Networks and large scale optimization

Open Data Science Conference
Boston, May 2018

Sam Safavi

On behalf of José Bento

Outline

- Why is optimization important?
- Large scale optimization
- Message-passing solver
- Benefits
- Application examples

Why is optimization important?

$$a, b = \text{data}$$
$$\theta = \text{parameters}$$

Machine learning examples:

- **Lasso** regression shrinkage and selection

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \left(\theta^T a_i - b_i\right)^2 + \lambda \|\theta\|_1$$

- **Sparse inverse covariance** estimation with the graphical lasso

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathrm{trace}(\theta a_i a_i^T) - \log\det\theta + \lambda \|\theta\|_1$$

- **Support-vector networks**

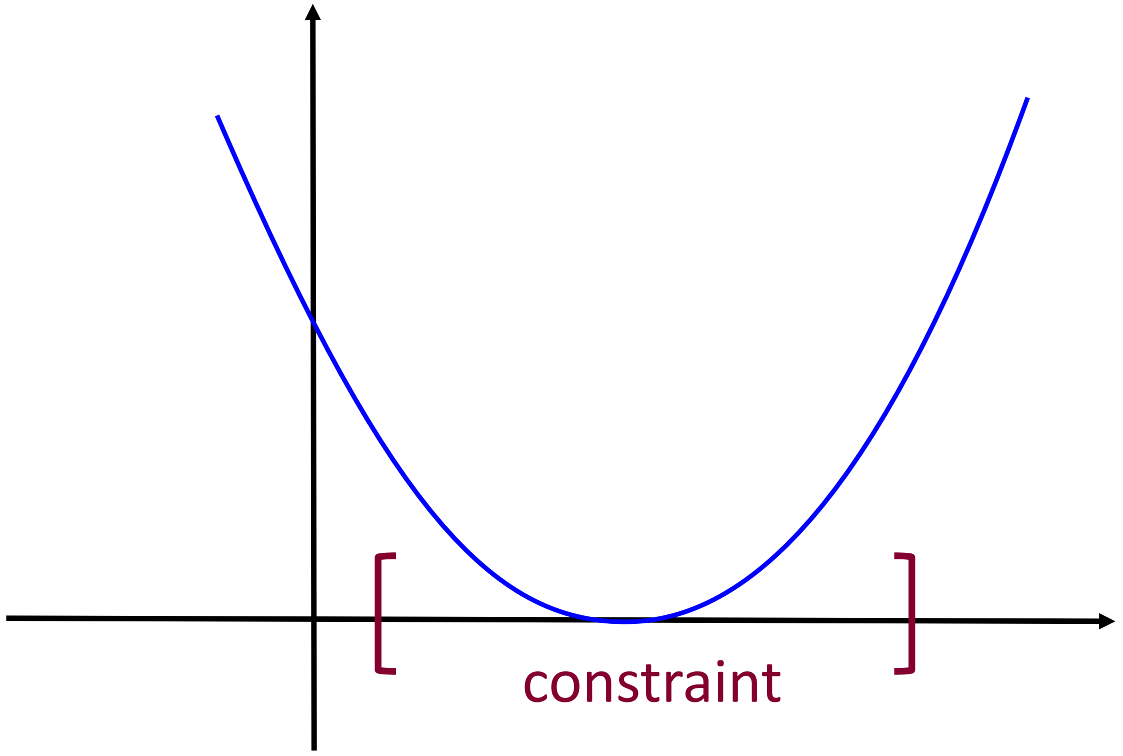
$$\min_{\theta, \theta'} \frac{1}{N} \sum_{i=1}^{N} \max\{0, 1 - b_i(\theta^T a_i + \theta')\}$$

The Alternating Direction Method of Multipliers (ADMM)

$$\text{minimize } f_1 + f_2 + f_3 + \ldots$$

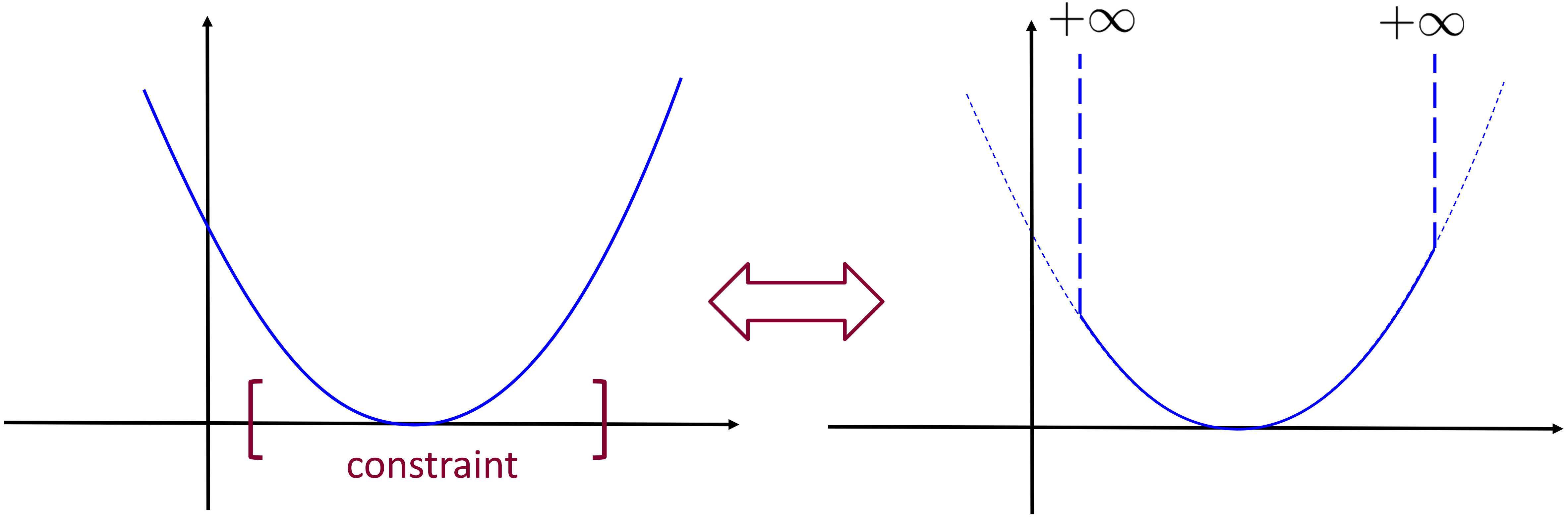
The Alternating Direction Method of Multipliers (ADMM)

$$\text{minimize } f_1 + f_2 + f_3 + \ldots$$



The Alternating Direction Method of Multipliers (ADMM)

$$\text{minimize } f_1 + f_2 + f_3 + \ldots$$



Large scale optimization

A simple example:

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$

Step1: Build Factor Graph

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$

Step1: Build Factor Graph

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$

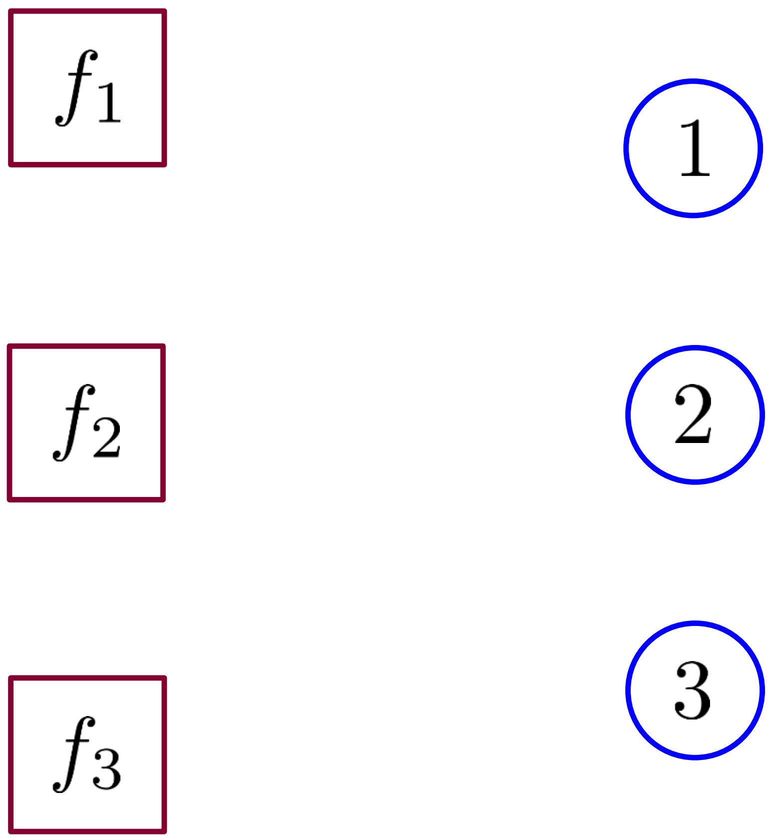
$\boxed{f_1}$

$\boxed{f_2}$

$\boxed{f_3}$

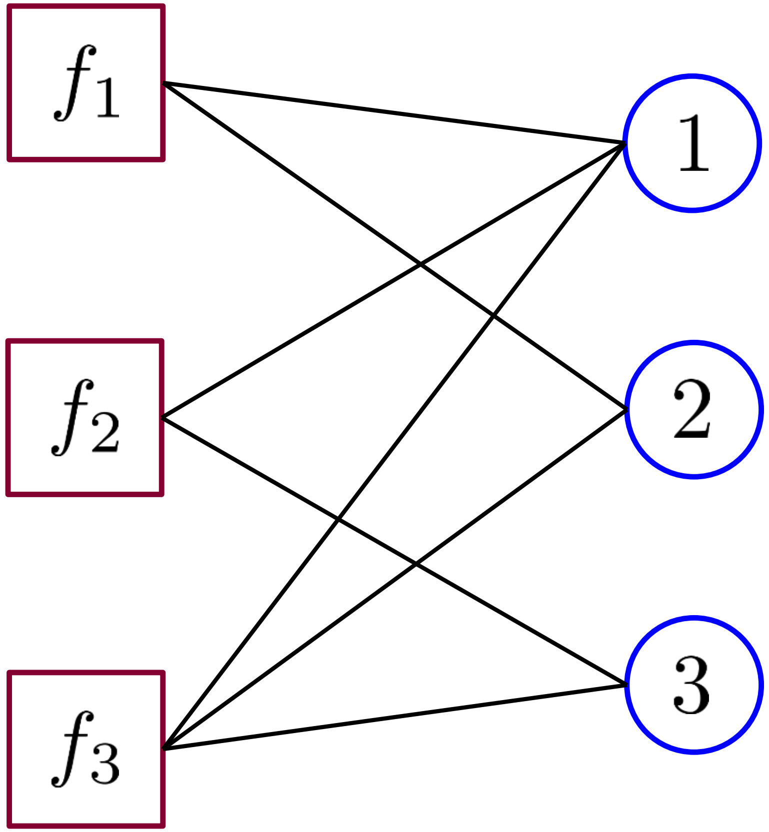
Step1: Build Factor Graph

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$



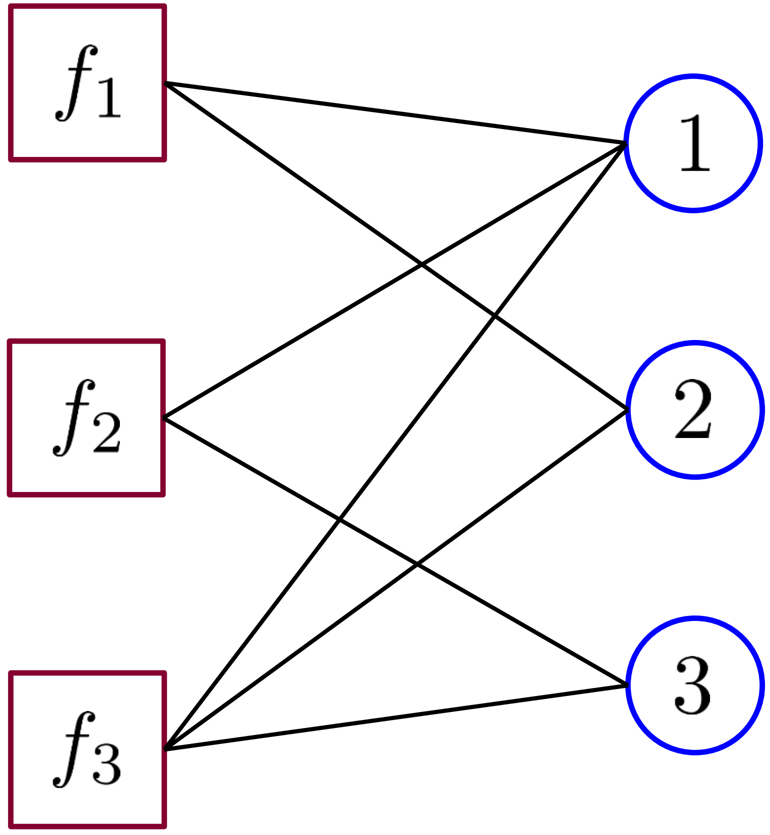
Step1: Build Factor Graph

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$



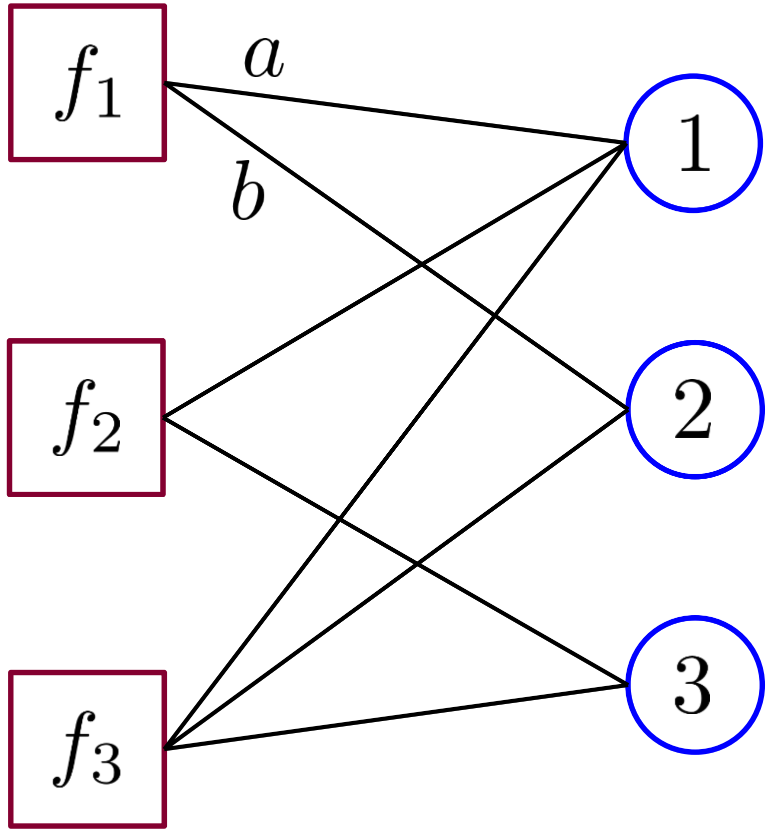
Step 2: Iterative message-passing scheme

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$



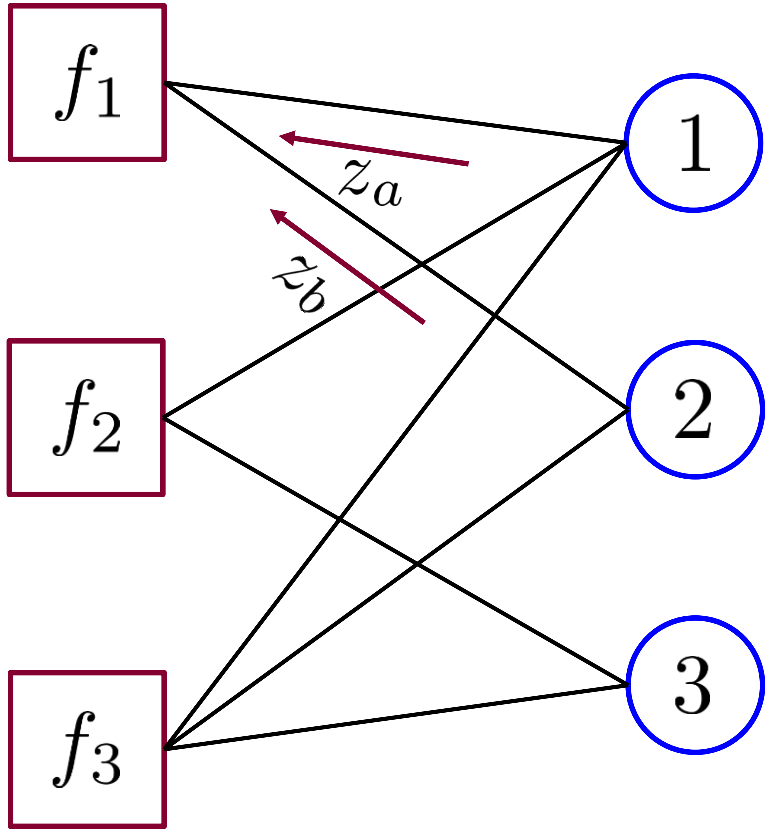
Step 2: Iterative message-passing scheme

$$\underset{z_1,z_2,z_3\in\mathbb{R}}{\text{minimize}}\ f_1(z_1,z_2) + f_2(z_1,z_3) + f_3(z_1,z_2,z_3)$$



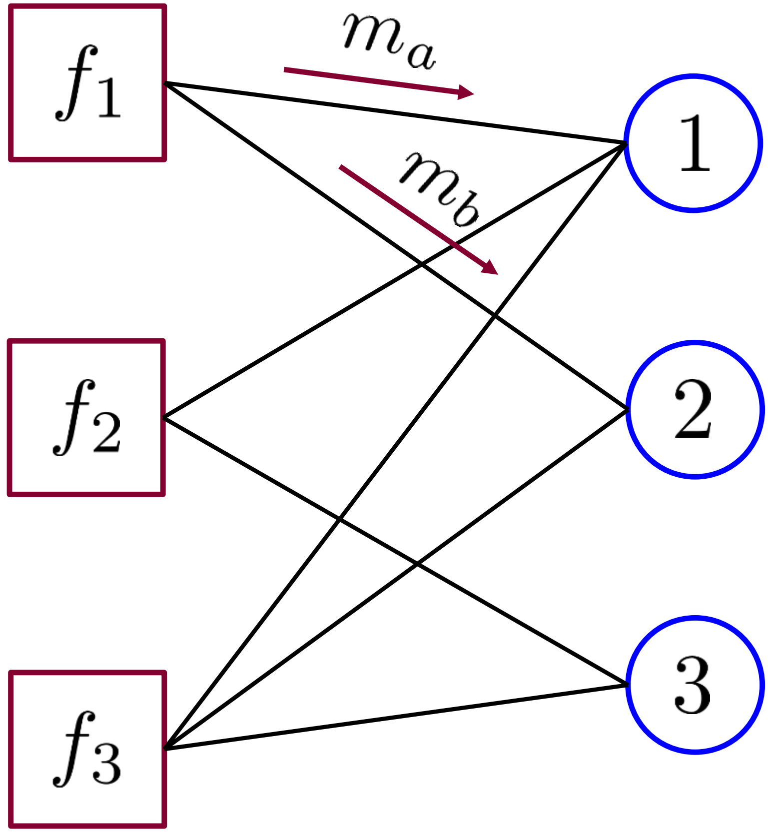
Step 2: Iterative message-passing scheme

$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$

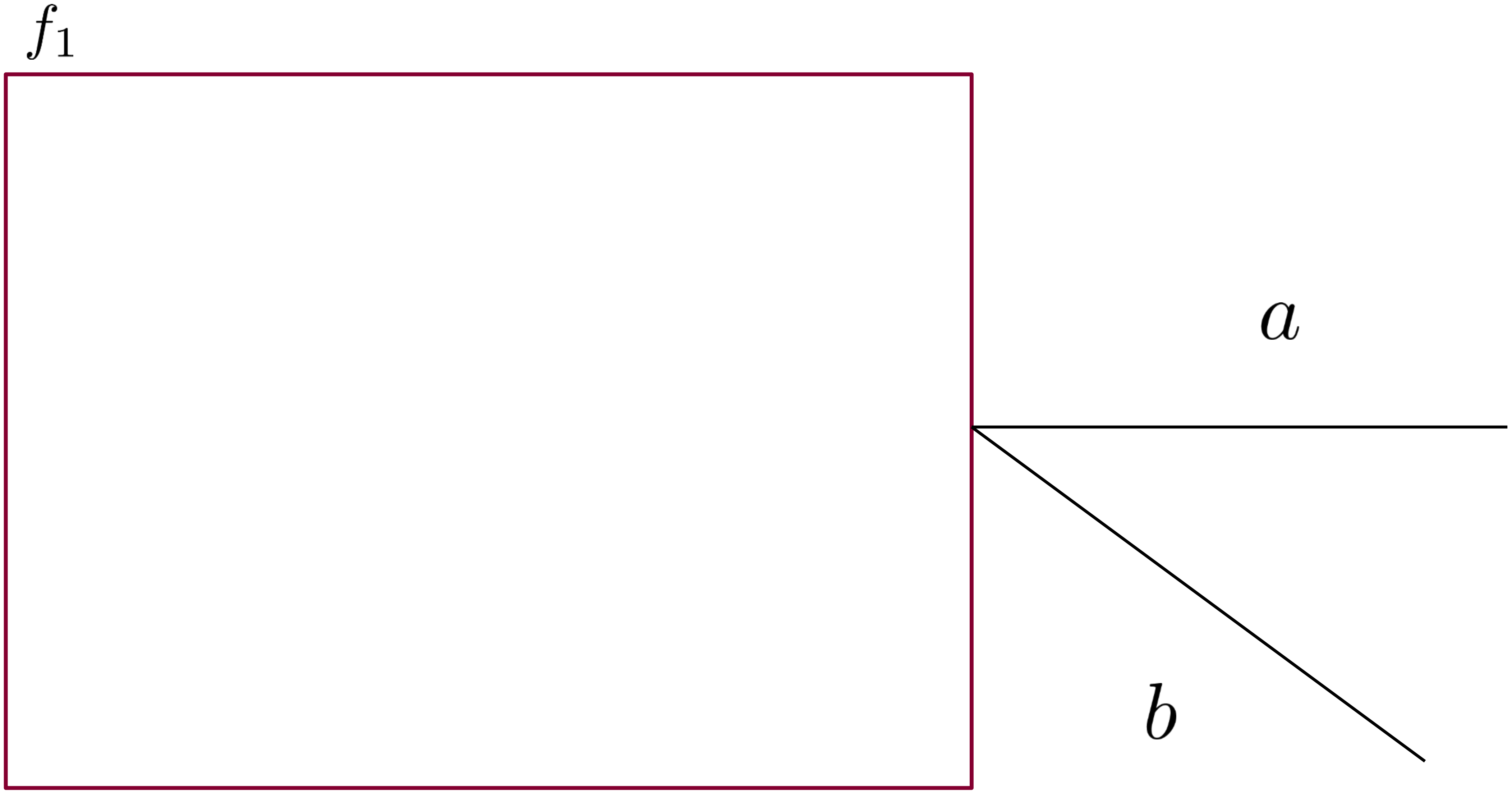


Step 2: Iterative message-passing scheme

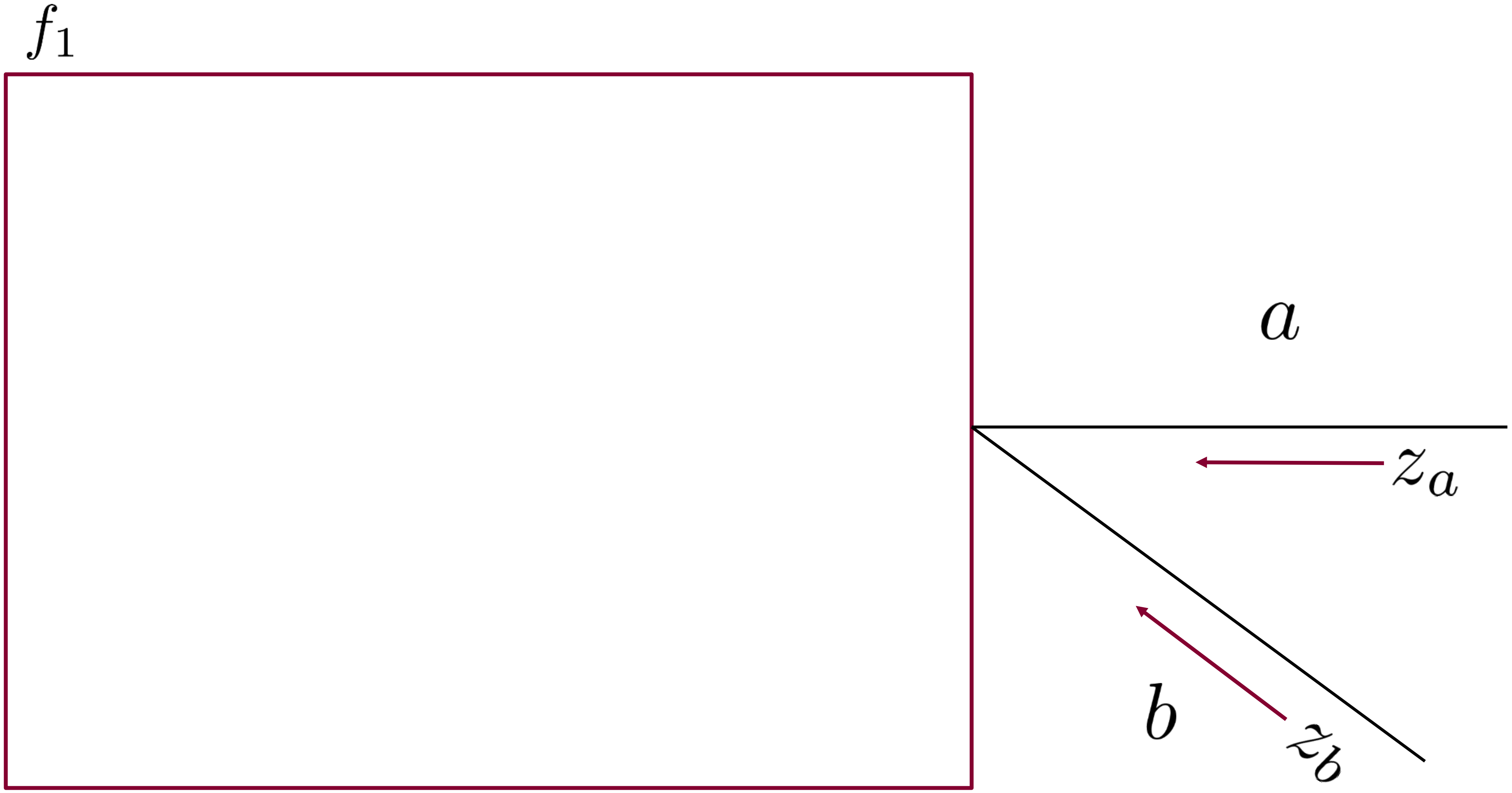
$$\underset{z_1, z_2, z_3 \in \mathbb{R}}{\text{minimize}} \; f_1(z_1, z_2) + f_2(z_1, z_3) + f_3(z_1, z_2, z_3)$$



