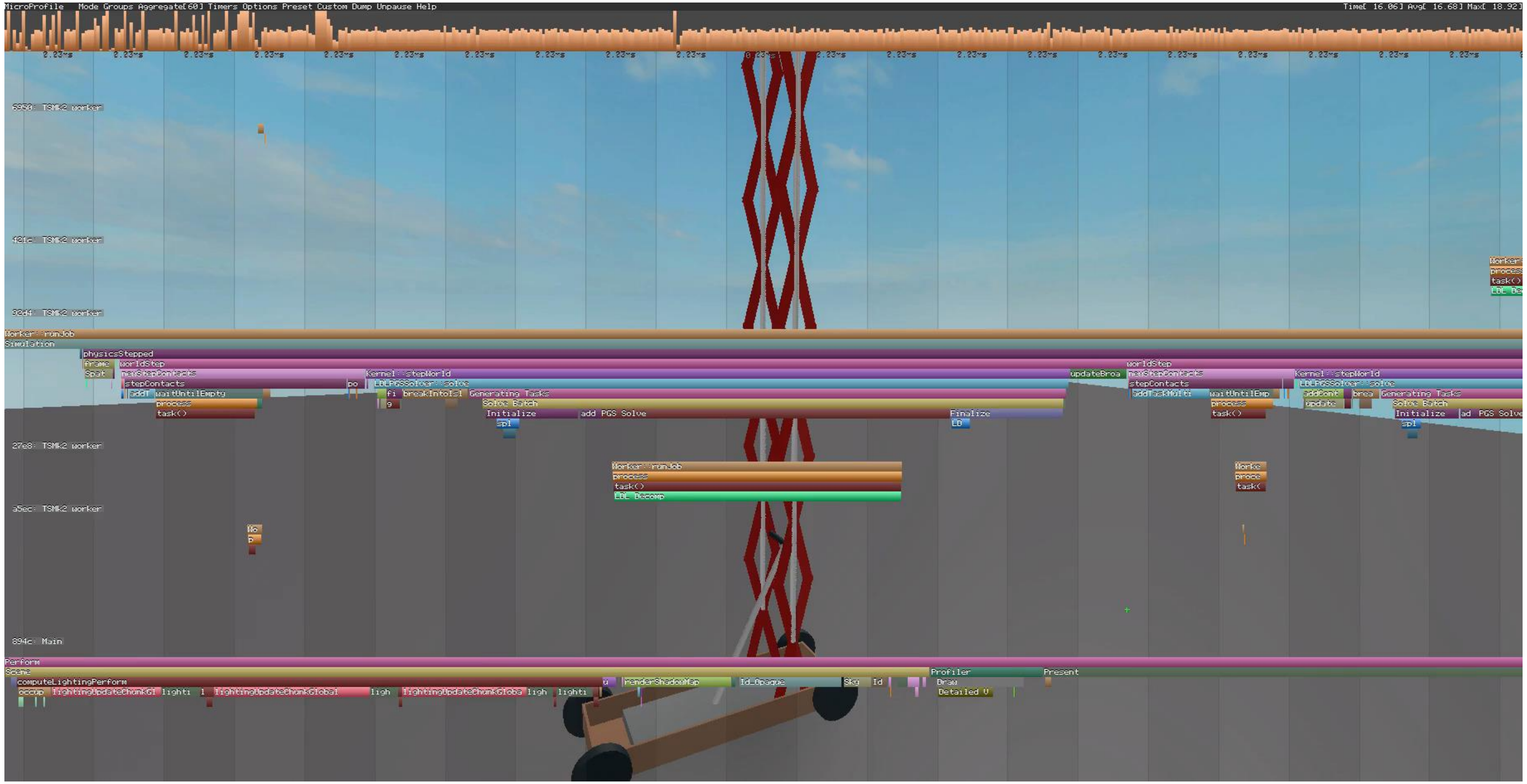
Body Shattering

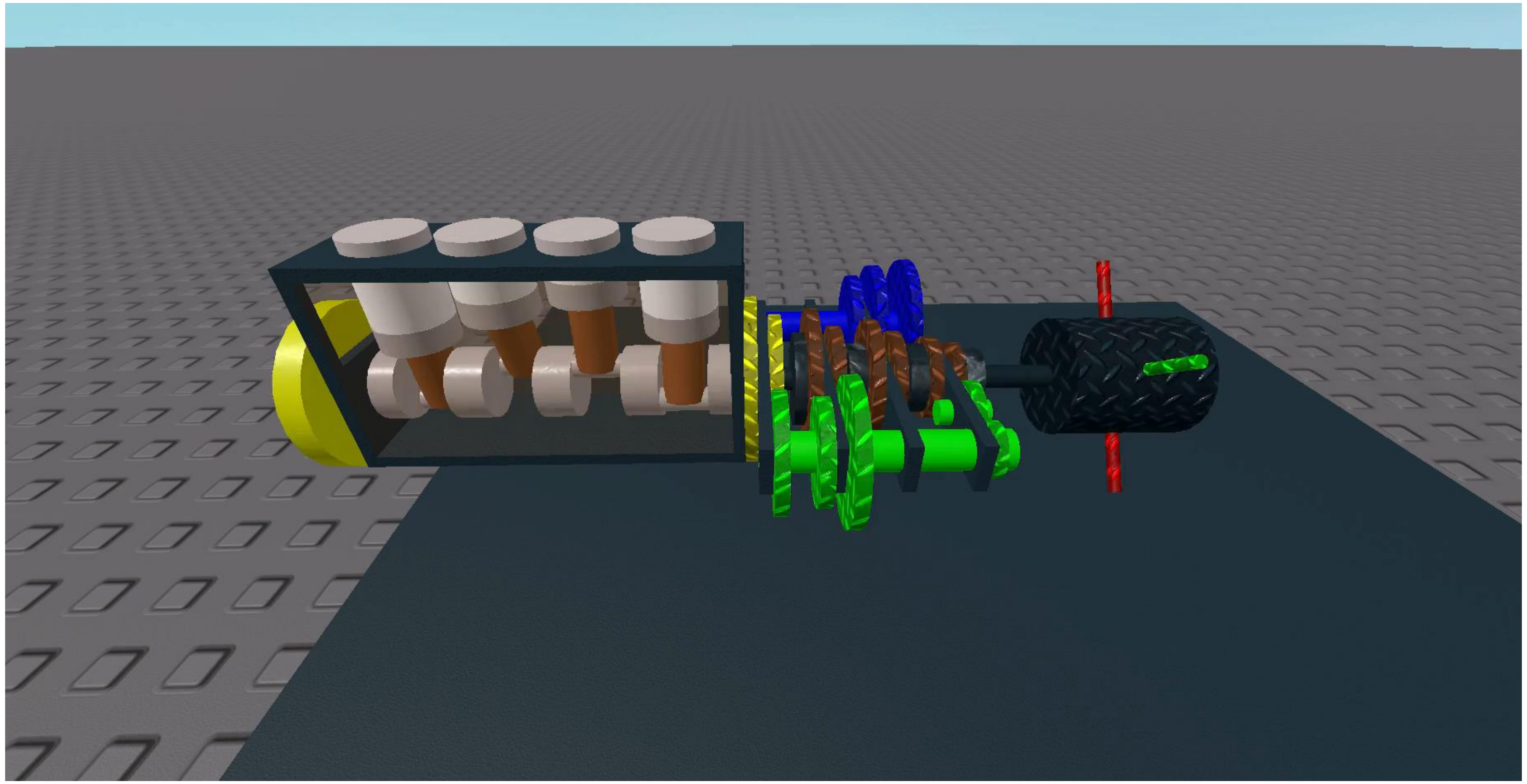
Body Graph

Constraint Graph




LDL-PGS Solver
Symbolic Phase
Once per mechanism
Numeric Phase
Every frame
Mechanism Structure
Body Graph
Shattered Body Graph
Constraint Graph
Elimination Seq. & RSI
Mechanism State
Jacobian & Mass Matrix
N-1 iterations of PGS
Constraint Matrix
LDL Decomp
1 iterations of Block GS
Constraint Impulses


MicroProfile Mode Groups Aggregate[60] Timers Options Preset Custom Dump Unpause Help
Time[16.06] Avg[16.68] Max[18.92]
physicsStepped
worldStep
Generating Tasks
Initialize
add PGS Solve
Finalize
Worker::runJob
computeLightingPerform
Profiler
Present



References

- Tim Davis, “Direct Methods for Sparse Linear Systems”
- Tim Davis’s Lectures on YouTube
- Kenny Erleben, “Physics Based Animation”
- Bullet Physics Forum

Many thanks to Roblox and the Simulation Team for caring about awesome physics!