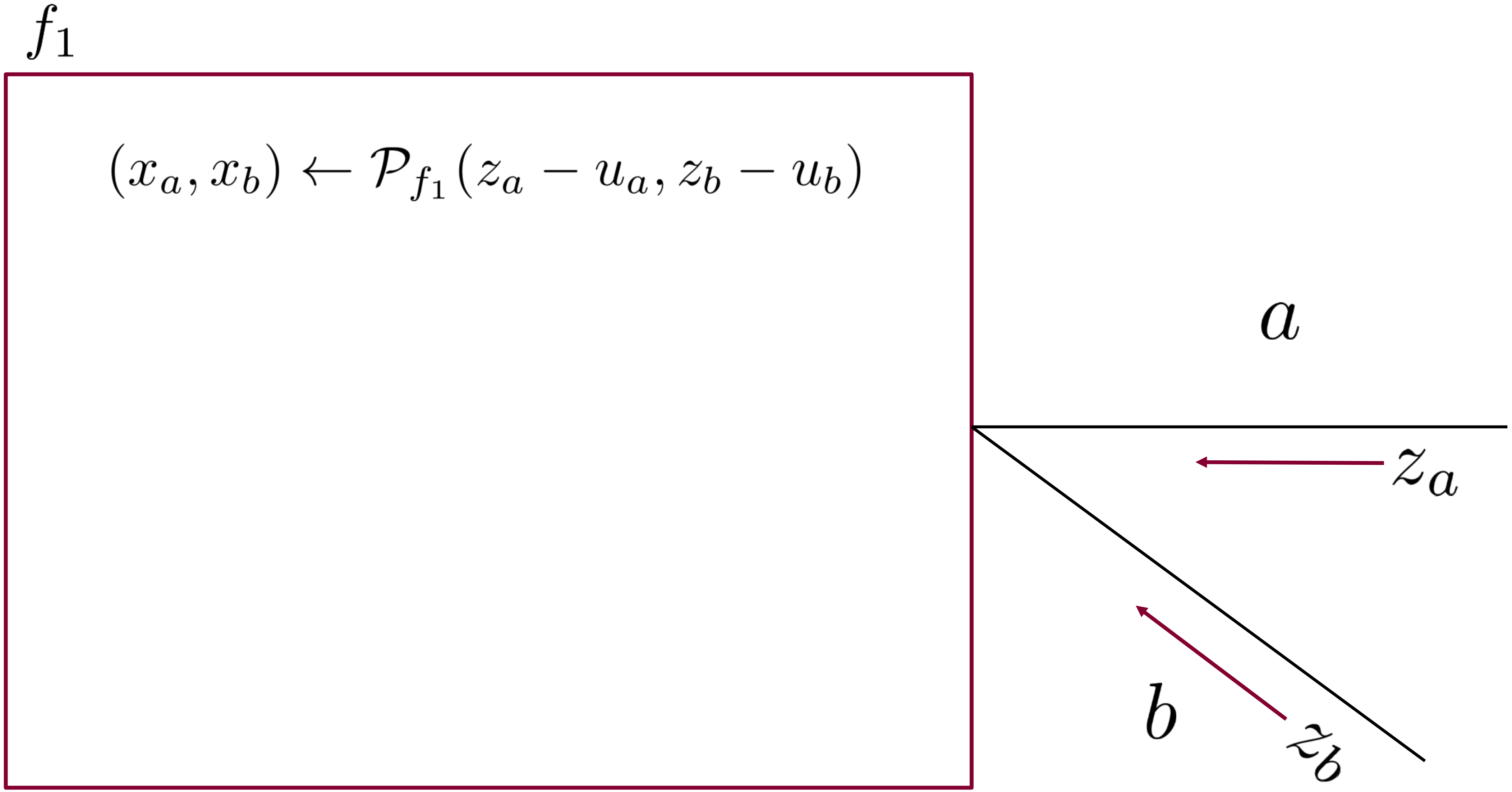
Iterative message-passing scheme



Iterative message-passing scheme



Iterative message-passing scheme

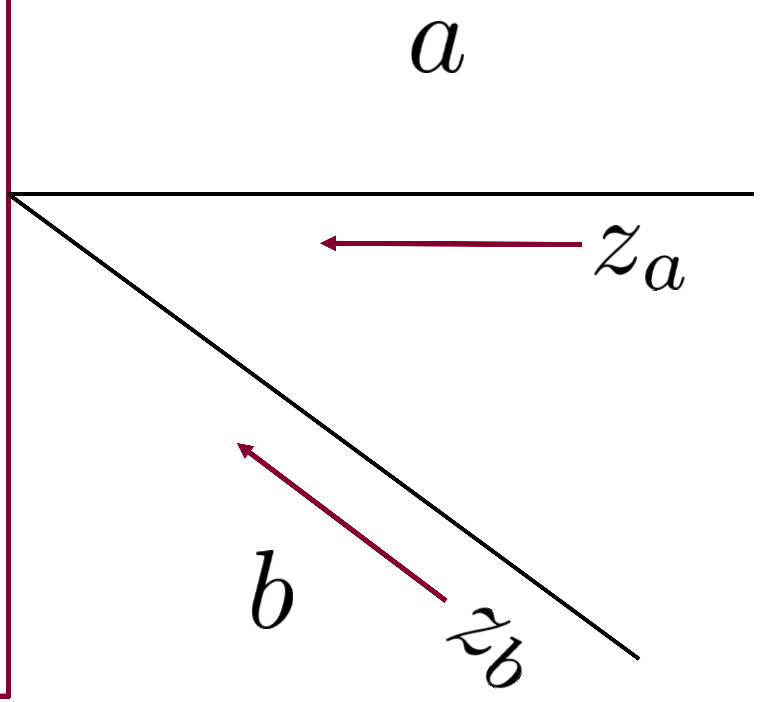


Iterative message-passing scheme

f_1

$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

$$(u_a, u_b) \leftarrow (u_a + x_a - z_a, u_b + x_b - z_b)$$



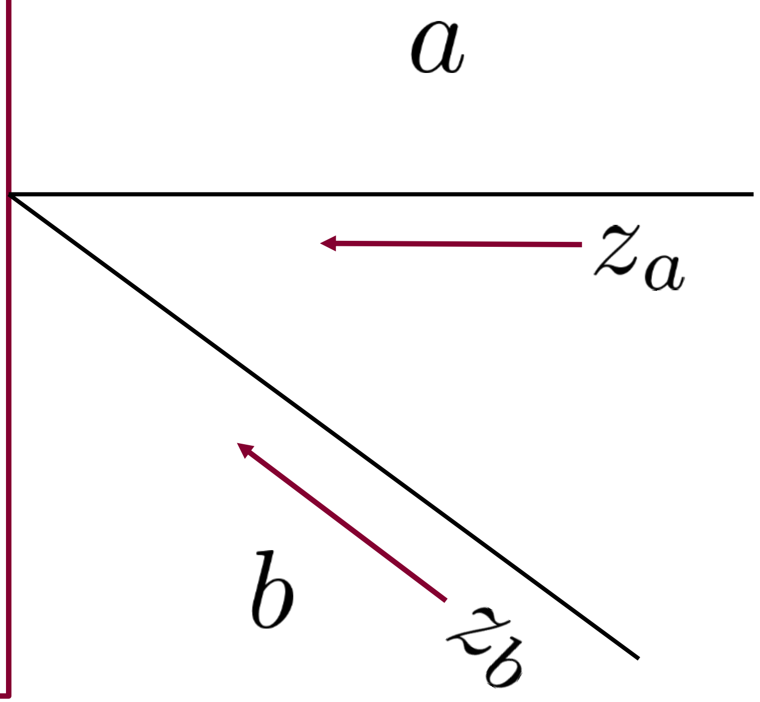
Iterative message-passing scheme

f_1

$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

$$(u_a, u_b) \leftarrow (u_a + x_a - z_a, u_b + x_b - z_b)$$

$$(m_a, m_b) \leftarrow (u_a + x_a, u_b + x_b)$$



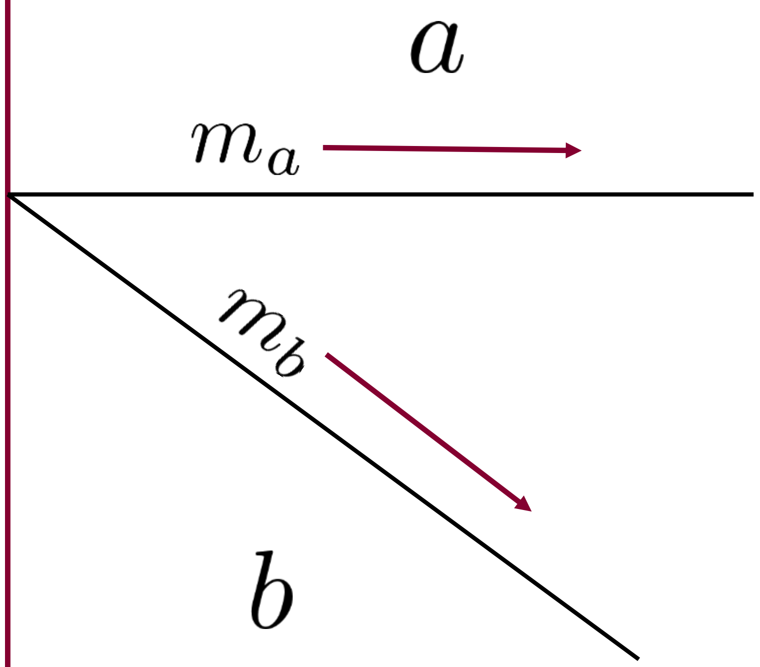
Iterative message-passing scheme

f_1

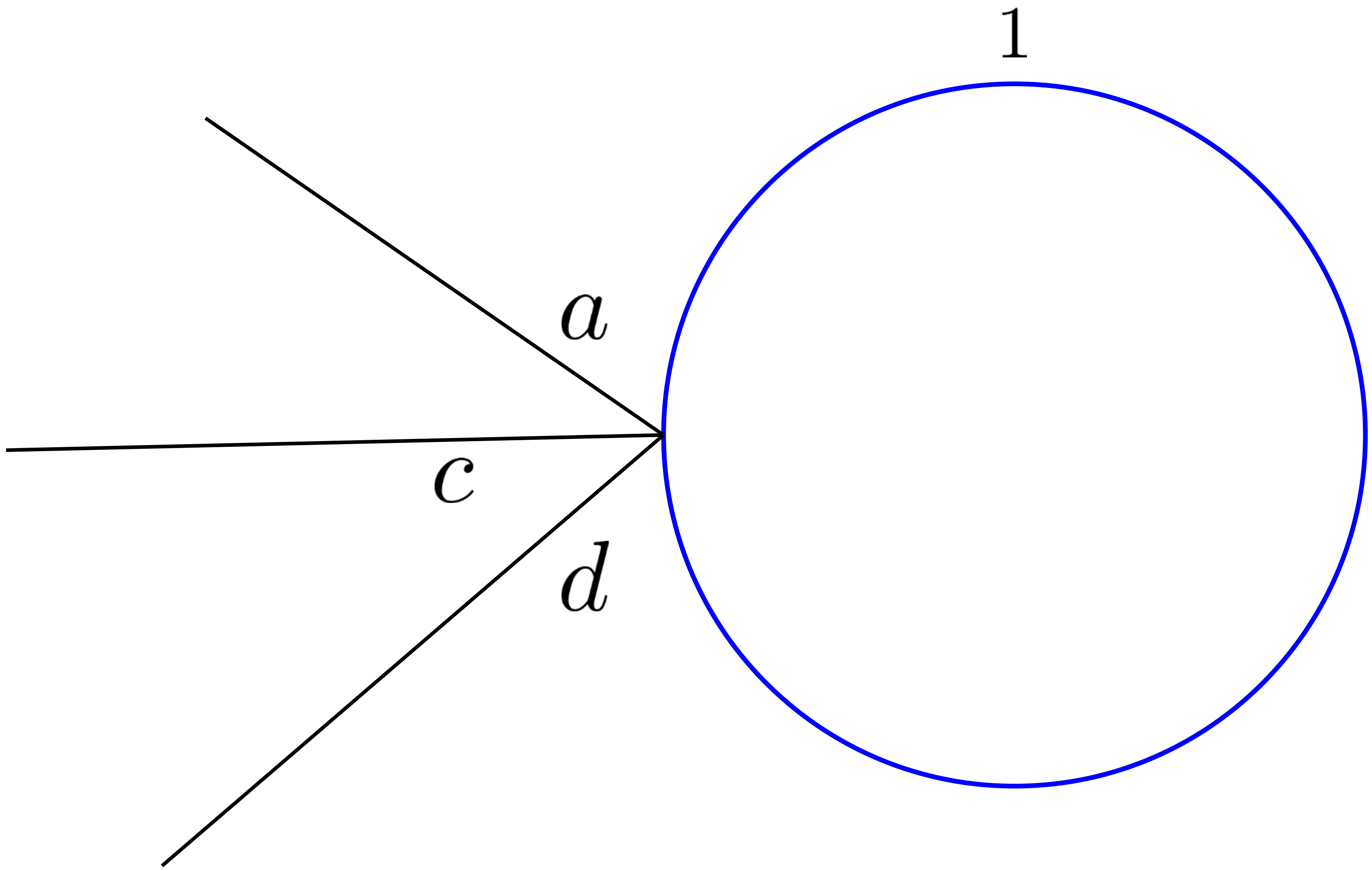
$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

$$(u_a, u_b) \leftarrow (u_a + x_a - z_a, u_b + x_b - z_b)$$

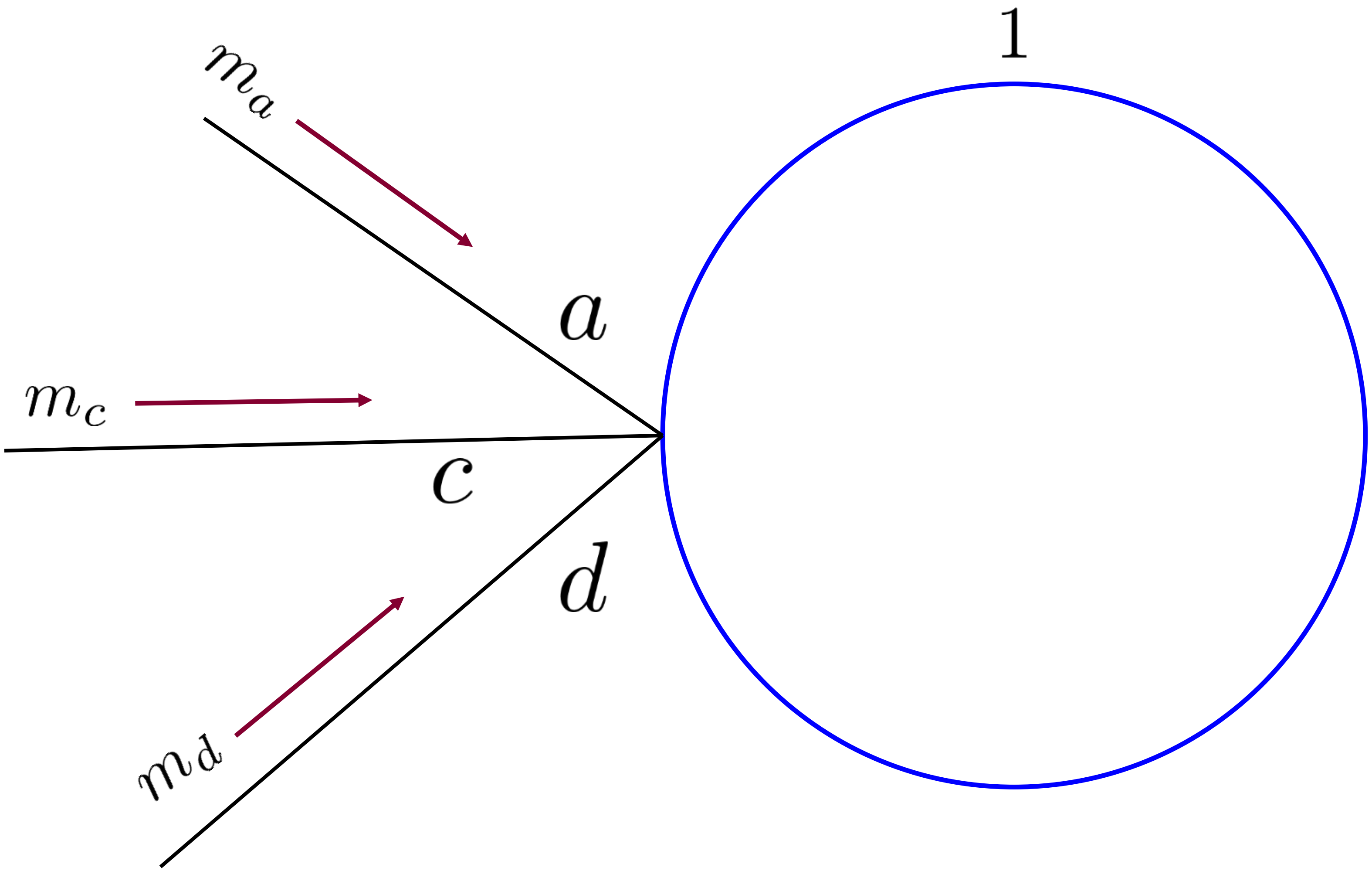
$$(m_a, m_b) \leftarrow (u_a + x_a, u_b + x_b)$$



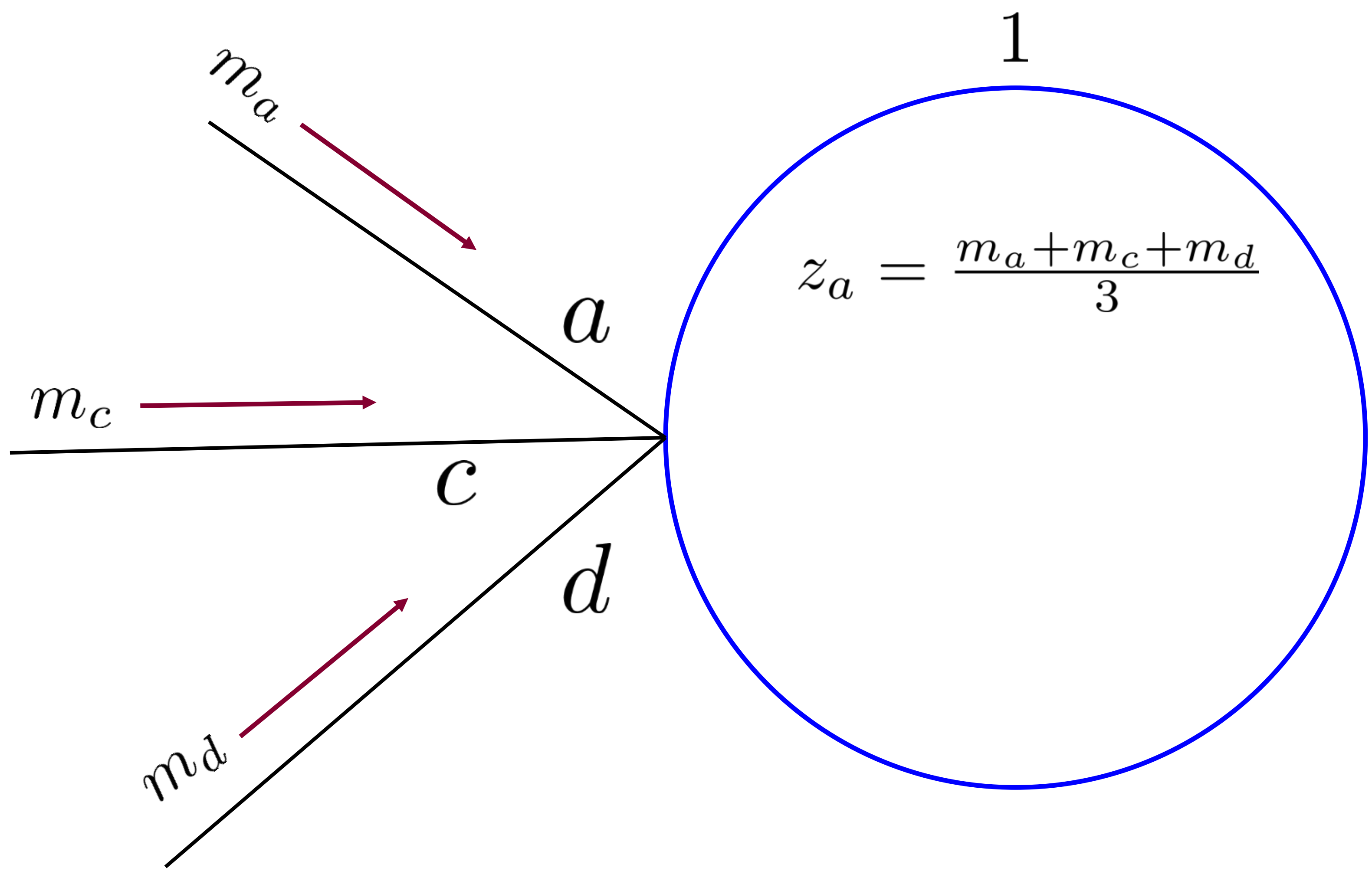
Iterative message-passing scheme



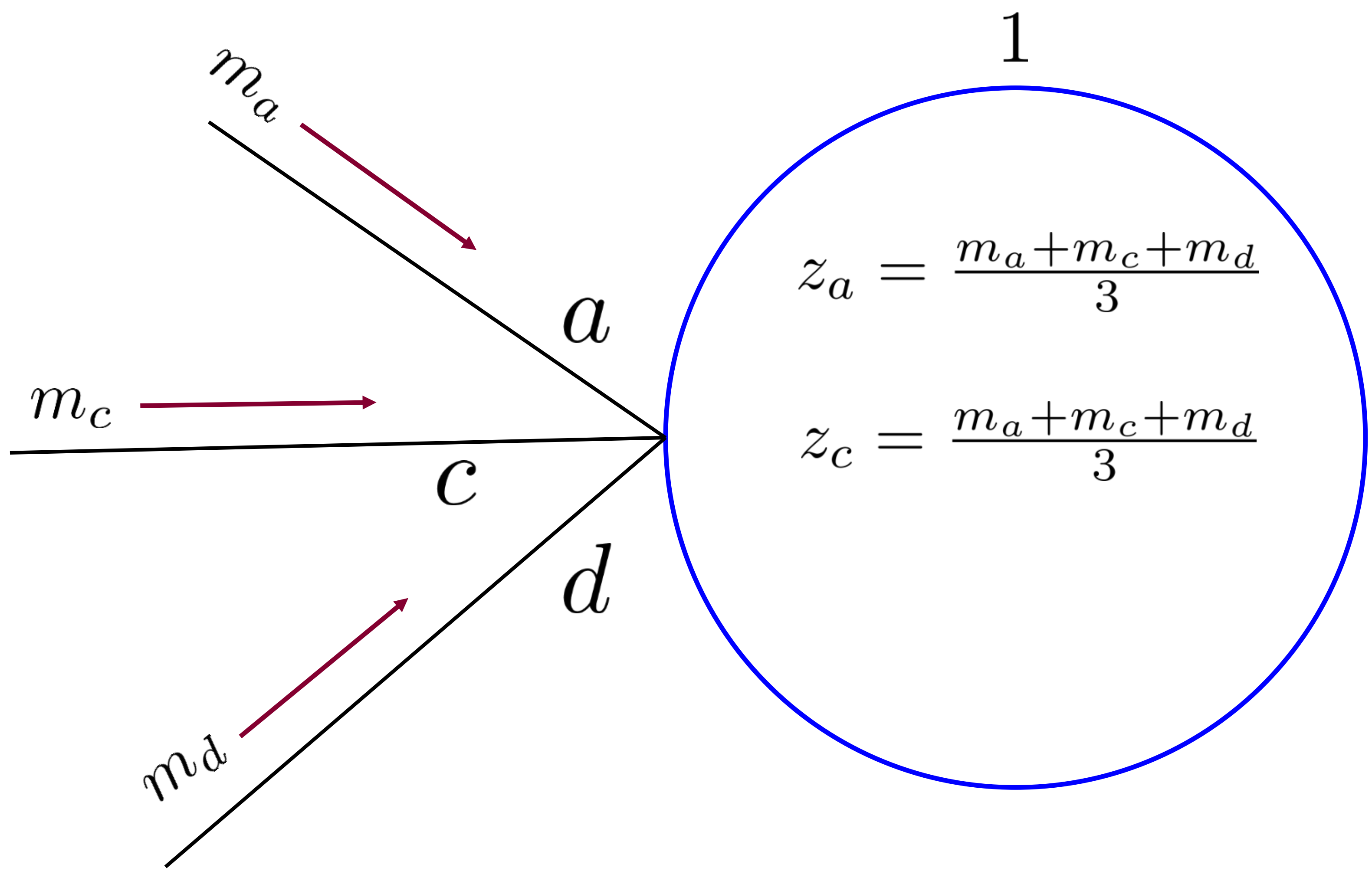
Iterative message-passing scheme



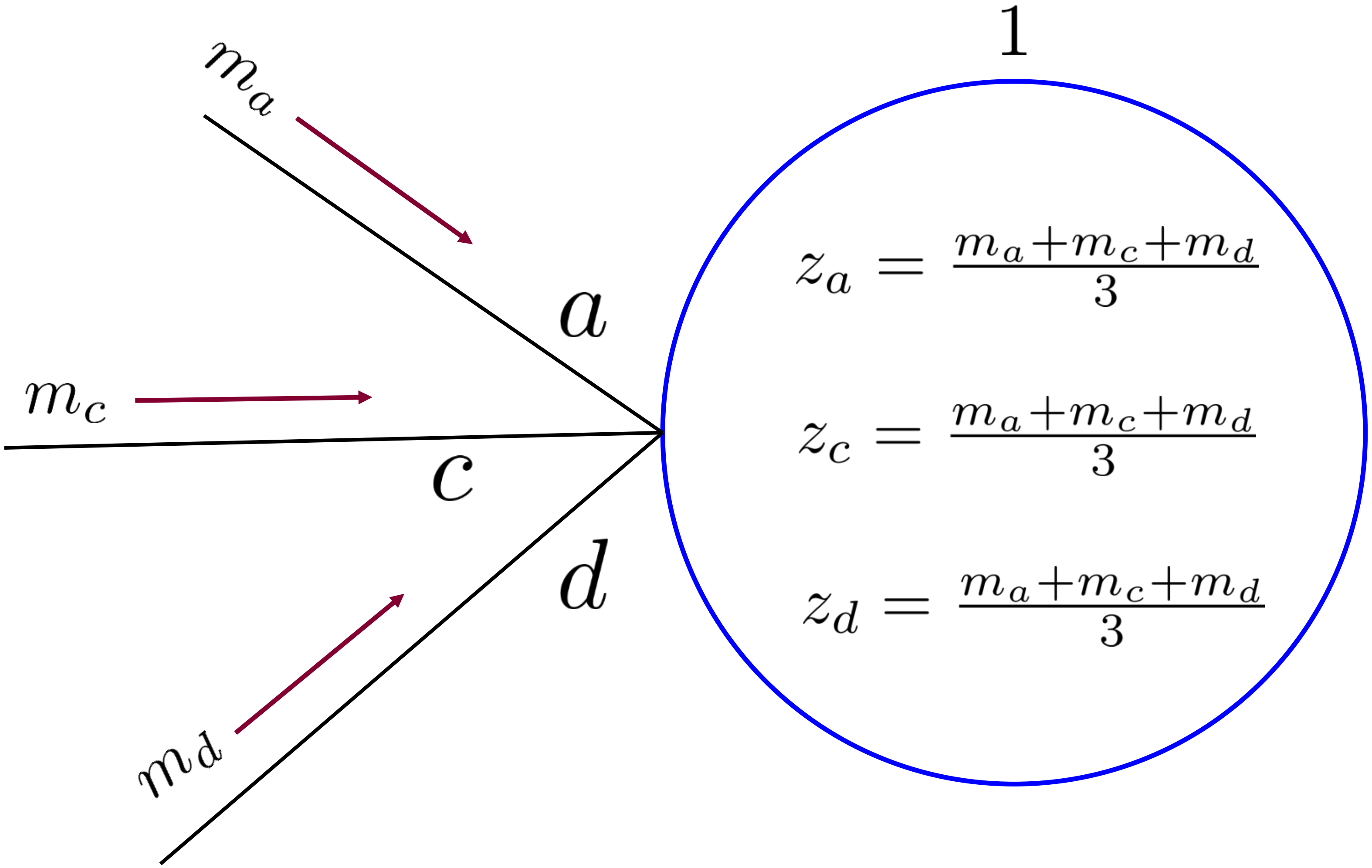
Iterative message-passing scheme



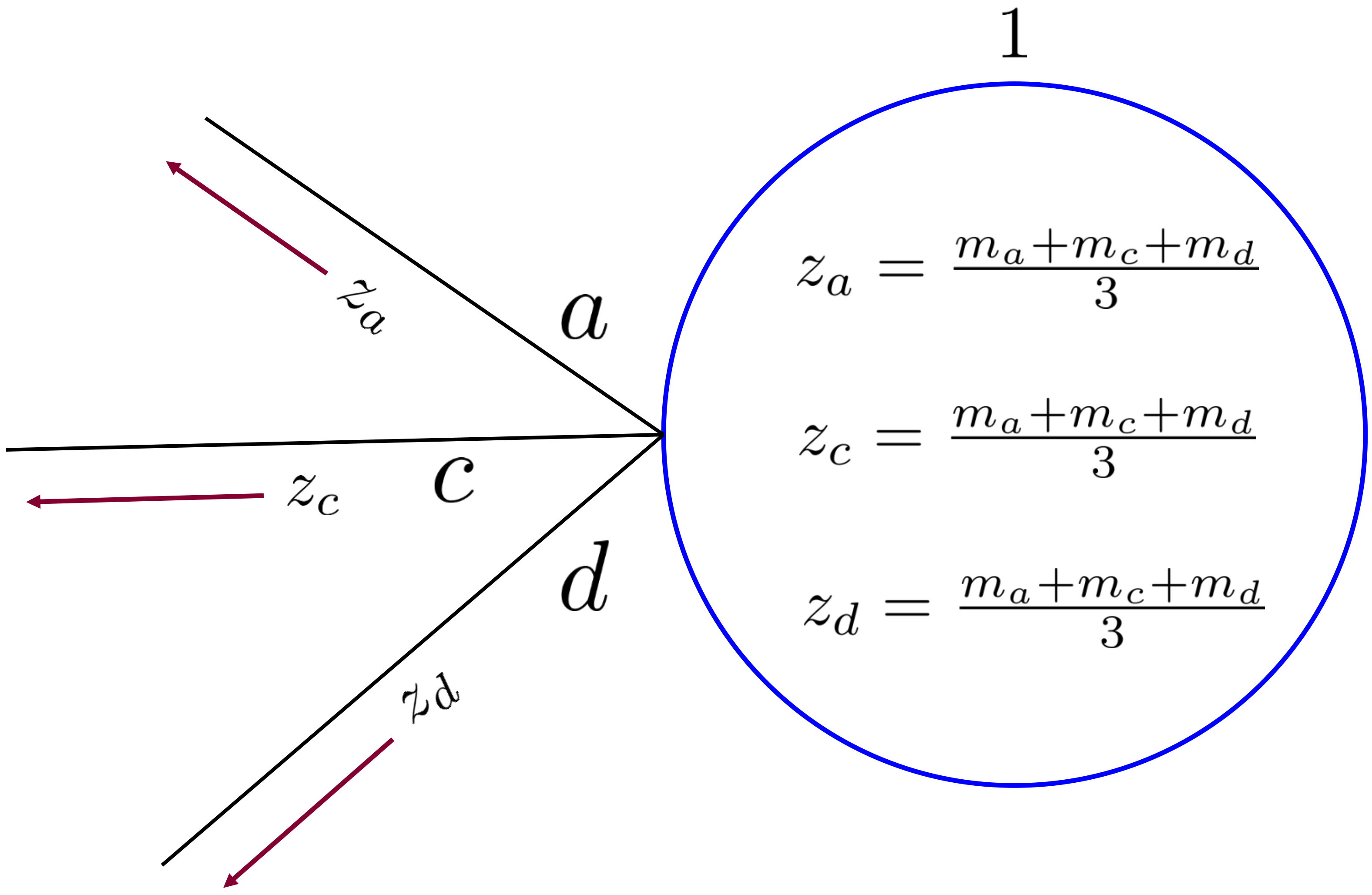
Iterative message-passing scheme



Iterative message-passing scheme



Iterative message-passing scheme



Computations

The "hard" part is to compute the following (all other computations are linear):

$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

Computations

The "hard" part is to compute the following (all other computations are linear):

$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

$$(x_a, x_b) \leftarrow \underset{s_a, s_b}{\arg\min}\ f_1(s_a, s_b) + \frac{\rho}{2}(s_a - z_a + u_a)^2 + \frac{\rho}{2}(s_b - z_b + u_b)^2$$

Computations

The "hard" part is to compute the following (all other computations are linear):

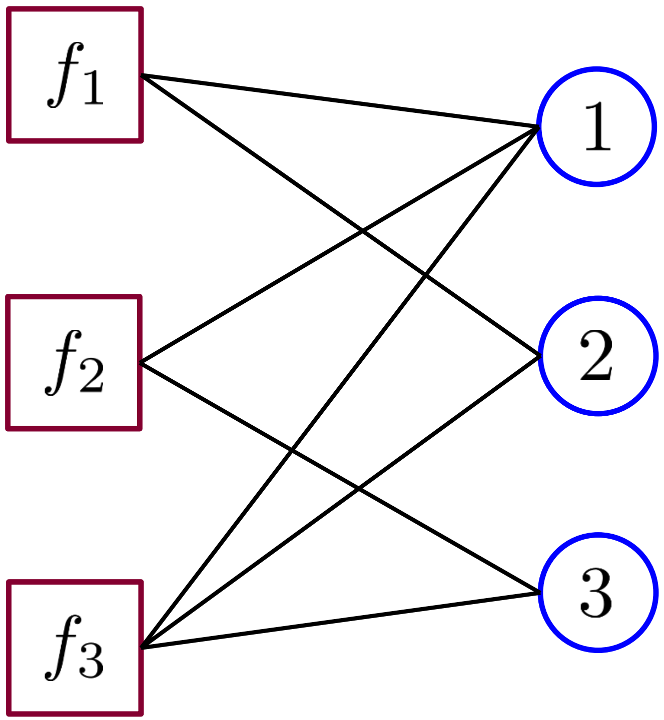
$$(x_a, x_b) \leftarrow \mathcal{P}_{f_1}(z_a - u_a, z_b - u_b)$$

$$(x_a, x_b) \leftarrow \underset{s_a, s_b}{\arg\min}\ f_1(s_a, s_b) + \frac{\rho}{2}(s_a - z_a + u_a)^2 + \frac{\rho}{2}(s_b - z_b + u_b)^2$$

$\mathcal{P}$ is called the "proximal map" or the "proximal function"

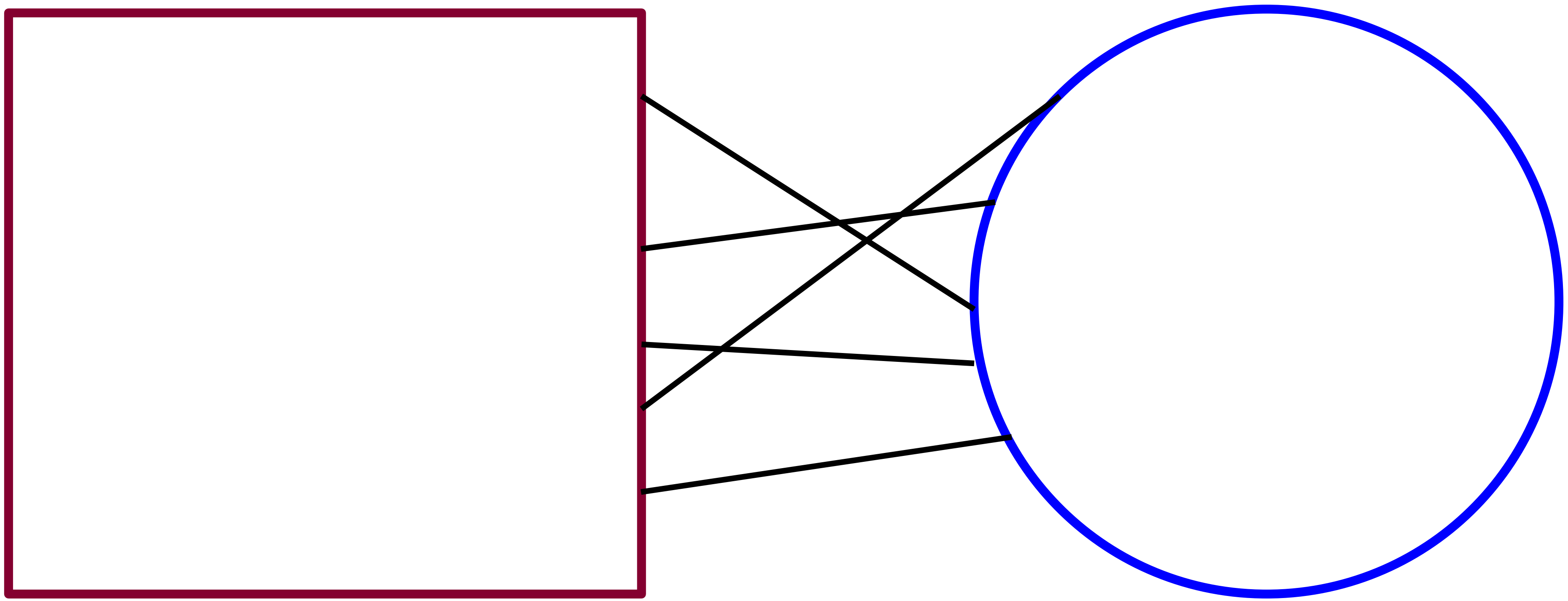
Step 3: Run until convergence

The updates in each side of the graph can be done in parallel

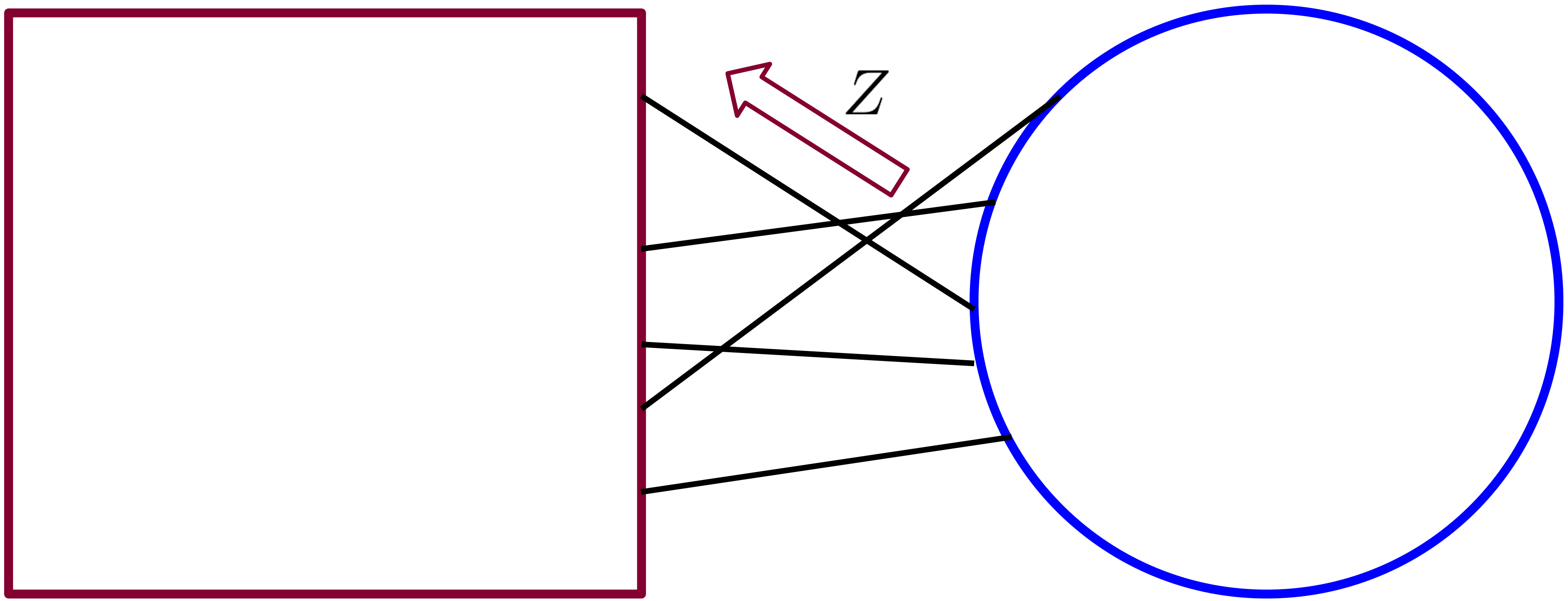


The final solution is read at variable nodes

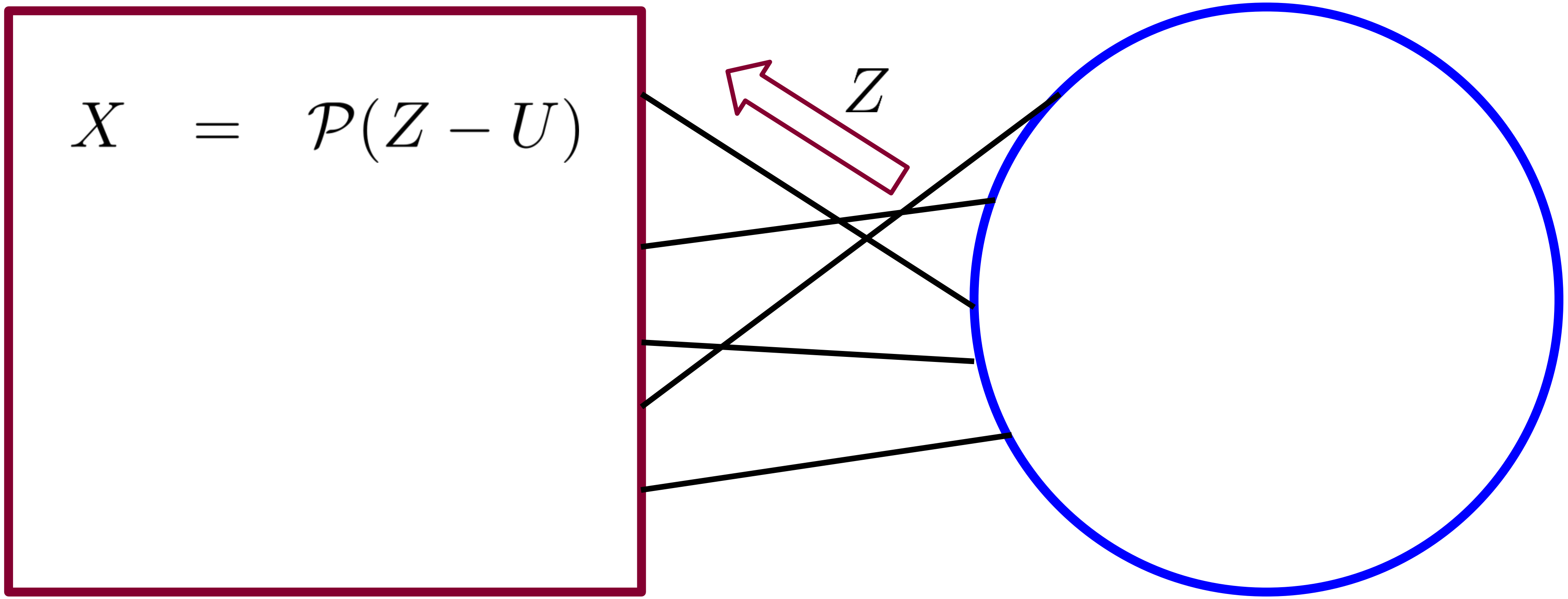
Compact representation



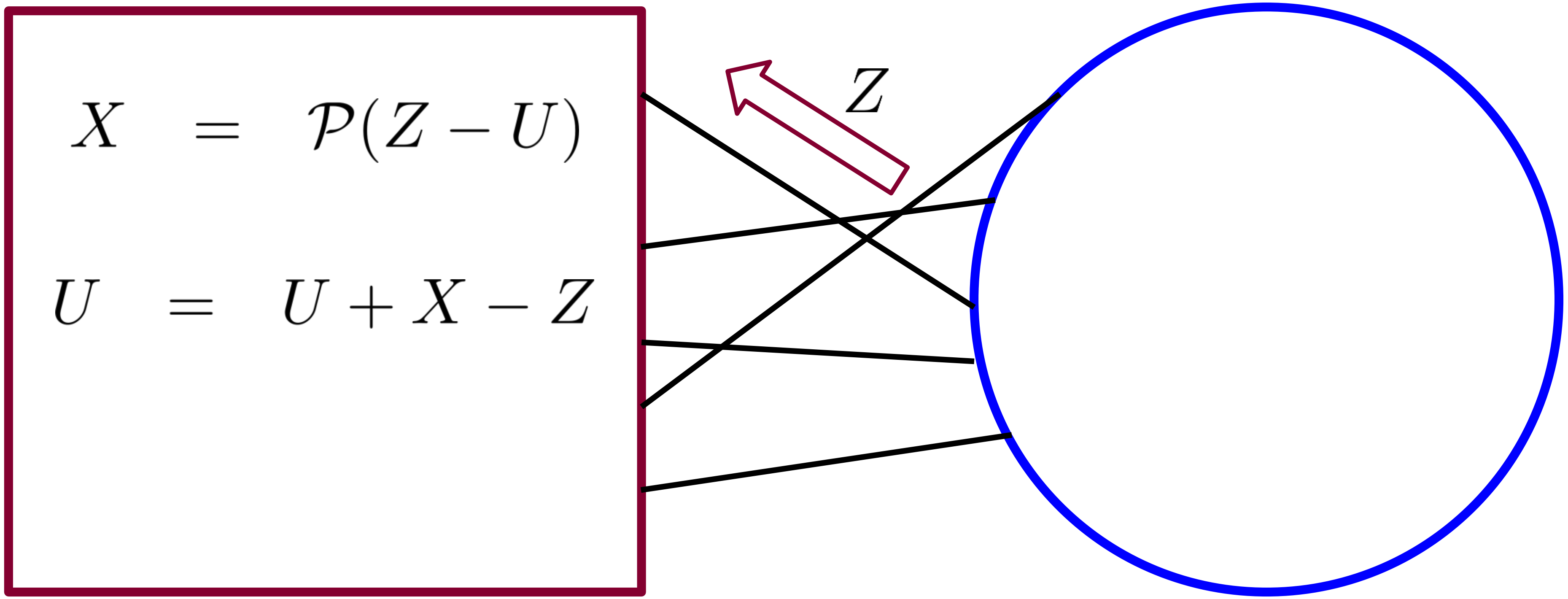
Compact representation



Compact representation

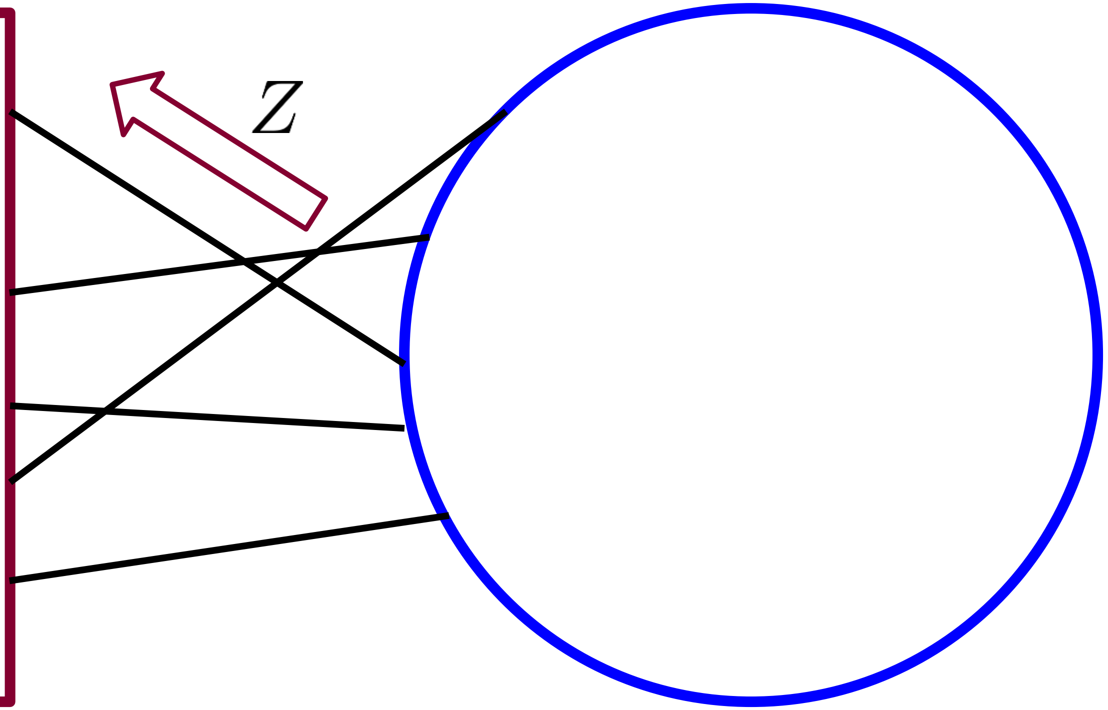


Compact representation



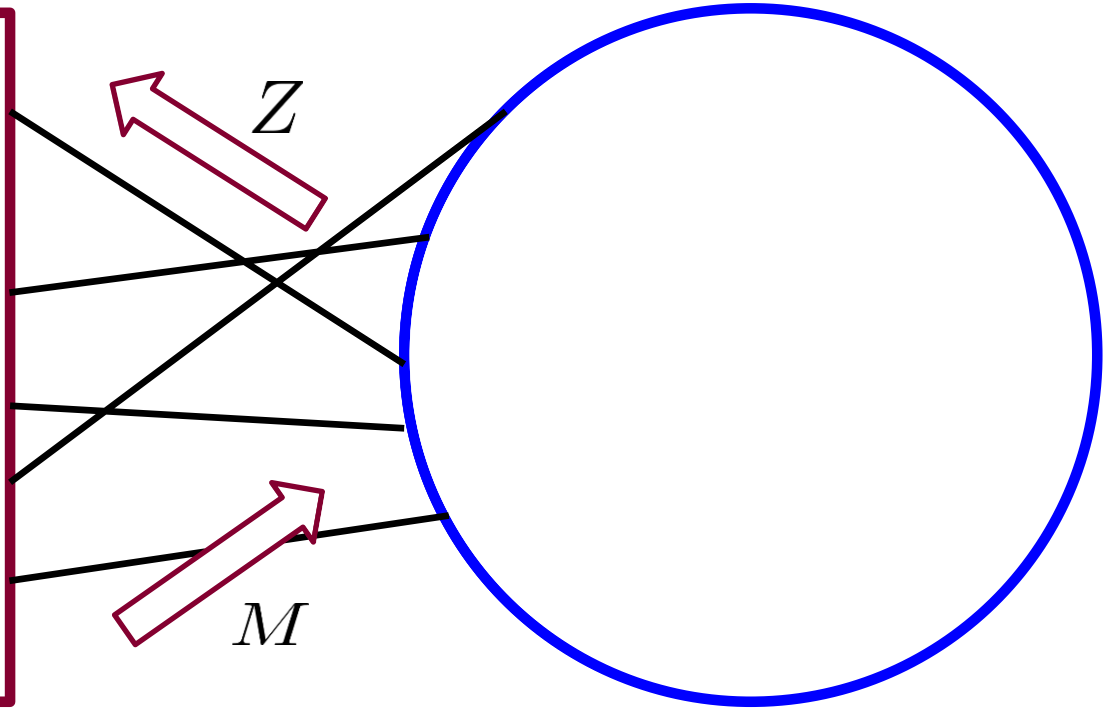
Compact representation

$$
\begin{aligned}
X &= \mathcal{P}(Z - U) \\
U &= U + X - Z \\
M &= X + U
\end{aligned}
$$

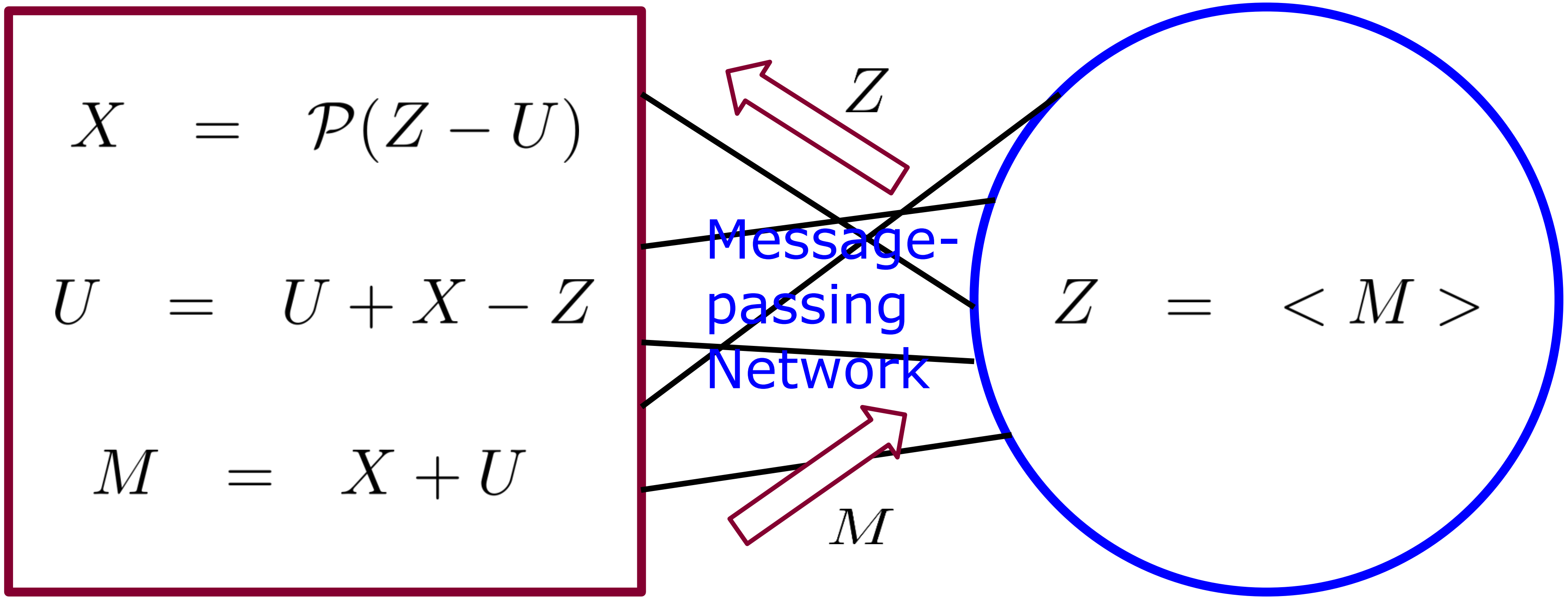


Compact representation

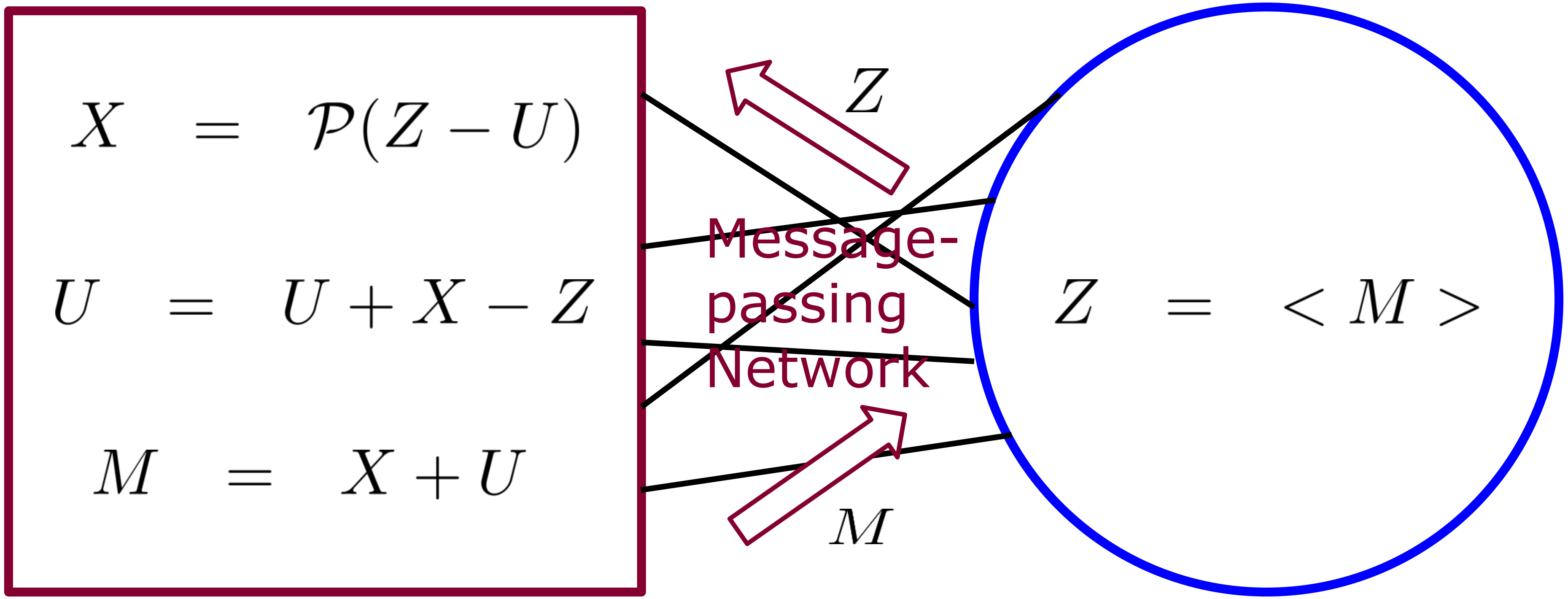
$$
\begin{aligned}
X &= \mathcal{P}(Z - U) \\
U &= U + X - Z \\
M &= X + U
\end{aligned}
$$



Compact representation

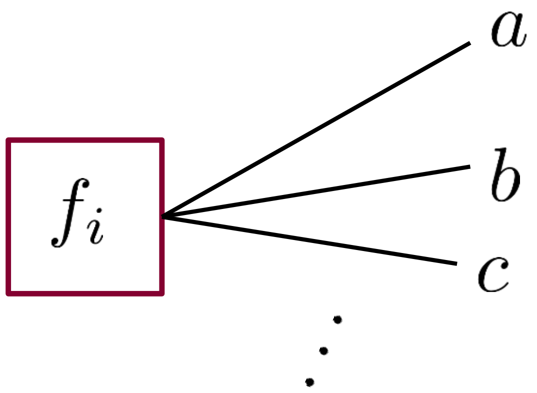


Compact representation



Compact representation

Define function $\mathcal{P}_{f_i}$ that for each f_i computes the following:



$$
\begin{aligned}
(x_a, x_b, \ldots) = \arg\min_{s_a, s_b, \ldots} f_i(s_a, s_b, \ldots) \quad &+ \quad \frac{\rho}{2}(s_a - (z_a - u_a))^2 \\
&+ \quad \frac{\rho}{2}(s_a - (z_a - u_a))^2 + \ldots
\end{aligned}
$$

$$
= \mathcal{P}_{f_i}(z_a - u_a, z_b - u_b, \ldots)
$$

Compact representation

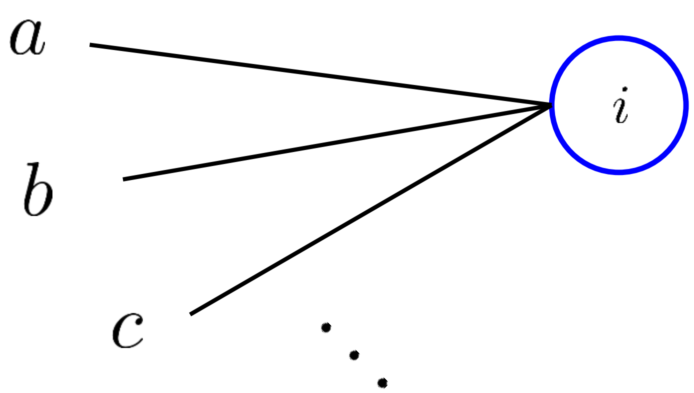
$U = U + X - Z$ does the following:

$$u_a = u_a + x_a - z_a, \;\; u_b = u_b + x_b - z_b, \ldots$$

$M \quad = \quad X + U$ does the following:

$$m_a = x_a + u_a, \;\; m_b = x_b + u_b, \ldots$$

$Z =< M >$ does the following:



$$z_a = \frac{1}{k_i}(m_a + m_b + \ldots)$$

# of edges

Benefits

- Computations are done in parallel over a distributed network
- Problem $\mathcal{P}_i$ is nice even when f_i is not
- ADMM is the fastest among all first-order methods*
- Converges under convexity*
- Empirically good even for non-convex problems**

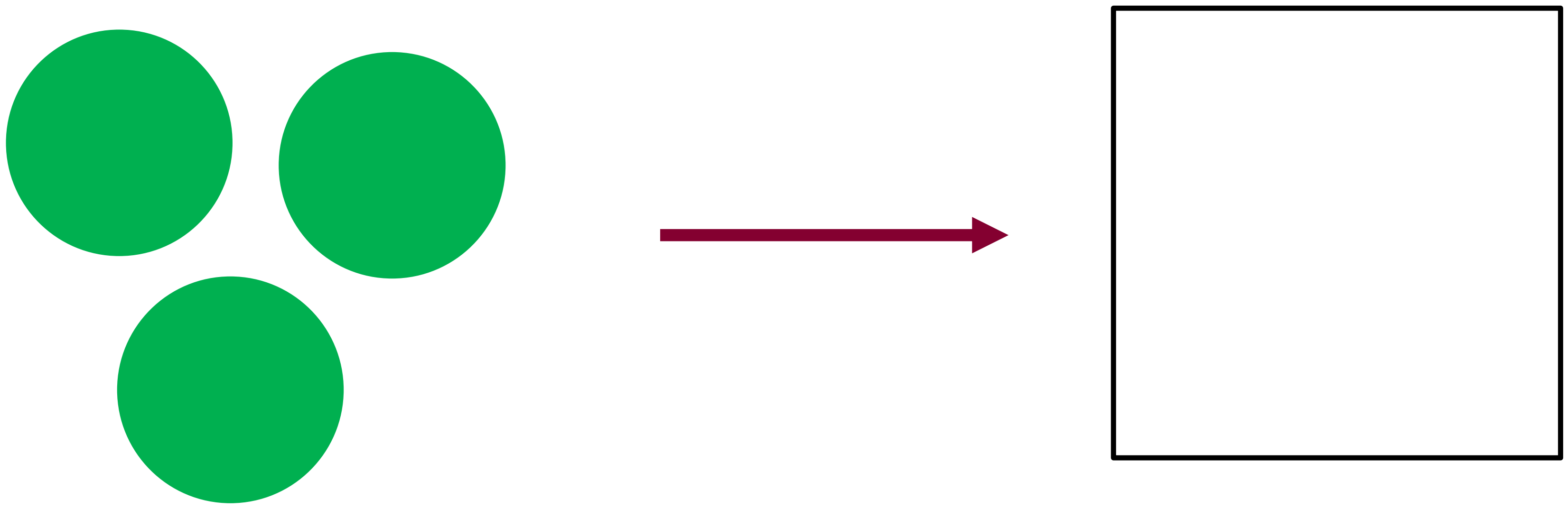
*França, Guilherme, and José Bento. *"An explicit rate bound for over-relaxed ADMM."* IEEE International Symposium on Information Theory (ISIT), 2016.
**Derbinsky, Nate, et al. "*An improved three-weight message-passing algorithm*." arXiv preprint arXiv:1305.1961 (2013).

Application examples

- Circle Packing
- Non-smooth Filtering
- Sudoku Puzzle
- Support Vector Machine

Circle Packing

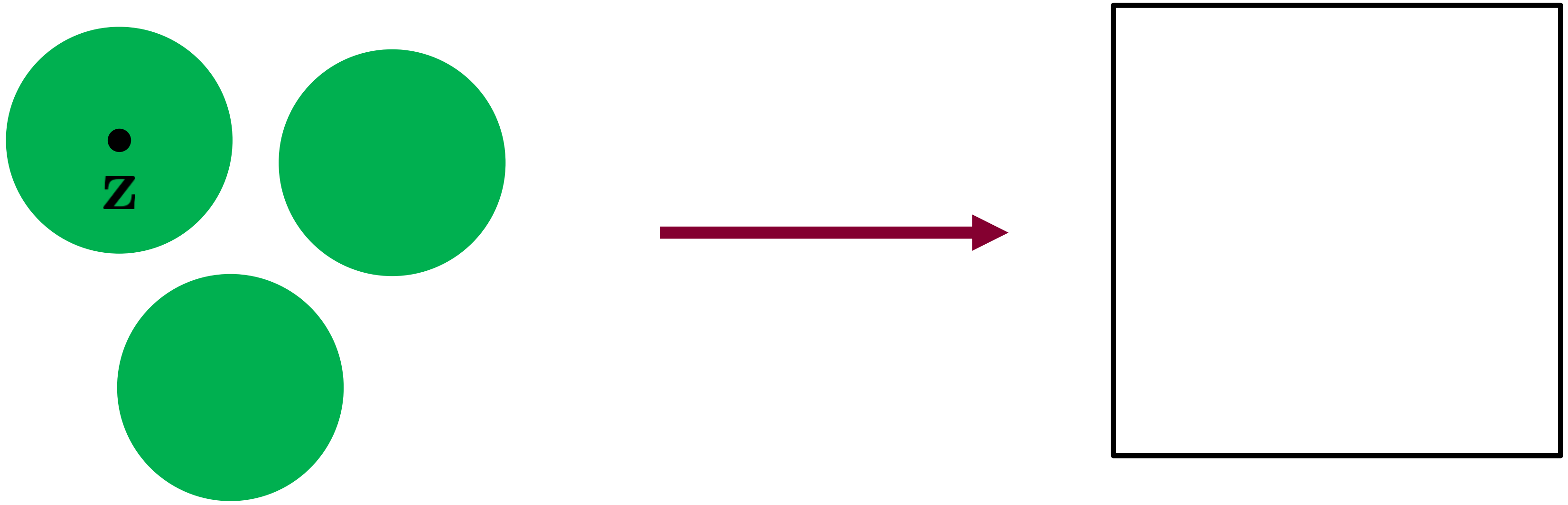
- Can we pack 3 circles of radius 0.253 in a box of size 1.0?
- Non-convex problem



Circle Packing

- Can we pack 3 circles of radius 0.253 in a box of size 1.0?
- Non-convex problem

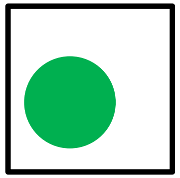
$$\mathbf{z} = (z_1, z_2)$$



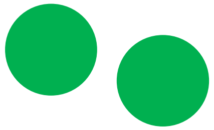
Circle Packing

- Can we pack 3 circles of radius 0.253 in a box of size 1.0?

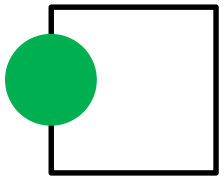
$$\min_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3} \ \mathrm{Box}(\mathbf{z}_1) + \mathrm{Box}(\mathbf{z}_2) + \mathrm{Box}(\mathbf{z}_3) + \mathrm{Coll}(\mathbf{z}_1,\mathbf{z}_2) + \mathrm{Coll}(\mathbf{z}_1,\mathbf{z}_3) + \mathrm{Coll}(\mathbf{z}_2,\mathbf{z}_3)$$



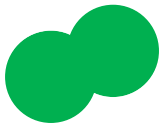
$$\mathrm{Box}(\mathbf{z}) = 0$$



$$\mathrm{Collision}(\mathbf{z},\mathbf{z}') = 0$$



$$\mathrm{Box}(\mathbf{z}) = \infty$$

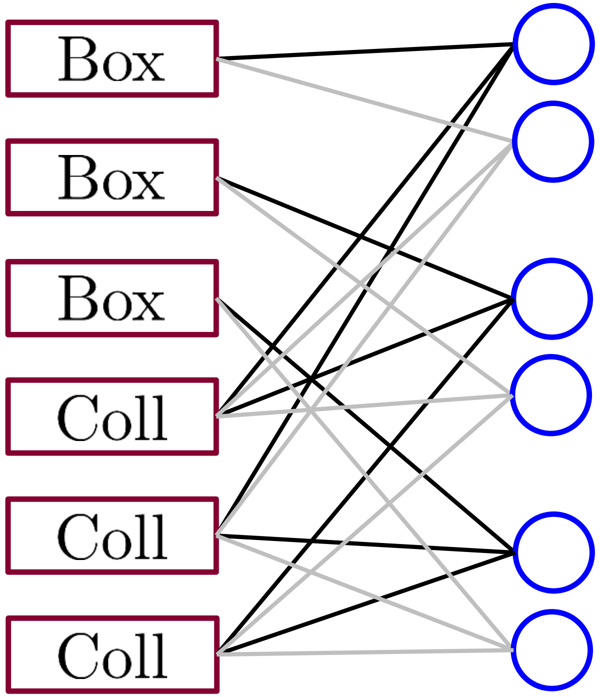


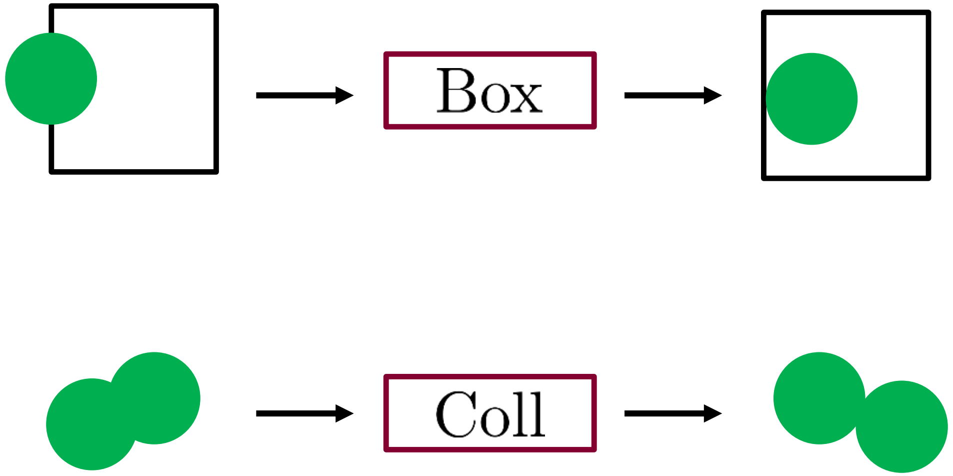
$$\mathrm{Collision}(\mathbf{z},\mathbf{z}') = \infty$$

Circle Packing

- Can we pack 3 circles of radius 0.253 in a box of size 1.0?

$$\min_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3} \ \mathrm{Box}(\mathbf{z}_1) + \mathrm{Box}(\mathbf{z}_2) + \mathrm{Box}(\mathbf{z}_3) + \mathrm{Coll}(\mathbf{z}_1,\mathbf{z}_2) + \mathrm{Coll}(\mathbf{z}_1,\mathbf{z}_3) + \mathrm{Coll}(\mathbf{z}_2,\mathbf{z}_3)$$

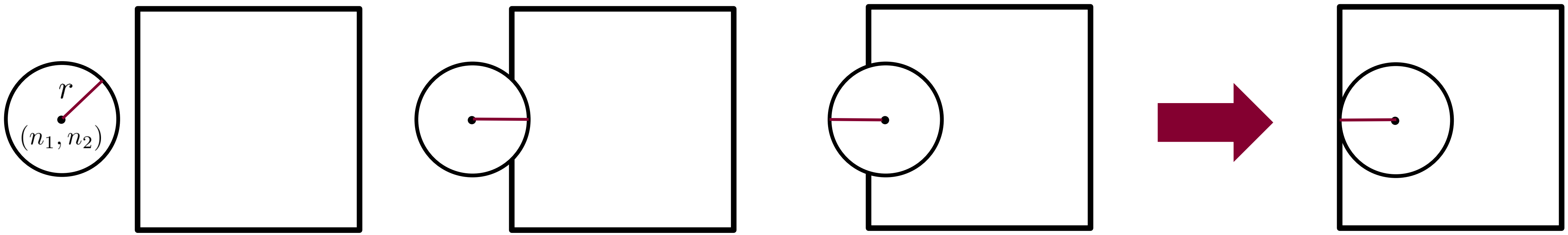




Circle Packing - Box

$$(x_1, x_2) = \underset{s_1, s_2, \ldots}{\arg\min}\ f(s_1, s_2) + \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2$$

$$f(s_1, s_2) = \begin{cases} 0 & \text{if } (s_1, s_2) \in \text{ box} \\ \infty & \text{if } (s_1, s_2) \notin \text{box} \end{cases}$$



$$x_i = \min(1 - r, \max(r, n_i)) \qquad i = 1, 2$$

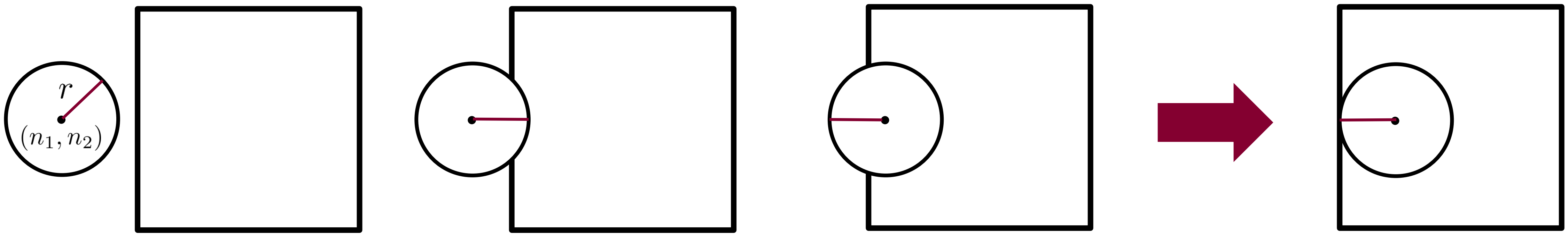
Circle Packing - Box

$z_1 - u_1$

$$(x_1, x_2) = \underset{s_1, s_2, \ldots}{\arg\min}\; f(s_1, s_2) + \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2$$

$z_2 - u_2$

$$f(s_1, s_2) = \begin{cases} 0 & \text{if } (s_1, s_2) \in \text{ box} \\ \infty & \text{if } (s_1, s_2) \notin \text{box} \end{cases}$$

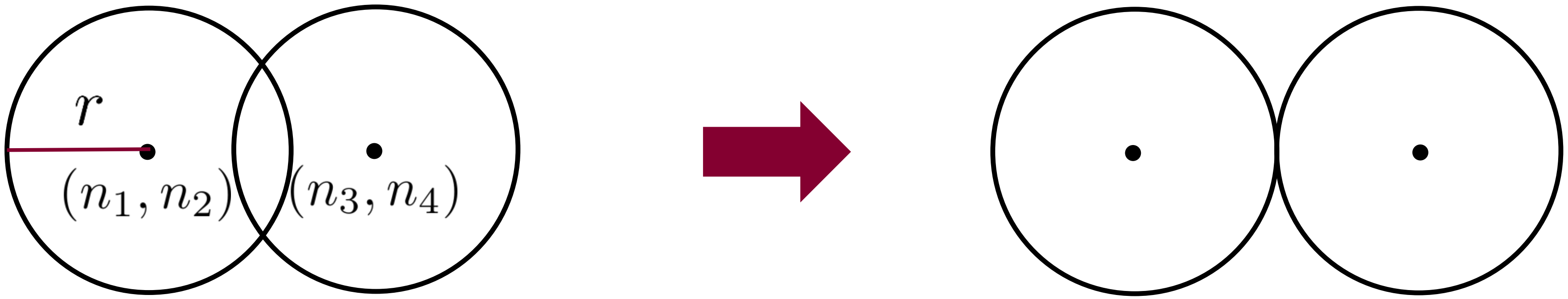


$$x_i = \min(1 - r, \max(r, n_i)) \qquad i = 1, 2$$

Circle Packing - Collision

$$(x_1, x_2, x_3, x_4) = \underset{s_1,s_2,s_3,s_4}{\arg\min}\ f(s_1, s_2, s_3, s_4) \quad + \quad \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2$$
$$+ \quad \frac{\rho}{2}(s_3 - n_3)^2 + \frac{\rho}{2}(s_4 - n_4)^2$$

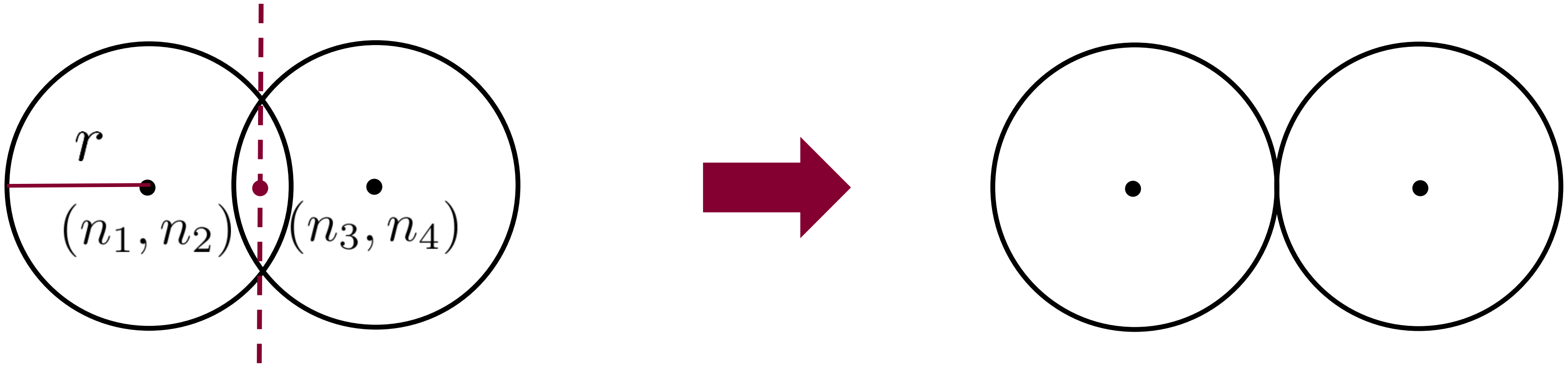
$$f(\underbrace{s_1, s_2}_{\mathbf{s}_1}, \underbrace{s_3, s_4}_{\mathbf{s}_2}) = \begin{cases} 0 & \text{if } d(\mathbf{s}_1, \mathbf{s}_2) \geq 2r \\ \infty & \text{if } d(\mathbf{s}_1, \mathbf{s}_2) < 2r \end{cases}$$



Circle Packing - Collision

$$x_1 = \frac{n_1 + n_3}{2} - r\frac{n_3 - n_1}{\|n_3 - n_1\|}, \qquad x_2 = \frac{n_2 + n_4}{2} - r\frac{n_2 - n_4}{\|n_2 - n_4\|}$$

$$x_3 = \frac{n_1 + n_3}{2} + r\frac{n_3 - n_1}{\|n_3 - n_1\|}, \qquad x_4 = \frac{n_2 + n_4}{2} + r\frac{n_2 - n_4}{\|n_2 - n_4\|}$$

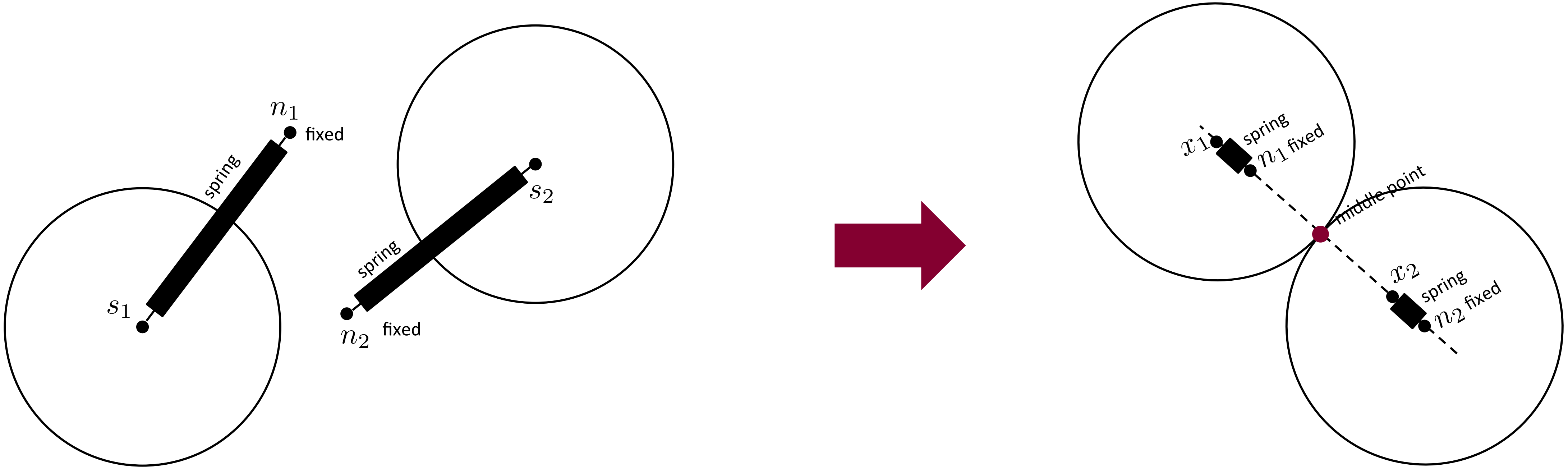


Circle Packing - Collision

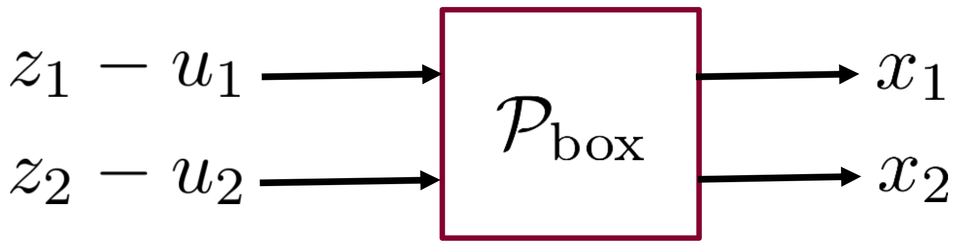
$$(\mathbf{x}_1, \mathbf{x}_2) = \underset{\mathbf{s}_1, \mathbf{s}_2}{\arg\min} \; \|\mathbf{s}_1 - \mathbf{n}_1\|^2 + \|\mathbf{s}_2 - \mathbf{n}_2\|^2$$

$$\text{subject to } \|\mathbf{s}_1 - \mathbf{s}_2\| > 2r$$

Mechanical analogy: minimize the energy of a system of balls and springs

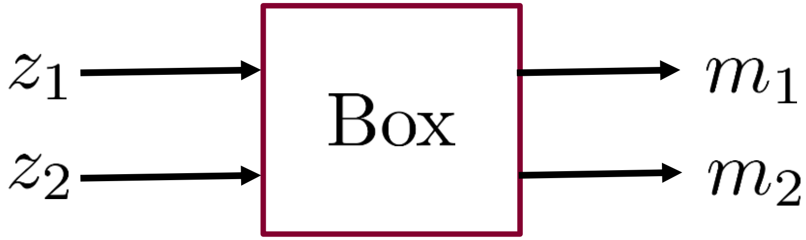


Circle Packing - Box



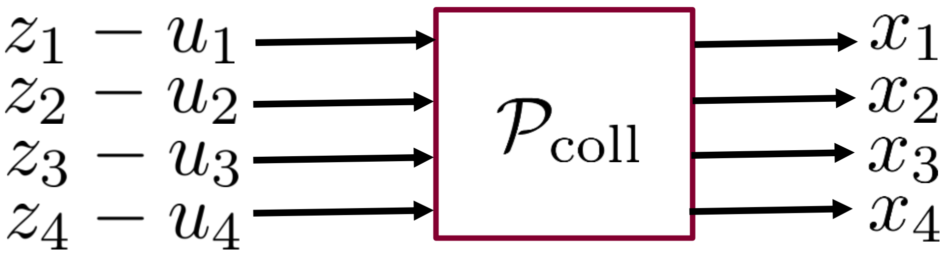

```
function [x_1 , x_2] = P_box(z_minus_u_1, z_minus_u_2)

    global r;

    x_1 = min([1-r, max([r, z_minus_u_1])]);
    x_2 = min([1-r, max([r, z_minus_u_2])]);

end
```

Circle Packing - Box



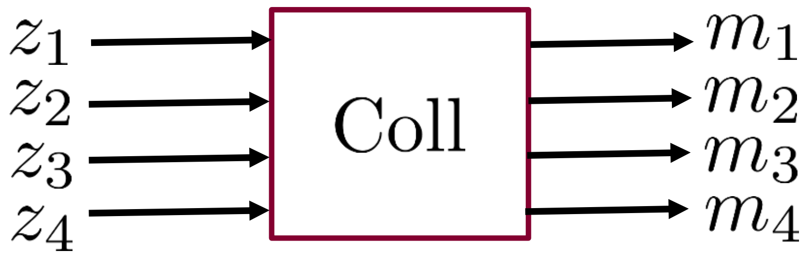

```
function [m_1, m_2, new_u_1, new_u_2] = F_box(z_1, z_2, u_1, u_2)

    % compute internal updates
    [x_1 , x_2] = P_box(z_1 - u_1, z_2 - u_2);

    new_u_1 = u_1 - (z_1 - x_1);
    new_u_2 = u_2 - (z_2 - x_2);

    % compute outgoing messages
    m_1 = new_u_1 + x_1;
    m_2 = new_u_2 + x_2;

end
```

Circle Packing - Collision



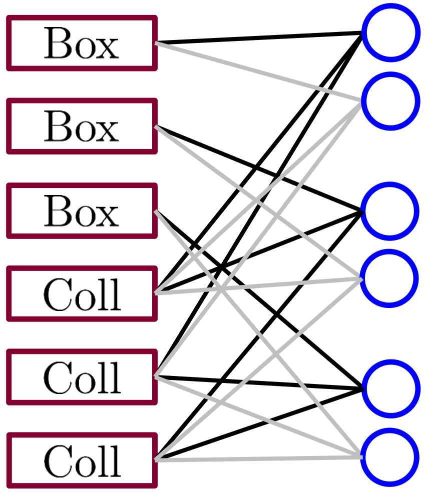

```
function [x_1, x_2, x_3, x_4] = P_coll(z_minus_u_1,z_minus_u_2,z_minus_u_3, z_minus_u_4)
    global r;
    d = sqrt((z_minus_u_1 - z_minus_u_3)^2  + (z_minus_u_2 - z_minus_u_4)^2);
    if (d > 2*r)
        x_1 = z_minus_u_1; x_2 = z_minus_u_2;
        x_3 = z_minus_u_3; x_4 = z_minus_u_4;
        return;
    end
    x_1 = 0.5*(z_minus_u_1 + z_minus_u_3)  +  r*(z_minus_u_1 - z_minus_u_3)/d;
    x_2 = 0.5*(z_minus_u_2 + z_minus_u_4)  +  r*(z_minus_u_2 - z_minus_u_4)/d;
    x_3 = 0.5*(z_minus_u_1 + z_minus_u_3)  -  r*(z_minus_u_1 - z_minus_u_3)/d;
    x_4 = 0.5*(z_minus_u_2 + z_minus_u_4)  -  r*(z_minus_u_2 - z_minus_u_4)/d;
end
```

Circle Packing - Collision



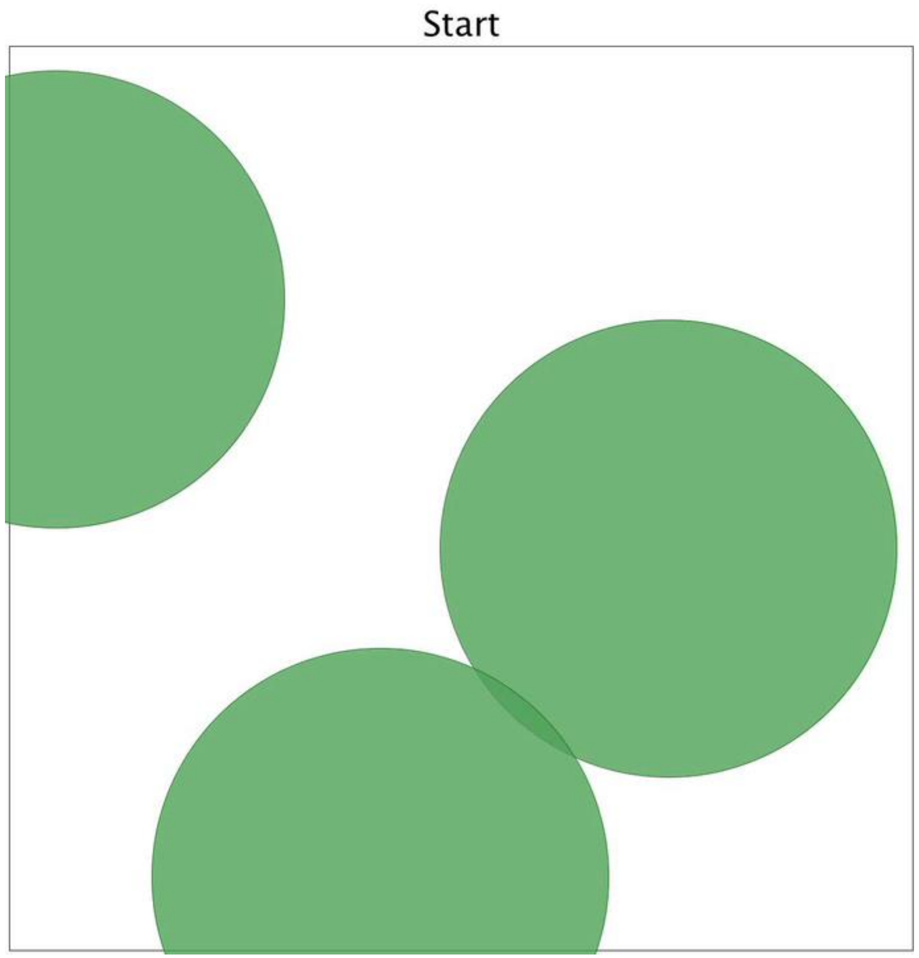

```
function [m_1,m_2,m_3,m_4,new_u_1,new_u_2,new_u_3,new_u_4] =
        F_coll(z_1, z_2, z_3, z_4, u_1, u_2, u_3, u_4)

    % Compute internal updates
    [x_1, x_2, x_3, x_4] = P_coll(z_1-u_1,z_2-u_2,z_3-u_3,z_4-u_4);

    new_u_1 = u_1-(z_1-x_1); new_u_2 = u_2-(z_2-x_2);
    new_u_3 = u_3-(z_3-x_3); new_u_4 = u_4-(z_4-x_4);

    % Compute outgoing messages
    m_1 = new_u_1 + x_1; m_2 = new_u_2 + x_2;
    m_3 = new_u_3 + x_3; m_4 = new_u_4 + x_4;
end
```

```
% Initialization
rho = 1; num_balls = 10; global r; r = 0.15; u_box = randn(num_balls,2); u_coll = randn(num_balls,
num_balls,4); m_box = randn(num_balls,2); m_coll = randn(num_balls, num_balls,4); z = randn(num_balls,2);

for i = 1:1000
% Process left nodes
    for j = 1:num_balls  % First process box nodes
       [m_box(j,1),m_box(j,2),u_box(j,1)u_box(j,2)]= F_box(z(j,1),z(j,2),u_box(j,1),u_box(j,2));
    end
    for j = 1:num_balls-1  % Second process coll nodes
        for k = j+1:num_balls
      [m_coll(j,k,1),m_coll(j,k,2),m_coll(j,k,3),m_coll(j,k,4),u_coll(j,k,1),u_coll(j,k,2),u_coll(j,k,3),
        u_coll(j,k,4)]=
    F_coll(z(j,1),z(j,2),z(k,1),z(k,2),u_coll(j,k,1),u_coll(j,k,2),u_coll(j,k,3),u_coll(j,k,4) );
        end
    end
% Process right nodes
    z = 0*z;
for i = 1:num_balls
        z(i,1) = z(i,1) + m_box(i,1);z(i,2) = z(i,2) + m_box(i,2);
    end
    for j = 1:num_balls-1
        for k = j+1:num_balls
            z(j,1) = z(j,1) + m_coll(j,k,1);z(j,2) = z(j,2) + m_coll(j,k,2);
            z(k,1) = z(k,1) + m_coll(j,k,3);z(k,2) = z(k,2) + m_coll(j,k,4);
        end
    end
    z = z / num_balls;
end
```



Circle Packing

Circle Packing



Non-smooth Filtering

Fused Lasso*:
$$\min_{z\in\mathbb{R}^p} \frac{1}{2}\sum_{i=1}^{p}(z_i - y_i)^2 + \lambda\sum_{i=1}^{p-1}|z_{i+1} - z_i|$$

*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering

Fused Lasso*:
$$\min_{z \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{p} \underbrace{(z_i - y_i)^2}_{\text{quad}} + \lambda \sum_{i=1}^{p-1} |z_{i+1} - z_i|$$

*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering

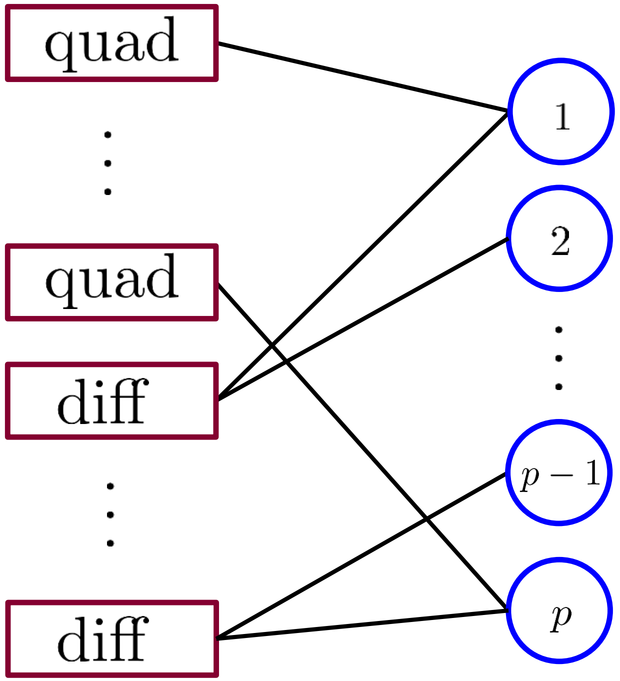
Fused Lasso*:

$$\min_{z \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{p} \underbrace{(z_i - y_i)^2}_{\text{quad}} + \lambda \sum_{i=1}^{p-1} \underbrace{|z_{i+1} - z_i|}_{\text{diff}}$$

*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering

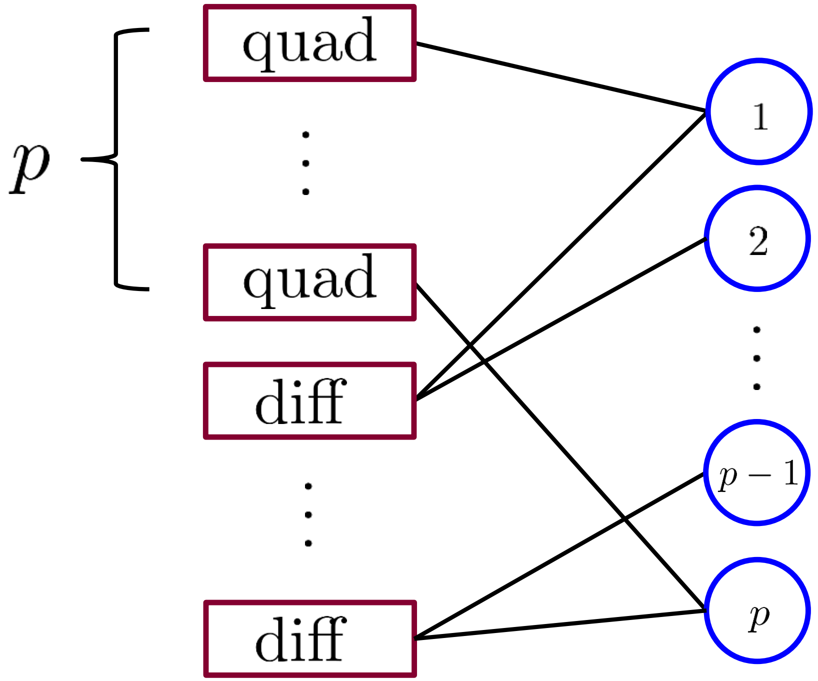
Fused Lasso*: $$\min_{z\in\mathbb{R}^p} \frac{1}{2}\sum_{i=1}^{p} \underbrace{(z_i - y_i)^2}_{\text{quad}} + \lambda \sum_{i=1}^{p-1} \underbrace{|z_{i+1} - z_i|}_{\text{diff}}$$



*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering

Fused Lasso*: $$\min_{z \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{p} \underbrace{(z_i - y_i)^2}_{\text{quad}} + \lambda \sum_{i=1}^{p-1} \underbrace{|z_{i+1} - z_i|}_{\text{diff}}$$

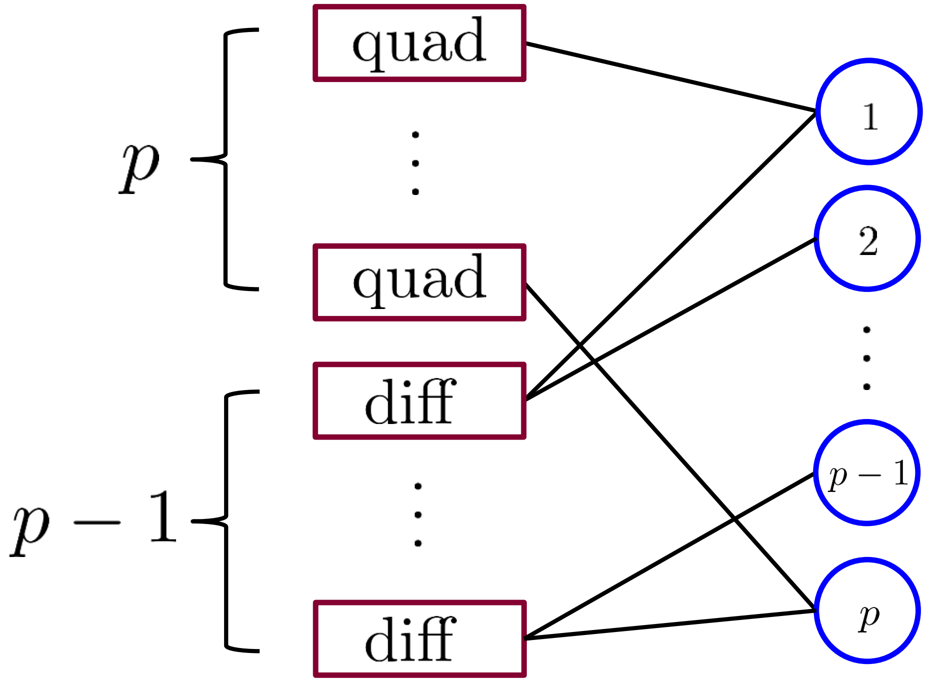


*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering

Fused Lasso*:

$$\min_{z\in\mathbb{R}^p} \frac{1}{2}\sum_{i=1}^{p}\underbrace{(z_i-y_i)^2}_{\text{quad}} + \lambda\sum_{i=1}^{p-1}\underbrace{|z_{i+1}-z_i|}_{\text{diff}}$$



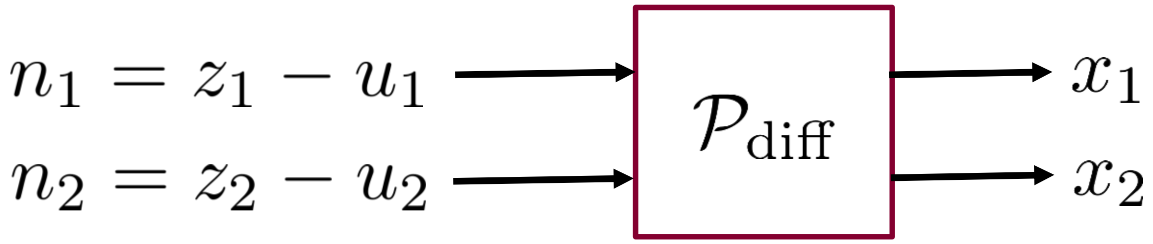
*For a different algorithm to solve a more general version of this problem see: J. Bento, R. Furmaniak, S. Ray, *"On the complexity of the weighted fused Lasso"*, 2018

Non-smooth Filtering - quad

$$n = z - u \longrightarrow \boxed{\mathcal{P}_{\text{quad}}} \longrightarrow x$$

$$x = \arg\min_{s} \frac{1}{2}(s - y_i)^2 + \frac{\rho}{2}(s - n)^2 \longrightarrow x = \frac{n\rho + y_i}{1 + \rho}$$

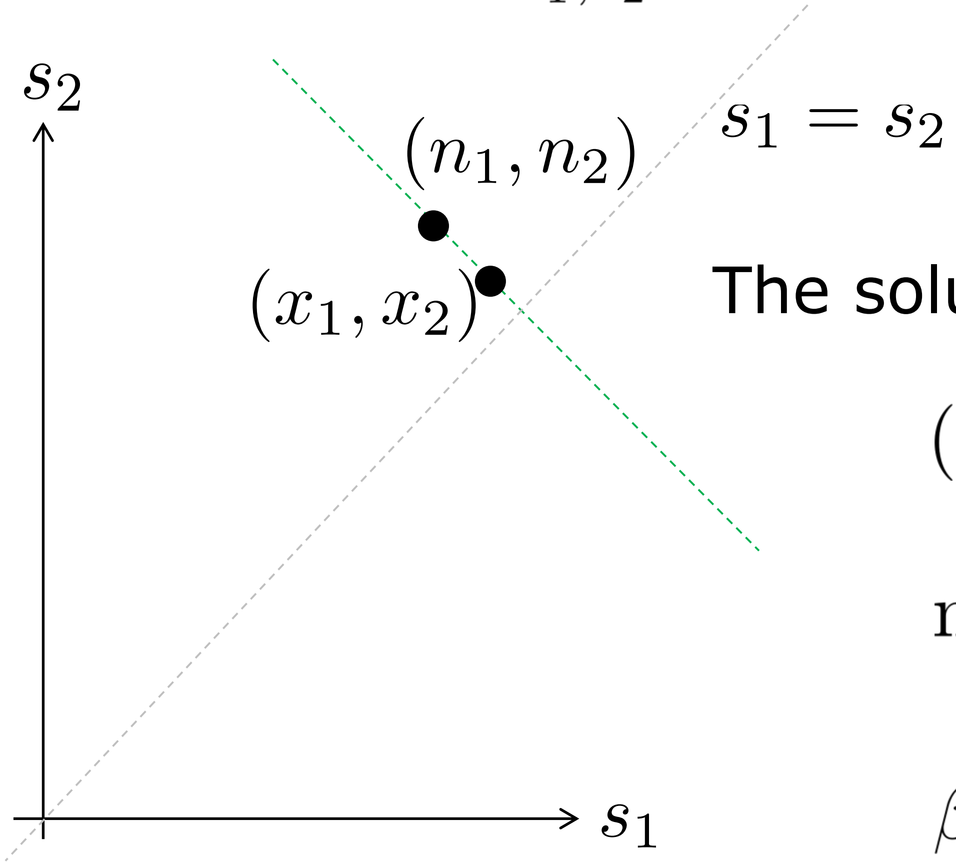
Non-smooth Filtering - diff



$$(x_1, x_2) = \arg\min_{s_1, s_2} \lambda |s_2 - s_1| + \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2$$

Non-smooth Filtering - diff

$$(x_1, x_2) = \arg\min_{s_1, s_2} \lambda |s_2 - s_1| + \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2$$



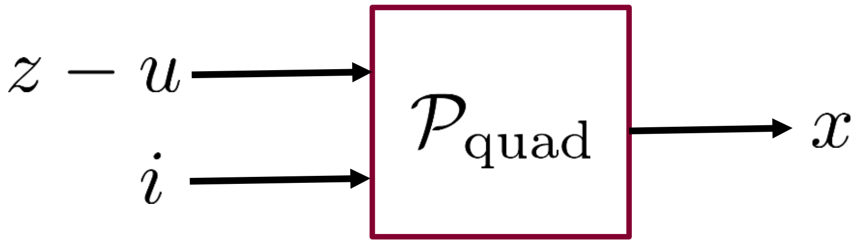
The solution must be along this line, thus:

$$(s_1, s_2) = (n_1, n_2) + \beta(1, -1)$$

$$\min_{\beta} \lambda \left| \frac{n_1 - n_2}{2} + \beta \right| + \frac{\rho}{2}\beta^2$$

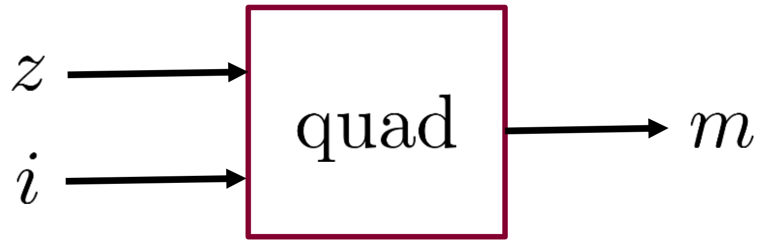
$$\beta = \text{thres}\left(\frac{n_1 - n_2}{2}, \frac{\lambda}{\rho} \right)$$

Non-smooth Filtering - quad



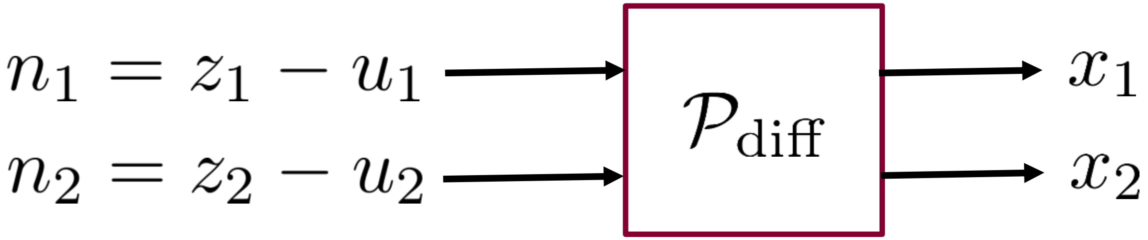

```
function [ x ] = P_quad( z_minus_u, i )

    global y;
    global rho;

    x = (z_minus_u*rho + y(i))/(1+rho);

end
```

Non-smooth Filtering - quad




```
function [ m, new_u] = F_quad(z, u, i)

    % Compute internal updates
    x = P_quad(z  -  u, i);

    new_u = u + (x  -  z);
    % Compute outgoing messages
    m = new_u + x;

end
```

Non-smooth Filtering - diff

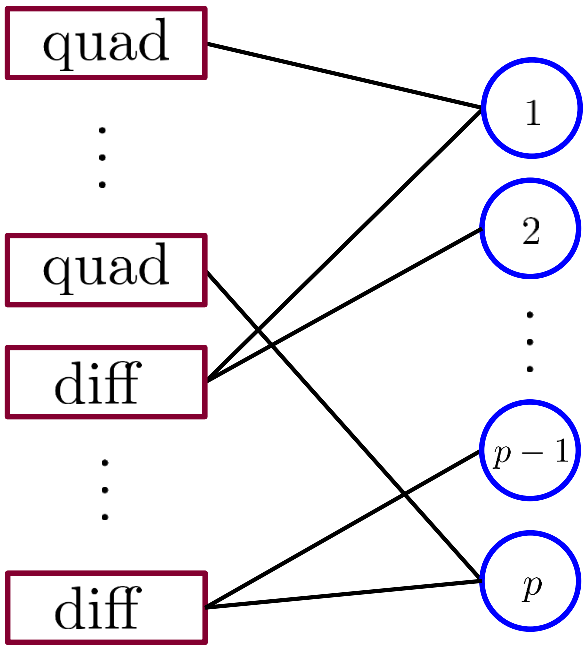



```
function [ x_1, x_2 ] = P_diff(z_minus_u_1, z_minus_u_2)

    global rho; global lambda;

    beta = max(-lambda/rho, min(lambda/rho,(z_minus_u_2 - z_minus_u_1)/2));
    x_1 = z_minus_u_1 + beta;
    x_2 = z_minus_u_2 - beta;

end
```

Non-smooth Filtering - diff

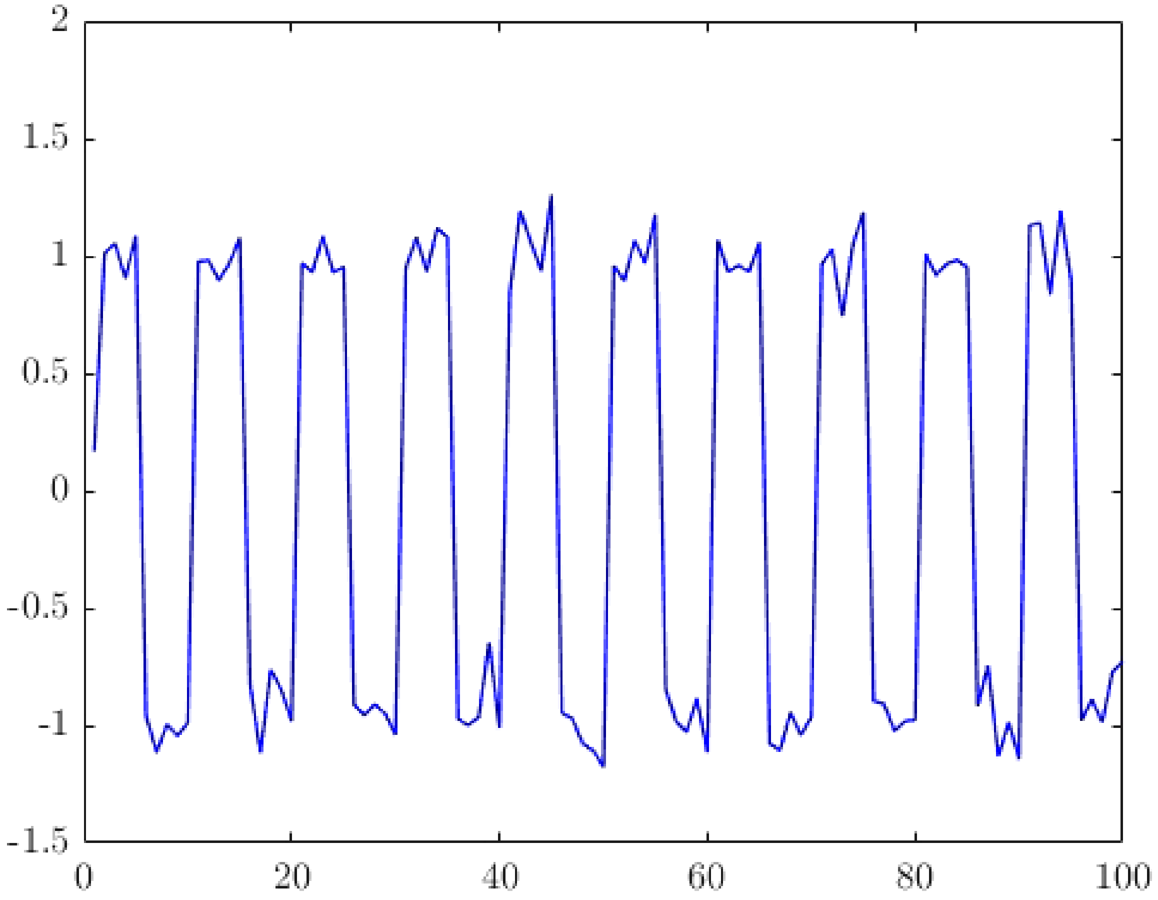

```
function [ m_1, m_2, new_u_1, new_u_2 ] = F_diff( z_1, z_2, u_1, u_2 )

    % Compute internal updates
    [x_1, x_2] = P_diff( z_1 - u_1, z_2 - u_2);

    new_u_1 = u_1 + (x_1  -  z_1);
    new_u_2 = u_2 + (x_2  -  z_2);

    % Compute outgoing messages
    m_1 = new_u_1 + x_1;
    m_2 = new_u_2 + x_2;

end
```

```
global y; global rho; global lambda;
n = 100; lambda = 0.7; rho = 1;
y = sign(sin(0:10*2*pi/(n-1):10*2*pi))' + 0.1*randn(n,1);

% Initialization
u_quad = randn(n,1); u_diff = randn(n-1,2); m_quad = randn(n,1); m_diff = randn(n-1,2);
z = randn(n,1);

for i=1:1000
    % Process left nodes
        % First process quad nodes
        for i = 1:n
            [m_quad(i) ,  u_quad(i)] = F_quad( z(i), u_quad(i),i );
        end
        % Second process diff nodes
        for j = 1:n-1
        [m_diff(j,1),m_diff(j,2),u_diff(j,1),u_diff(j,2)]
        = F_diff(z(j),z(j+1),u_diff(j,1), u_diff(j,2));
        end
    % Process right nodes
    z = 0*z;
    for i = 2:n-1
       z(i)= (m_quad(i) + m_diff(i-1,2) + m_diff(i,1))/3;
    end
    z(1) = (m_quad(1) + m_diff(1,1))/2;
    z(n) = (m_quad(n) + m_diff(n-1,2))/2;
end
```



Non-smooth Filtering

Non-smooth Filtering



Sudoku Puzzle

			4
	3		
			2
	1		

Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block

			4
	3		
			2
	1		

Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block

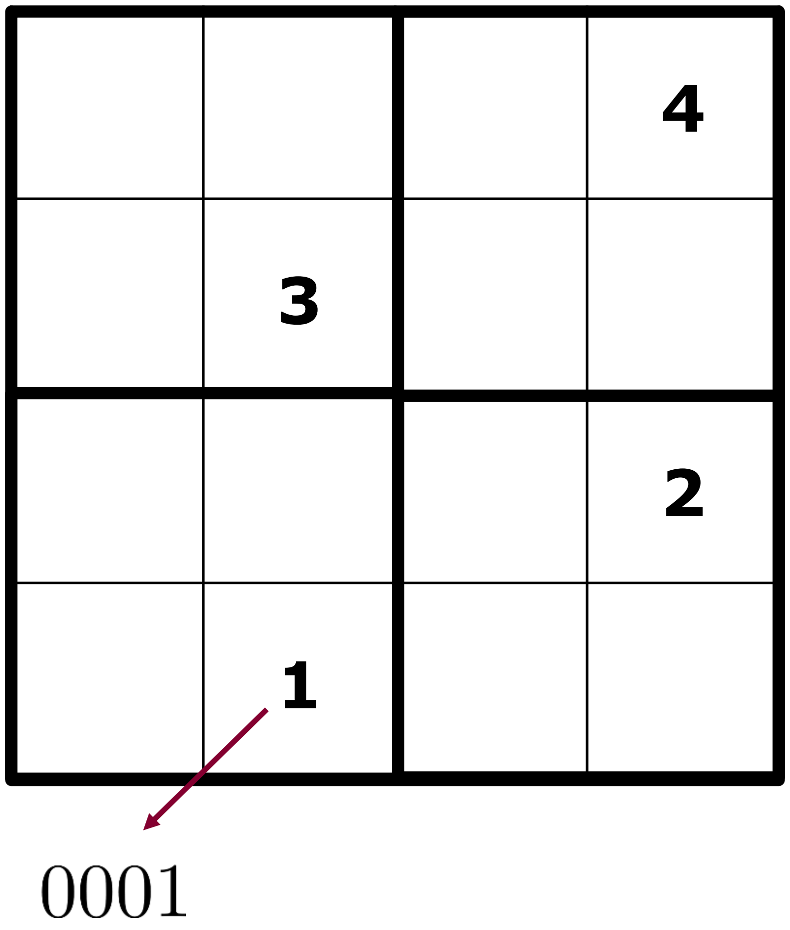
- Bit representations

			4
	3		
			2
	1		

Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block

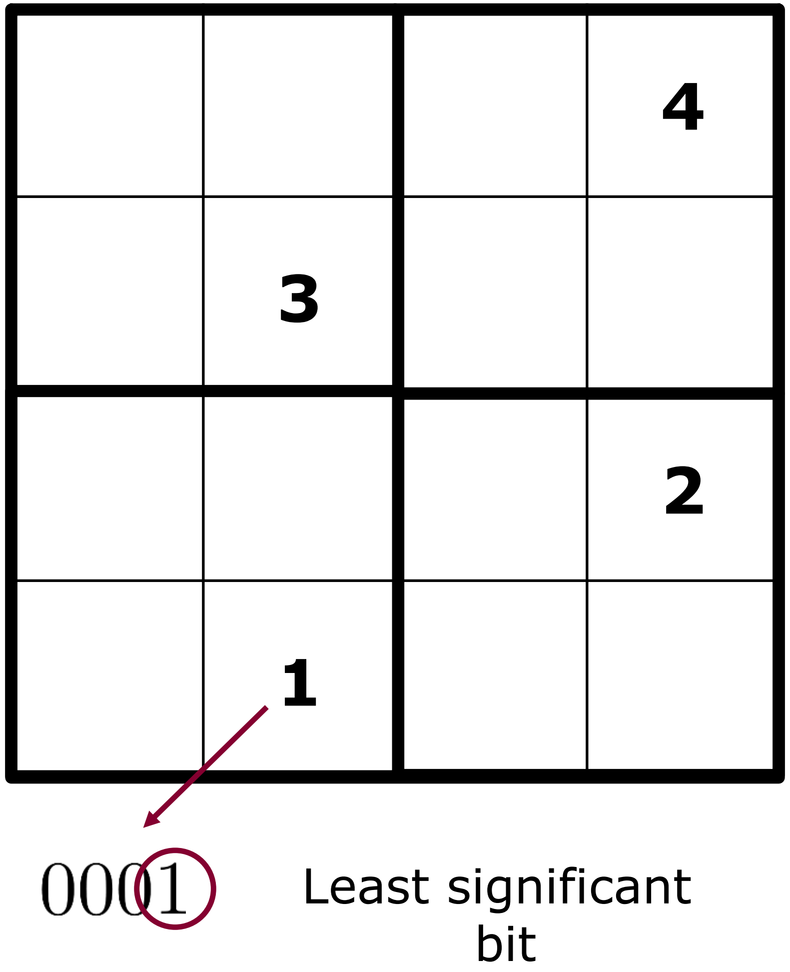
- Bit representations



Sudoku Puzzle

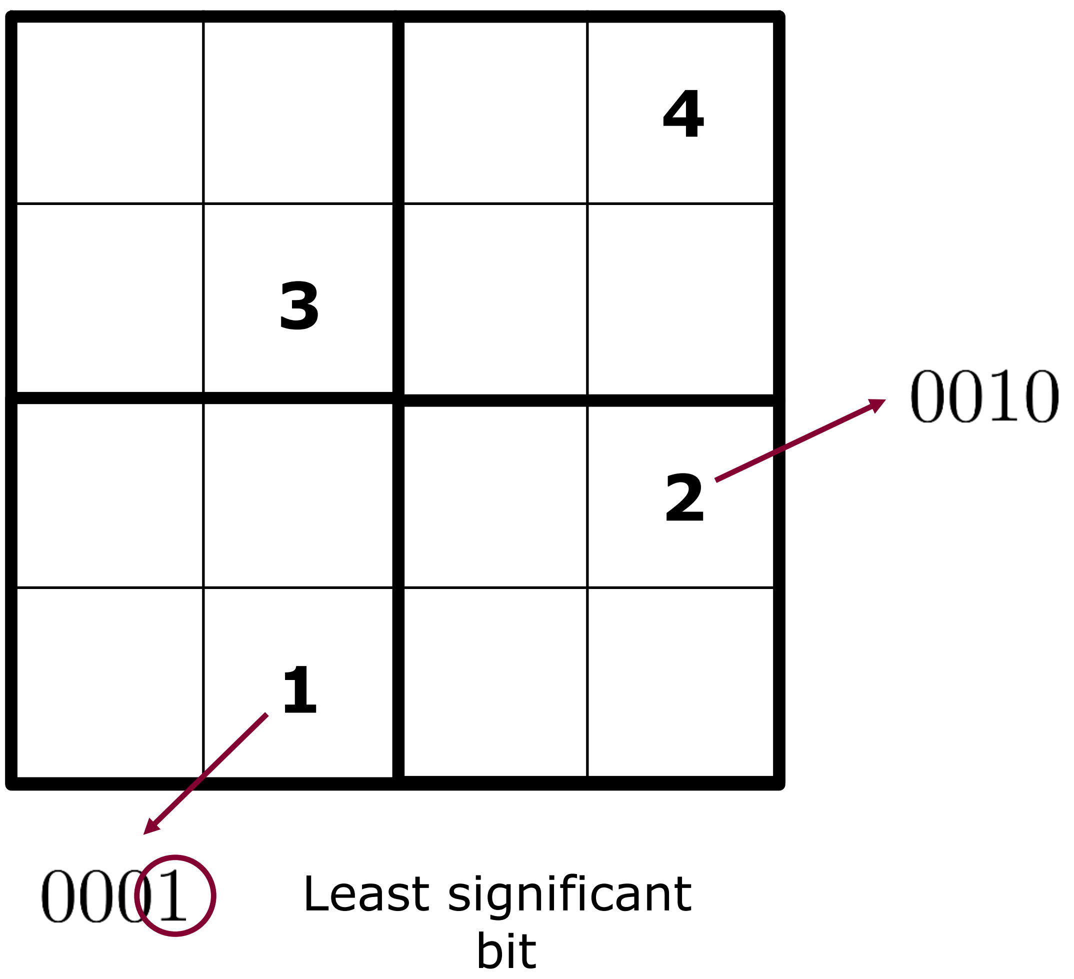
- Each number should be included once in each:
  - Row
  - Column
  - Block

- Bit representations



Sudoku Puzzle

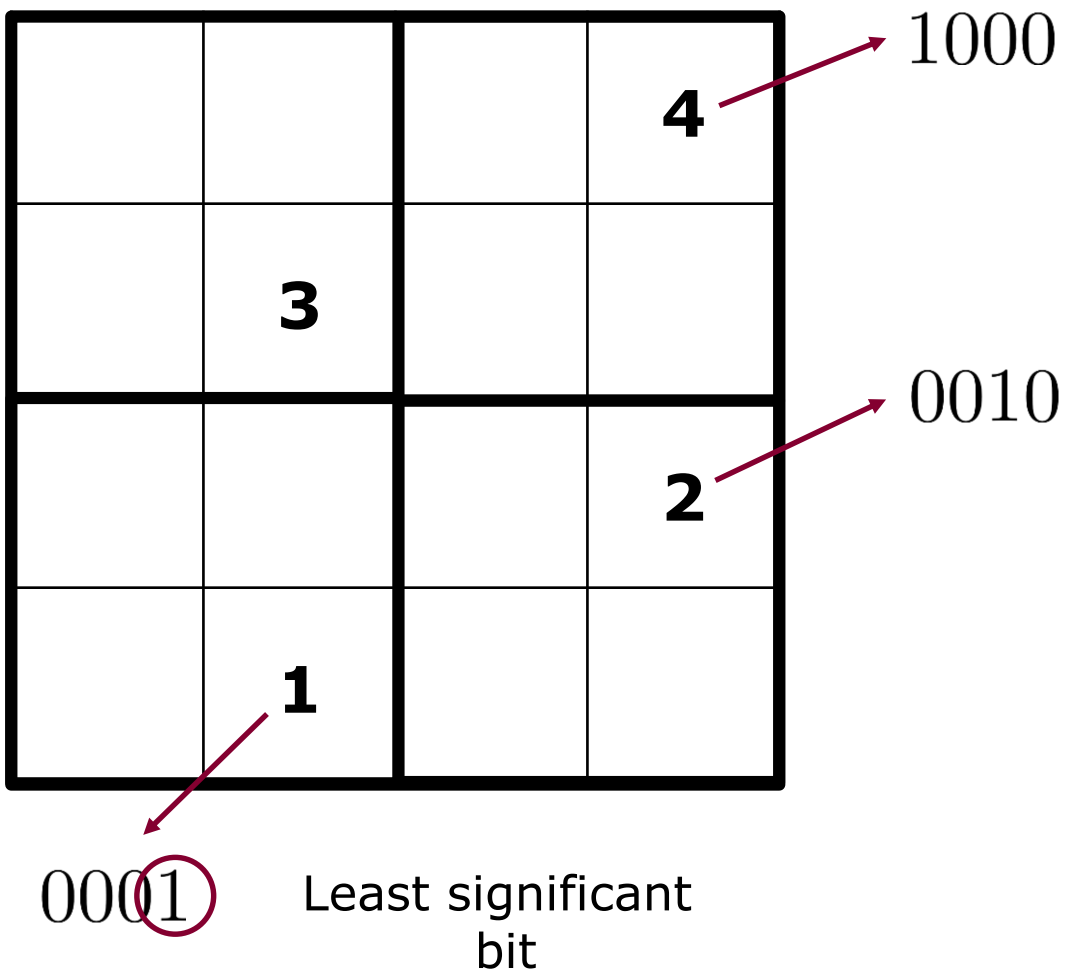
- Each number should be included once in each:
  - Row
  - Column
  - Block
- Bit representations



Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block

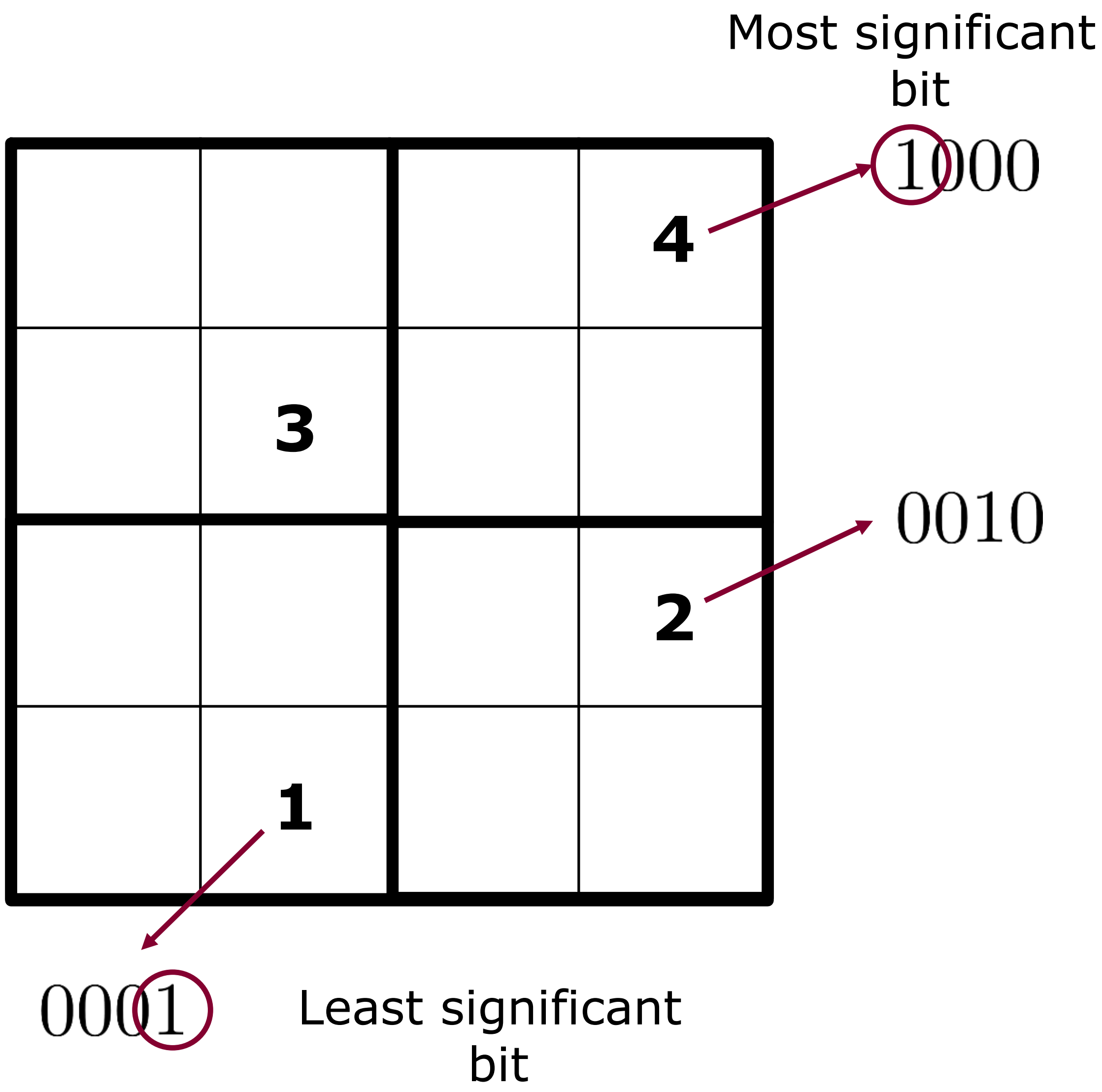
- Bit representations



Sudoku Puzzle

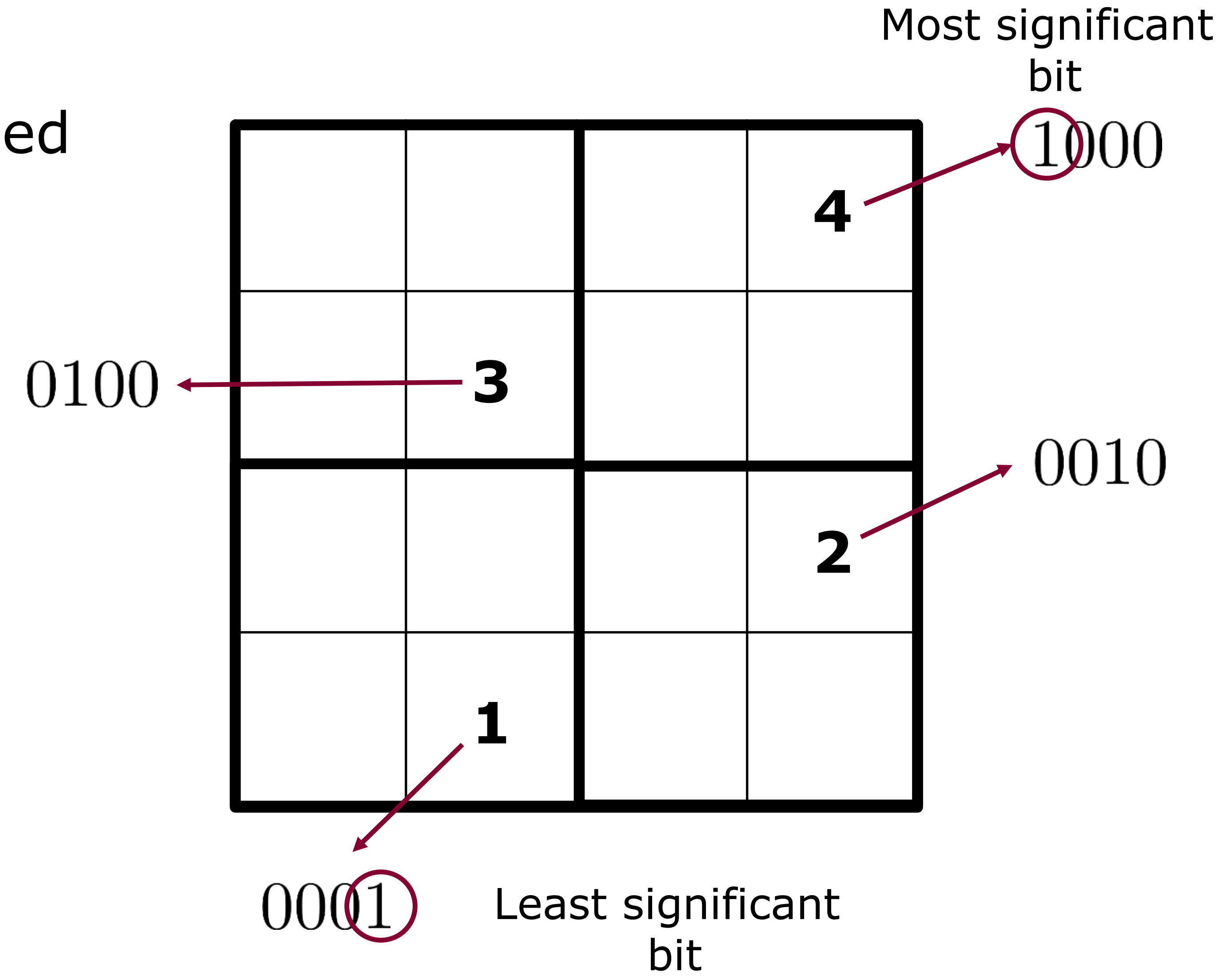
- Each number should be included once in each:
  - Row
  - Column
  - Block

- Bit representations



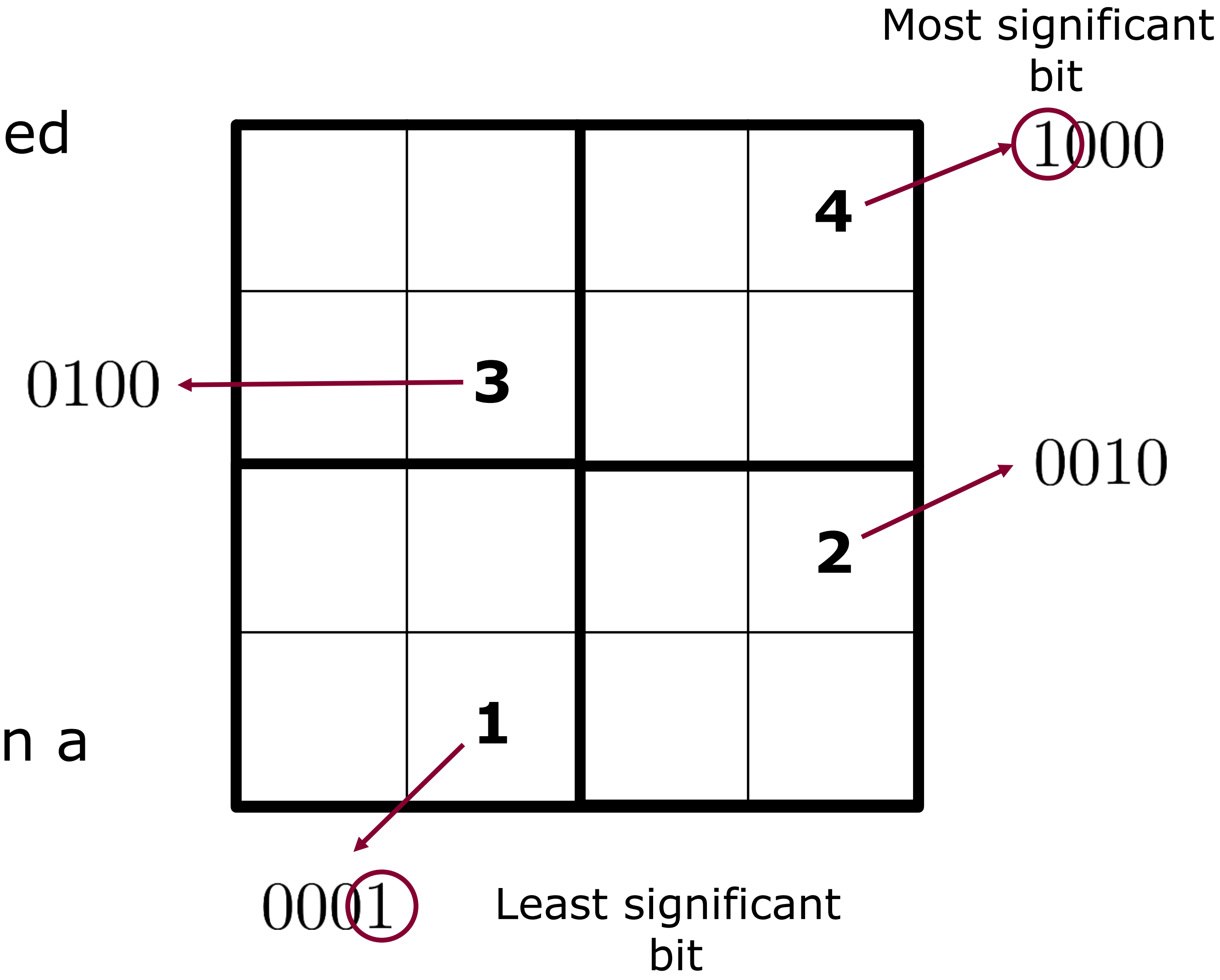
Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block
- Bit representations



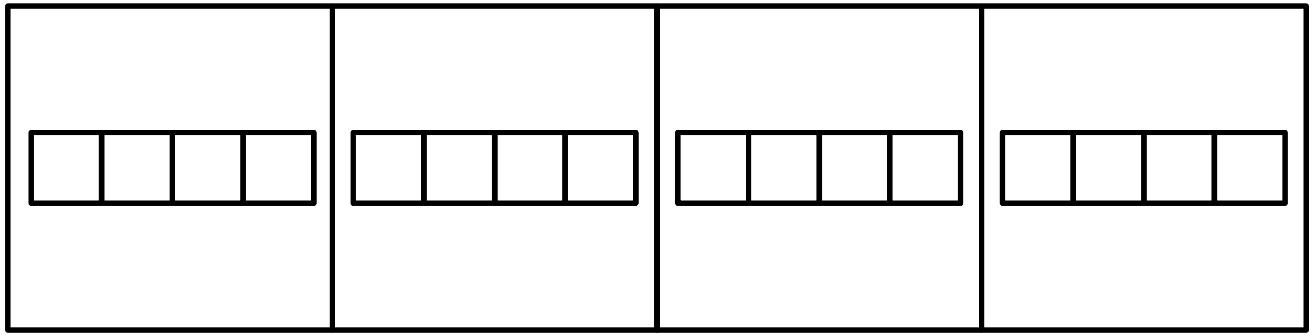
Sudoku Puzzle

- Each number should be included once in each:
  - Row
  - Column
  - Block
- Bit representations
- Only one digit should be one in a given cell

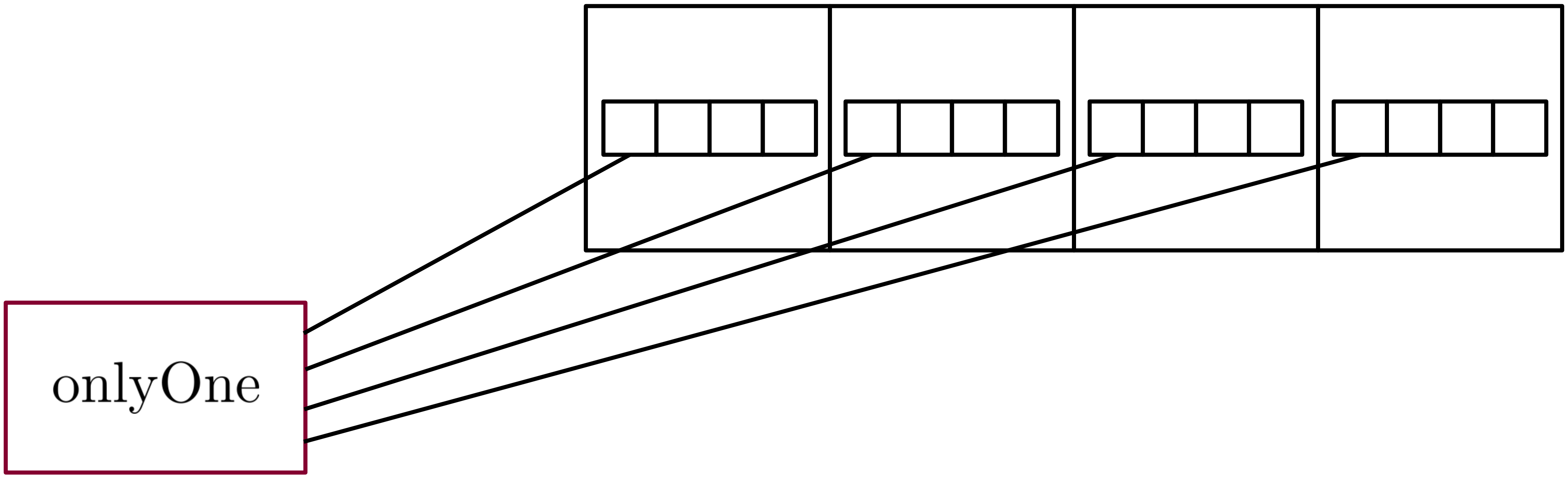


Sudoku Puzzle - onlyOne

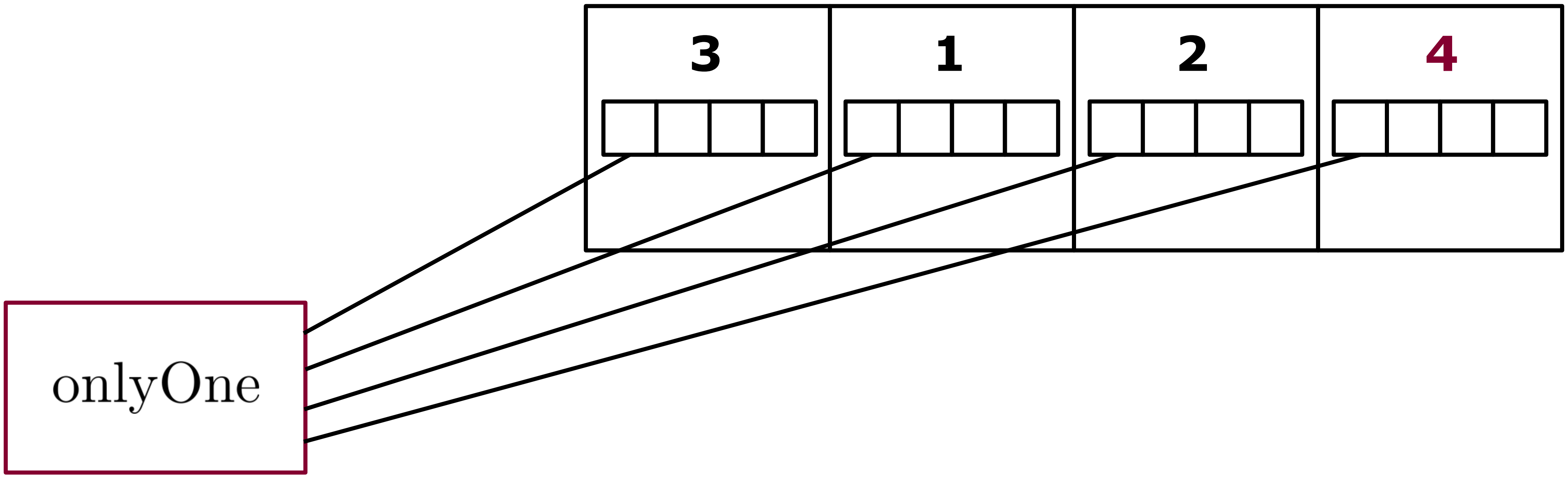
Sudoku Puzzle - onlyOne



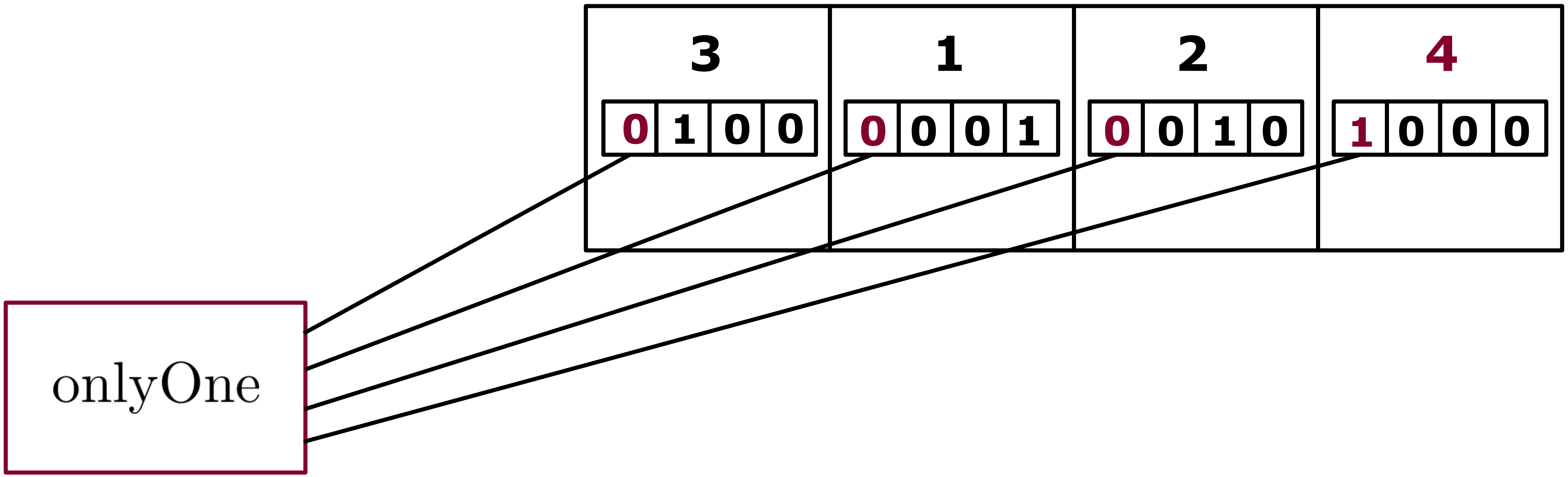
Sudoku Puzzle - onlyOne



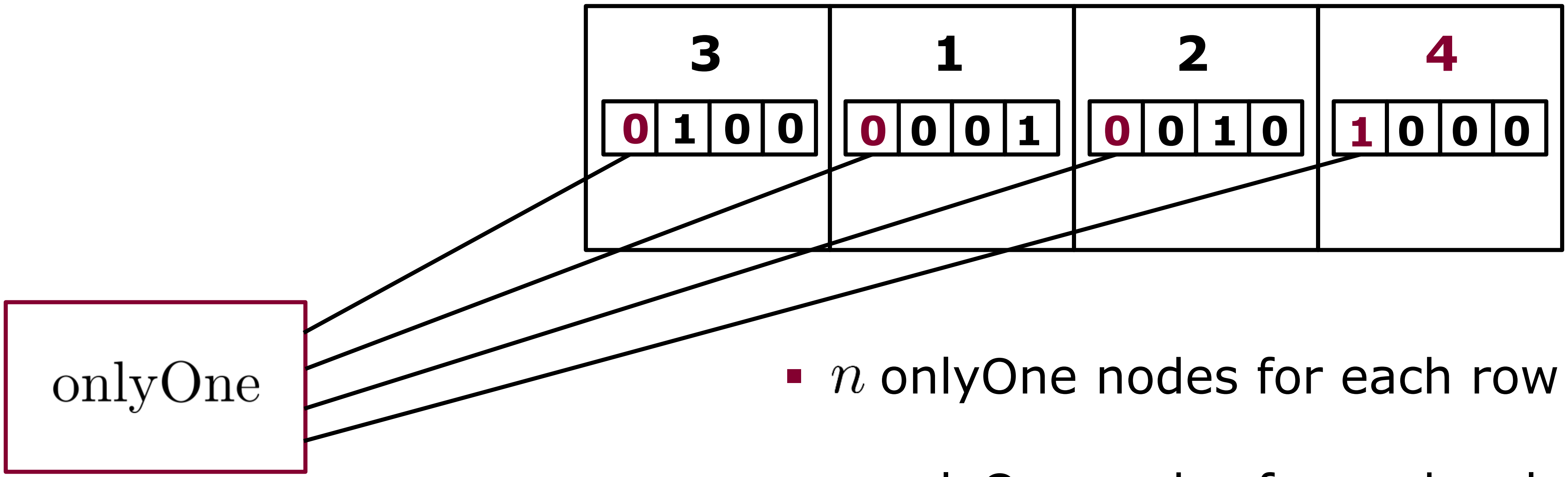
Sudoku Puzzle - onlyOne



Sudoku Puzzle - onlyOne



Sudoku Puzzle - onlyOne



- n onlyOne nodes for each row
- n onlyOne nodes for each column
- n onlyOne nodes for each block
- n onlyOne nodes for each cell

Sudoku Puzzle - onlyOne

$$(x_1, x_1, \ldots) = \underset{s_1, s_2, \ldots}{\arg\min}\ \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots$$

$$\text{subject to}$$
$$\text{only one } s_i \text{ is 1 and all others are 0}$$

Find the minimum via direct inspection of the different solutions values

$$(1 - n_1)^2 + (0 - n_2)^2 + (0 - n_3)^2 + \ldots$$
$$(0 - n_1)^2 + (1 - n_2)^2 + (0 - n_3)^2 + \ldots$$
$$(0 - n_1)^2 + (0 - n_2)^2 + (1 - n_3)^2 + \ldots$$

Sudoku Puzzle - onlyOne

Compare each of the following values

$$(1-n_1)^2 + (0-n_2)^2 + (0-n_3)^2 + \dots$$

$$(0-n_1)^2 + (1-n_2)^2 + (0-n_3)^2 + \dots$$

$$(0-n_1)^2 + (0-n_2)^2 + (1-n_3)^2 + \dots$$

against the reference

$$n_1^2 + n_2^2 + n_3^2 + \dots$$

Sudoku Puzzle - onlyOne

Compare each of the following values

$$(1-n_1)^2+(0-n_2)^2+(0-n_3)^2+\ldots$$
$$(0-n_1)^2+(1-n_2)^2+(0-n_3)^2+\ldots$$
$$(0-n_1)^2+(0-n_2)^2+(1-n_3)^2+\ldots$$

against the reference

$$n_1^2+n_2^2+n_3^2+\ldots$$

notice that

$$((1-n_1)^2+(0-n_2)^2+\ldots)-({n_1}^2+{n_2}^2+\ldots)=-2n_1$$

Sudoku Puzzle - onlyOne

Compare each of the following values

$$(1-n_1)^2+(0-n_2)^2+(0-n_3)^2+\dots$$
$$(0-n_1)^2+(1-n_2)^2+(0-n_3)^2+\dots$$
$$(0-n_1)^2+(0-n_2)^2+(1-n_3)^2+\dots$$

against the reference

$$n_1^2+n_2^2+n_3^2+\dots$$

notice that

$$((1-n_1)^2+(0-n_2)^2+\dots)-({n_1}^2+{n_2}^2+\dots)=-2n_1$$

therefore $(x_1,x_2,\dots)=(0,\dots,0,1,0,\dots,0)$

Sudoku Puzzle - onlyOne

Compare each of the following values

$$(1-n_1)^2 + (0-n_2)^2 + (0-n_3)^2 + \ldots$$
$$(0-n_1)^2 + (1-n_2)^2 + (0-n_3)^2 + \ldots$$
$$(0-n_1)^2 + (0-n_2)^2 + (1-n_3)^2 + \ldots$$

against the reference

$$n_1^2 + n_2^2 + n_3^2 + \ldots$$

notice that

$$((1-n_1)^2 + (0-n_2)^2 + \ldots) - ({n_1}^2 + {n_2}^2 + \ldots) = -2n_1$$

therefore $$(x_1, x_2, \ldots) = (0, \ldots, 0, 1, 0, \ldots, 0)$$

Index corresponds to the maximum n_i

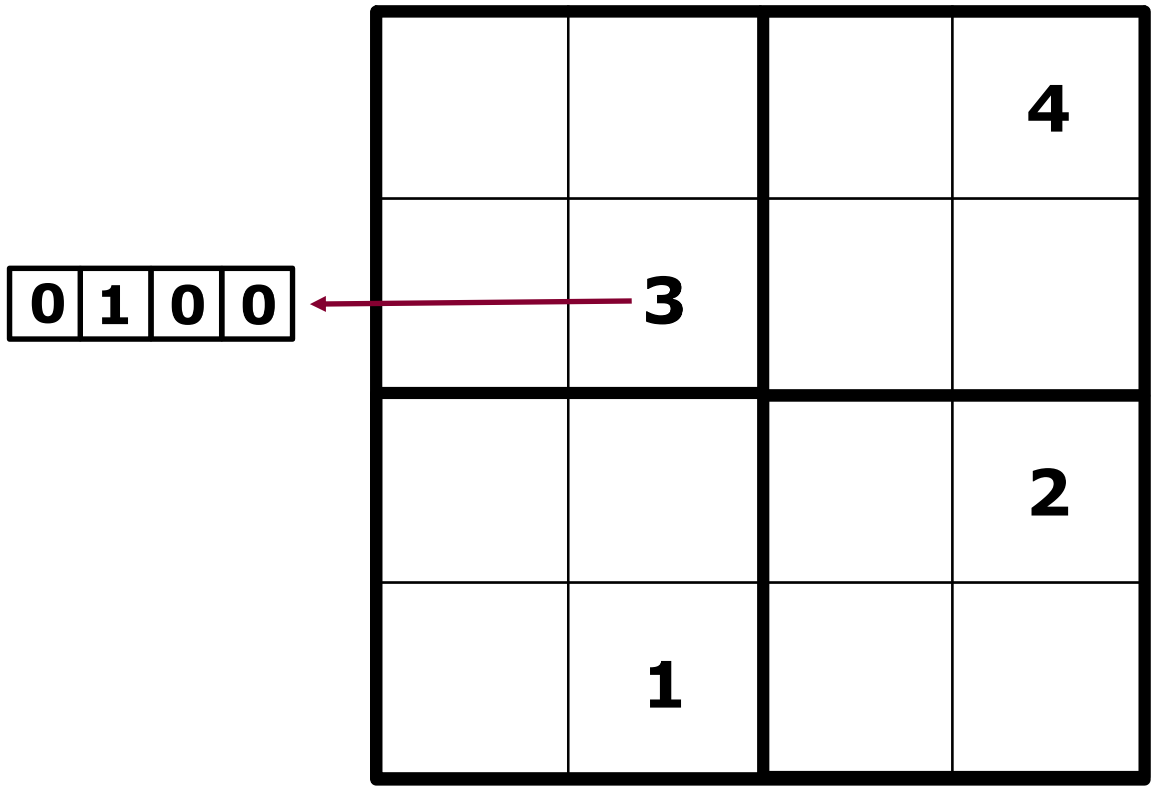
Sudoku Puzzle - knowThat

- Some cell values are known from the beginning
- knowThat functions constantly produce those values for the corresponding cells

			4
	3		
			2
	1		

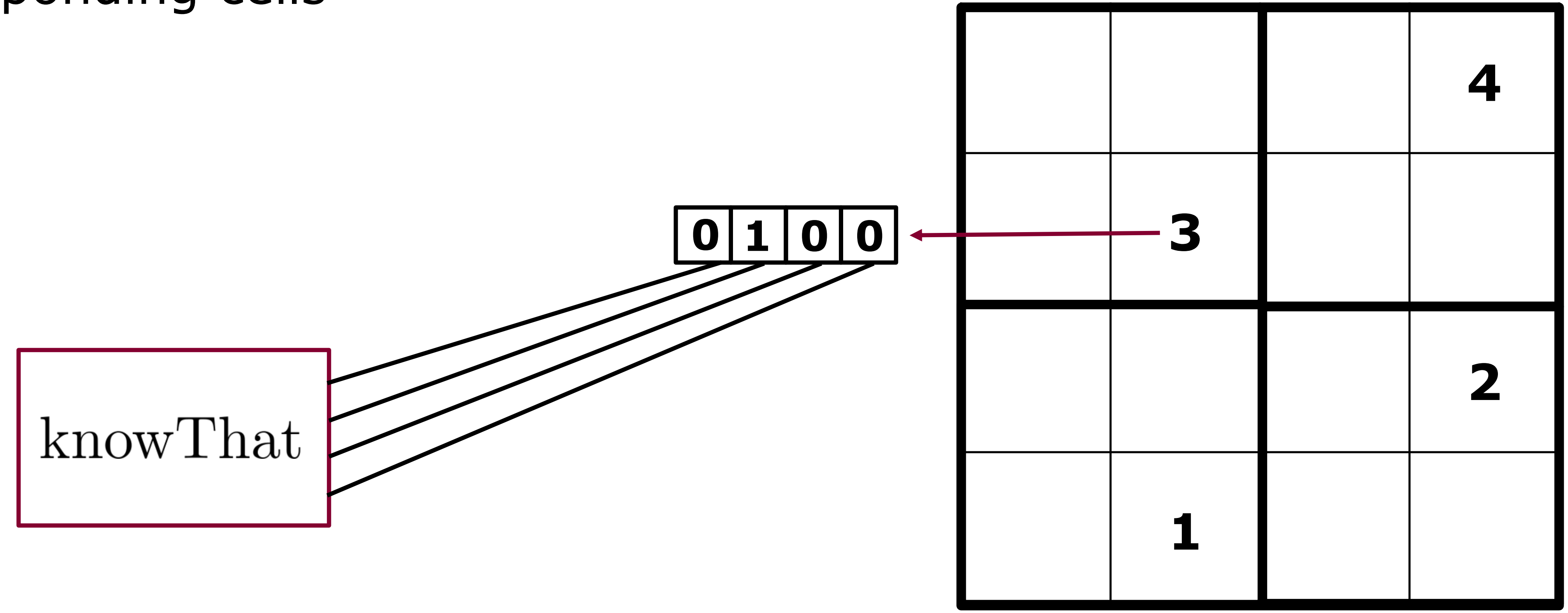
Sudoku Puzzle - knowThat

- Some cell values are known from the beginning
- knowThat functions constantly produce those values for the corresponding cells

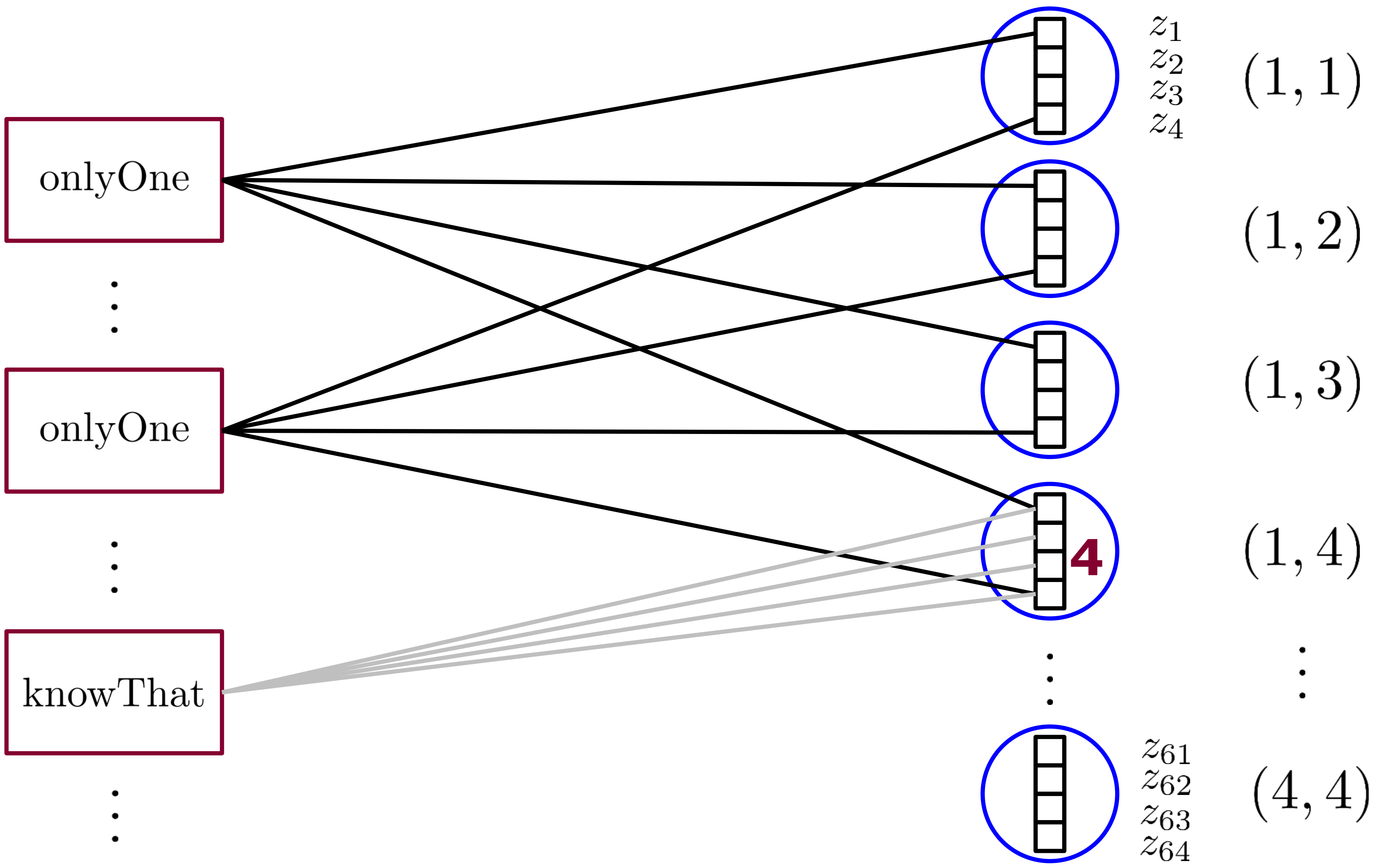


Sudoku Puzzle - knowThat

- Some cell values are known from the beginning
- knowThat functions constantly produce those values for the corresponding cells



Sudoku Puzzle – Factor graph



Sudoku Puzzle - onlyOne

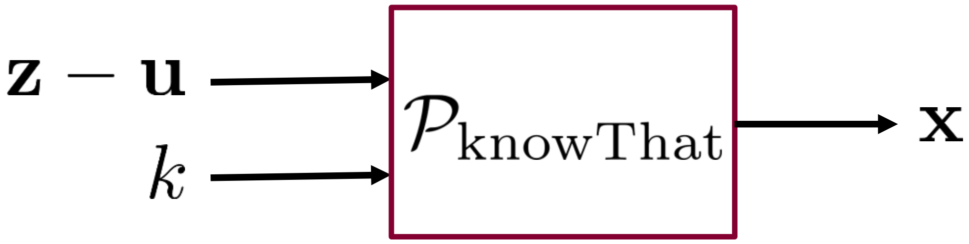
$$\mathbf{z} - \mathbf{u} \longrightarrow \boxed{\mathcal{P}_{\text{onlyOne}}} \longrightarrow \mathbf{x}$$

```
function [ X ] = P_onlyOne( Z_minus_U )
    %X and Z_minus U are n by one vectors

    X =0*Z_minus_U;
    [~,b] = max(Z_minus_U);
    X(b) = 1;

end
```

Sudoku Puzzle - onlyOne


```
function [ M, new_U ] = F_onlyOne( Z, U )
    %M, Z and U are n by one vectors

    % Compute internal updates
    X = P_onlyOne( Z - U );

    new_U = U + (X  -  Z);

    % Compute outgoing messages
    M = new_U + X;

end
```

Sudoku Puzzle - knowThat




```
function [ X ] = P_knowThat( k, Z_minus_U )
    %Z_minus_U is an n by 1 vector

    X = 0*Z_minus_U;
    X(k) = 1;

end
```

Sudoku Puzzle - knowThat


```
function [ M, new_U ] = F_knowThat(k, Z, U )

    % Compute internal updates
    X = P_knowThat(k, Z - U );

    new_U = U + (X  -  Z);

    % Compute outgoing messages
    M = new_U + X;

end
```

```
n = 9; known_data = [1,4,6;1,7,4;2,1,7;2,6,3;2,7,6;3,5,9;3,6,1;3,8,8;5,2,5;5,4,1;5,5,8;5,9,3;6,4,3;6,6,6;6,8,4;6,9,5;7,2,4;7,4,2;7,8,6;8,1,9;8,3,3;9,2,2;9,7,1;];
box_indices = 1:n;box_indices = reshape(box_indices,sqrt(n),sqrt(n));box_indices = kron(box_indices,ones(sqrt(n)));% box indexing
u_onlyOne_rows = randn(n,n,n);u_onlyOne_cols = randn(n,n,n);u_onlyOne_boxes = randn(n,n,n);u_onlyOne_cells = randn(n,n,n); % Initialization  (number , row, col)
m_onlyOne_rows = randn(n,n,n);m_onlyOne_cols = randn(n,n,n);m_onlyOne_boxes = randn(n,n,n);m_onlyOne_cells = randn(n,n,n);
u_knowThat = randn(n,n,n);m_knowThat = randn(n,n,n);z = randn(n,n,n);
for t = 1:1000
    % Process left nodes
    % First process knowThat nodes
    for i = 1:size(known_data,1)
        number = known_data(i,3);pos_row = known_data(i,1);pos_col = known_data(i,2);
        [m_knowThat(:,pos_row,pos_col),u_knowThat(:,pos_row,pos_col)] = F_knowThat(number,z(:,pos_row,pos_col),u_knowThat(:,pos_row,pos_col));
    end
    % Second process onlyOne nodes
    for number = 1:n % rows
        for pos_row = 1:n
            [m_onlyOne_rows(number,pos_row,:), u_onlyOne_rows(number,pos_row,:)] = F_onlyOne(z(number,pos_row,:),u_onlyOne_rows(number,pos_row,:));
        end
    end
    for number = 1:n %columns
        for pos_col = 1:n
            [m_onlyOne_cols(number,:,pos_col),u_onlyOne_cols(number,:,pos_col)] = F_onlyOne(z(number,:,pos_col),u_onlyOne_cols(number,:,pos_col));
        end
    end
    for number = 1:n %boxes
        for pos_box = 1:n
            [pos_row,pos_col] = find(box_indices==pos_box); linear_indices_for_box_ele = sub2ind([n,n,n],number*ones(n,1),pos_row,pos_col);
            [m_onlyOne_boxes(linear_indices_for_box_ele),u_onlyOne_boxes(linear_indices_for_box_ele)] =
            F_onlyOne(z(linear_indices_for_box_ele),u_onlyOne_boxes(linear_indices_for_box_ele) );
        end
    end
    for pos_col = 1:n  %cells
        for pos_row = 1:n
            [m_onlyOne_cells(:,pos_col,pos_row),u_onlyOne_cells(:,pos_col,pos_row) ] = F_onlyOne(z(:,pos_col,pos_row),u_onlyOne_cells(:,pos_col,pos_row));
        end
    end
    % Process right nodes
    z = 0*z;z = (m_onlyOne_rows + m_onlyOne_cols + m_onlyOne_boxes + m_onlyOne_cells)/4;
    for i = 1:size(known_data,1)
        number = known_data(i,3);pos_row = known_data(i,1);pos_col = known_data(i,2);
        z(number,pos_row,pos_col) = (4*z(number,pos_row,pos_col) + m_knowThat(number,pos_row,pos_col))/5;
    end
    final = zeros(n);
    for i = 1:n
        final = final + i*reshape(z(i,:,:),n,n);
    end
    disp(final);
end
```

Sudoku Puzzle – A (difficult) 9 by 9 example

			6			4		
7					3	6		
				9	1		8	
	5		1	8				3
			3		6		4	5
	4		2				6	
9		3						
	2					1		

http://elmo.sbs.arizona.edu/sandiway/sudoku/examples.html

Sudoku Puzzle – A (difficult) 9 by 9 example

Sudoku Puzzle – A (difficult) 9 by 9 example

7.0000	6.0000	-2.7500	4.0000	4.5000	3.0000	4.8000	2.5000	8.5000
7.3500	4.7500	7.5000	5.7500	4.2500	1.8000	6.6500	1.5000	12.2500
8.0000	4.5000	6.5000	5.7500	13.0000	-4.2000	-4.0000	11.6000	11.5000
4.2500	0.7500	-3.7500	18.0000	6.2500	2.7500	-1.2500	2.7500	-1.2500
4.5000	4.0000	9.7500	2.1500	14.3500	-1.0000	4.7500	6.7500	3.1000
4.2500	0.7500	5.7500	4.1000	13.0000	4.3000	4.5000	3.4500	12.5000
5.0000	-0.9500	10.7500	-0.1000	3.0000	8.0000	-3.7500	12.3500	2.5000
10.5000	0.7500	8.9000	0.2500	2.5000	8.5000	5.5000	5.0000	5.5000
1.5000	5.9000	7.7500	1.0000	6.0000	4.0000	3.9000	8.2500	1.7500

Sudoku Puzzle – A (difficult) 9 by 9 example

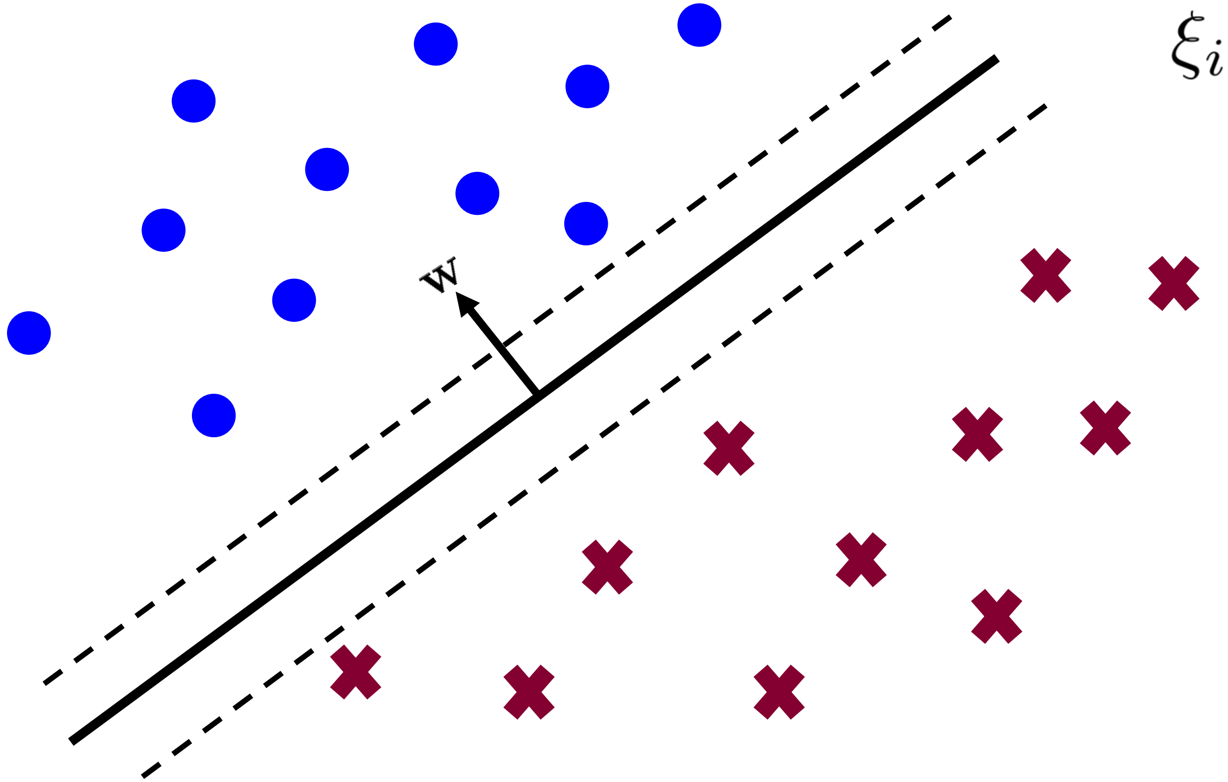
5.0000	8.0000	1.0000	6.0000	7.0000	2.0000	4.0000	3.0000	9.0000
7.0000	9.0000	2.0000	8.0000	4.0000	3.0000	6.0000	5.0000	1.0000
3.0000	6.0000	4.0000	5.0000	9.0000	1.0000	7.0000	8.0000	2.0000
4.0000	3.0000	8.0000	9.0000	5.0000	7.0000	2.0000	1.0000	6.0000
2.0000	5.0000	6.0000	1.0000	8.0000	4.0000	9.0000	7.0000	3.0000
1.0000	7.0000	9.0000	3.0000	2.0000	6.0000	8.0000	4.0000	5.0000
8.0000	4.0000	5.0000	2.0000	1.0000	9.0000	3.0000	6.0000	7.0000
9.0000	1.0000	3.0000	7.0000	6.0000	8.0000	5.0000	2.0000	4.0000
6.0000	2.0000	7.0000	4.0000	3.0000	5.0000	1.0000	9.0000	8.0000

Support Vector Machine

$$\min_{\xi, \mathbf{w}} \sum_{i=1}^{k} \xi_i + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\text{subject to} \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$$

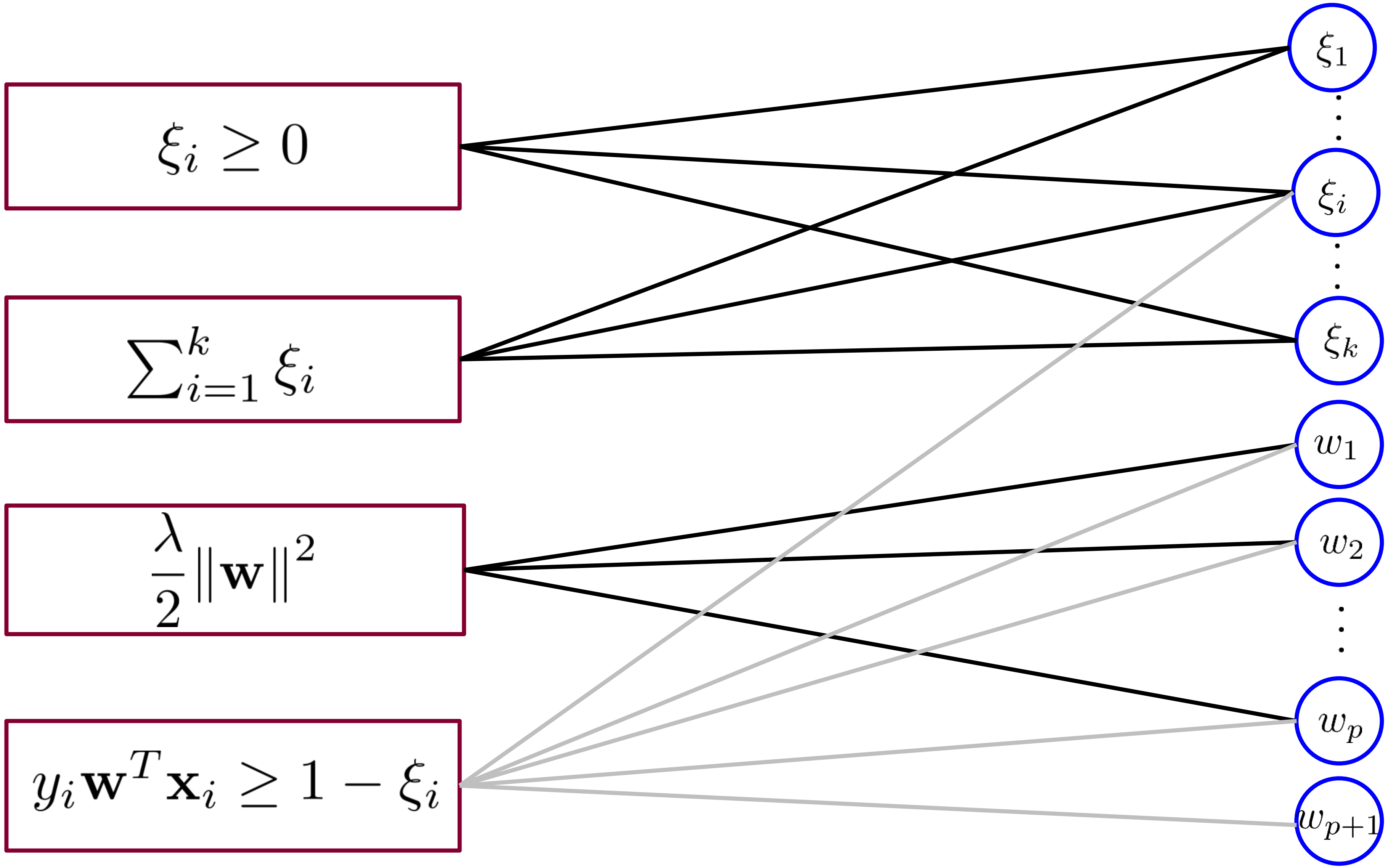
$$\xi_i \geq 0$$



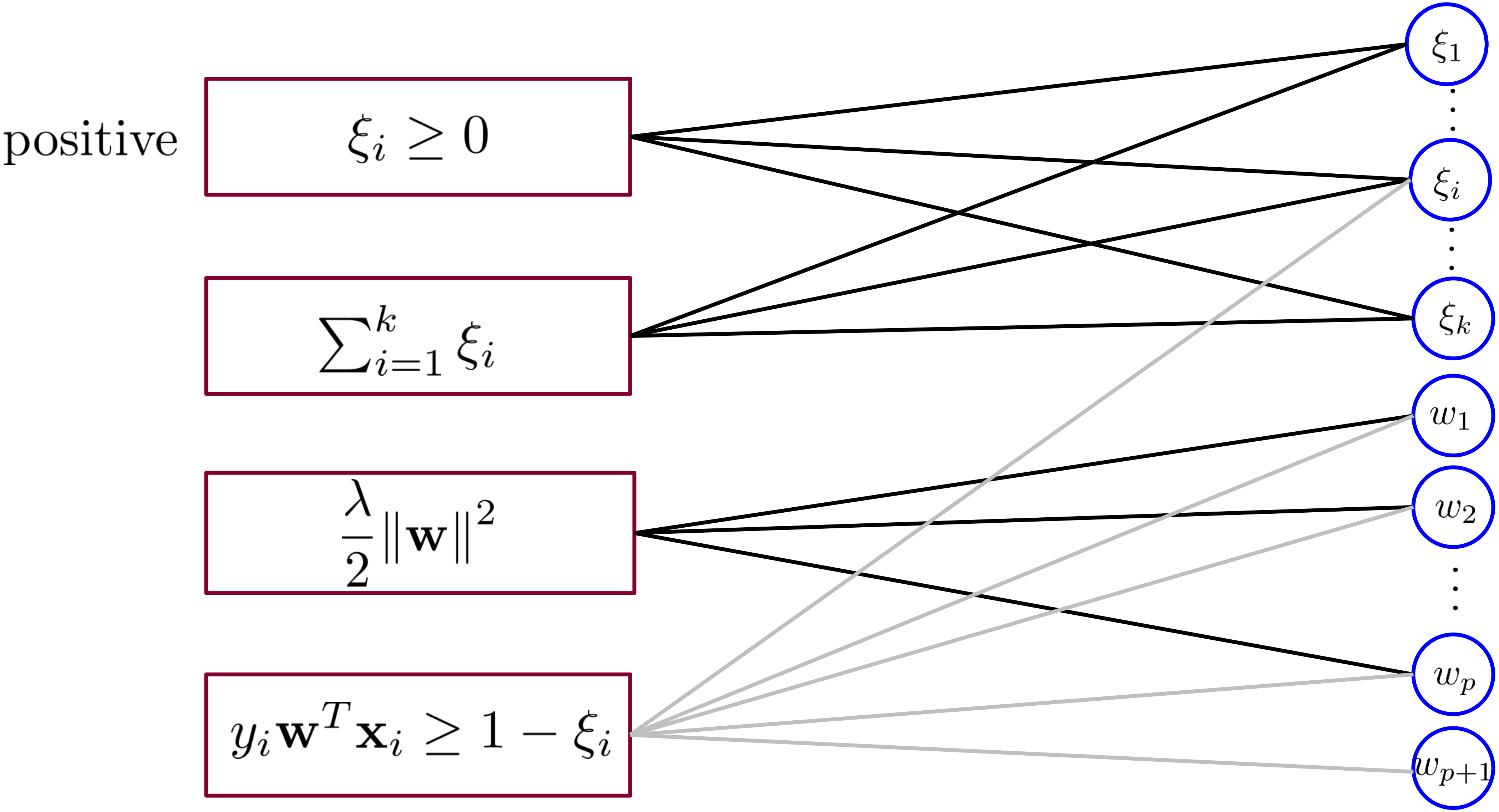
$\mathbf{x}_i$: p-dimensional data

y_i: i-th label $\quad i \in \{1, \ldots, k\}$

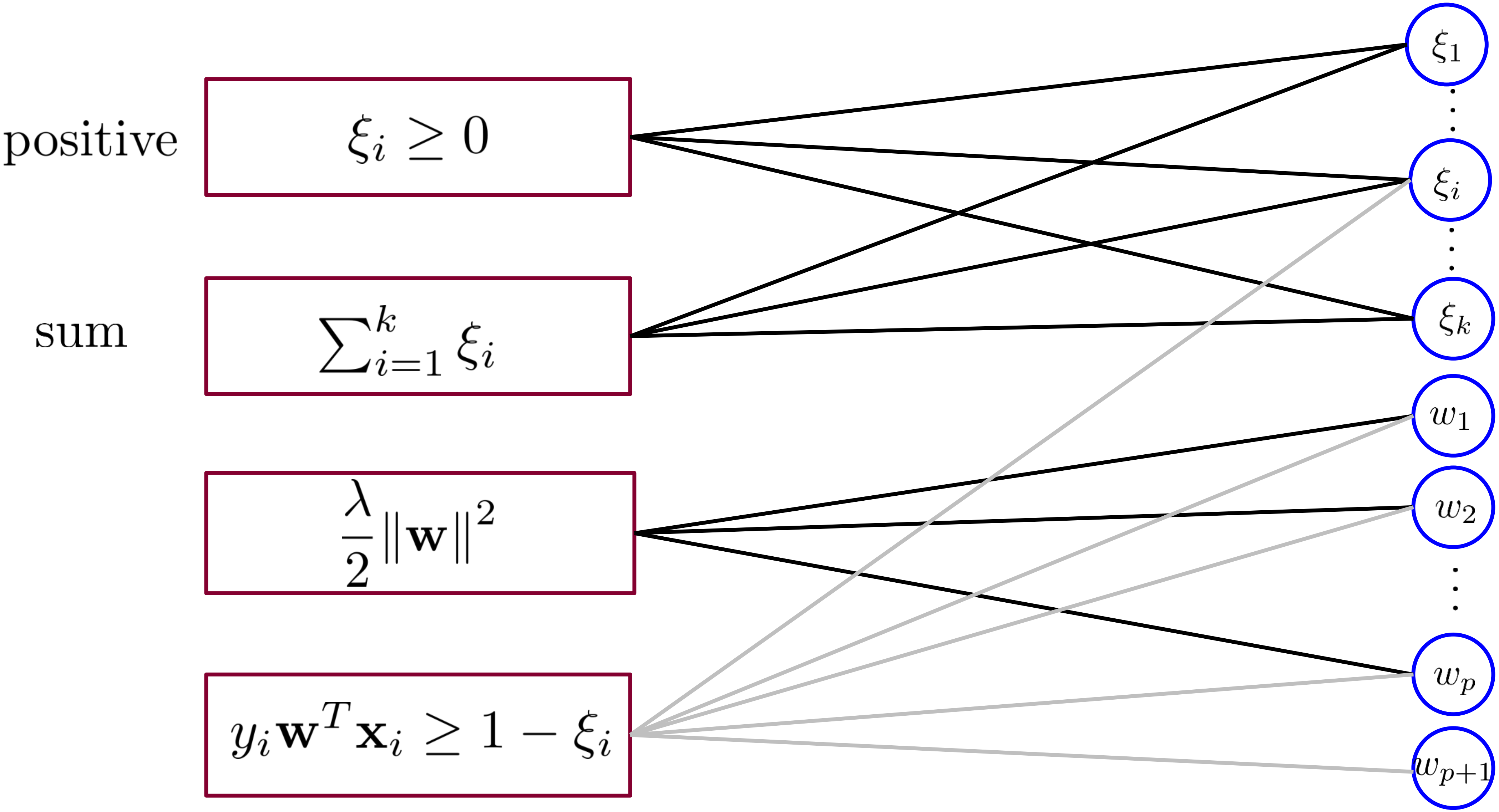
Support Vector Machine - ADMM



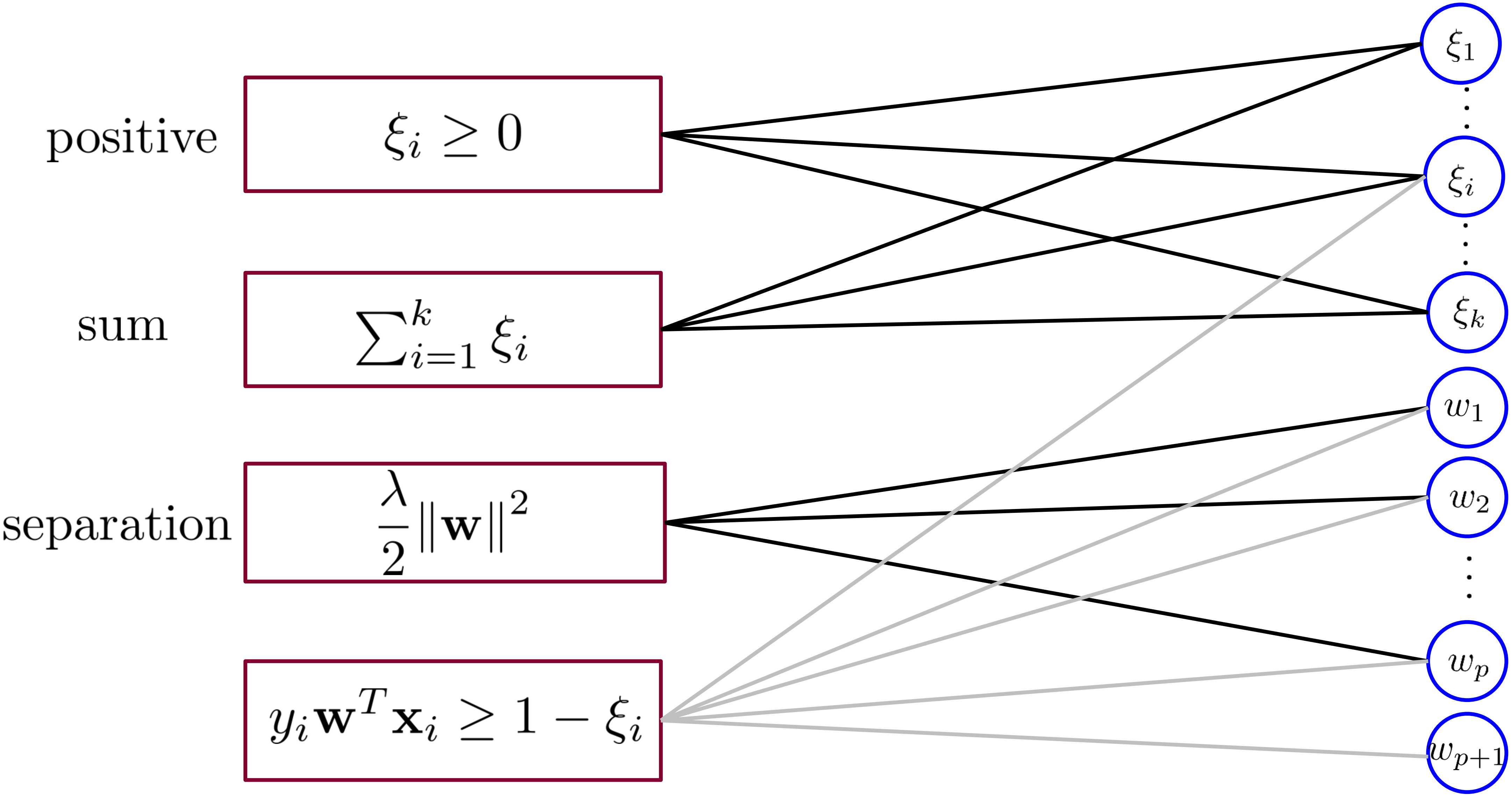
Support Vector Machine - ADMM



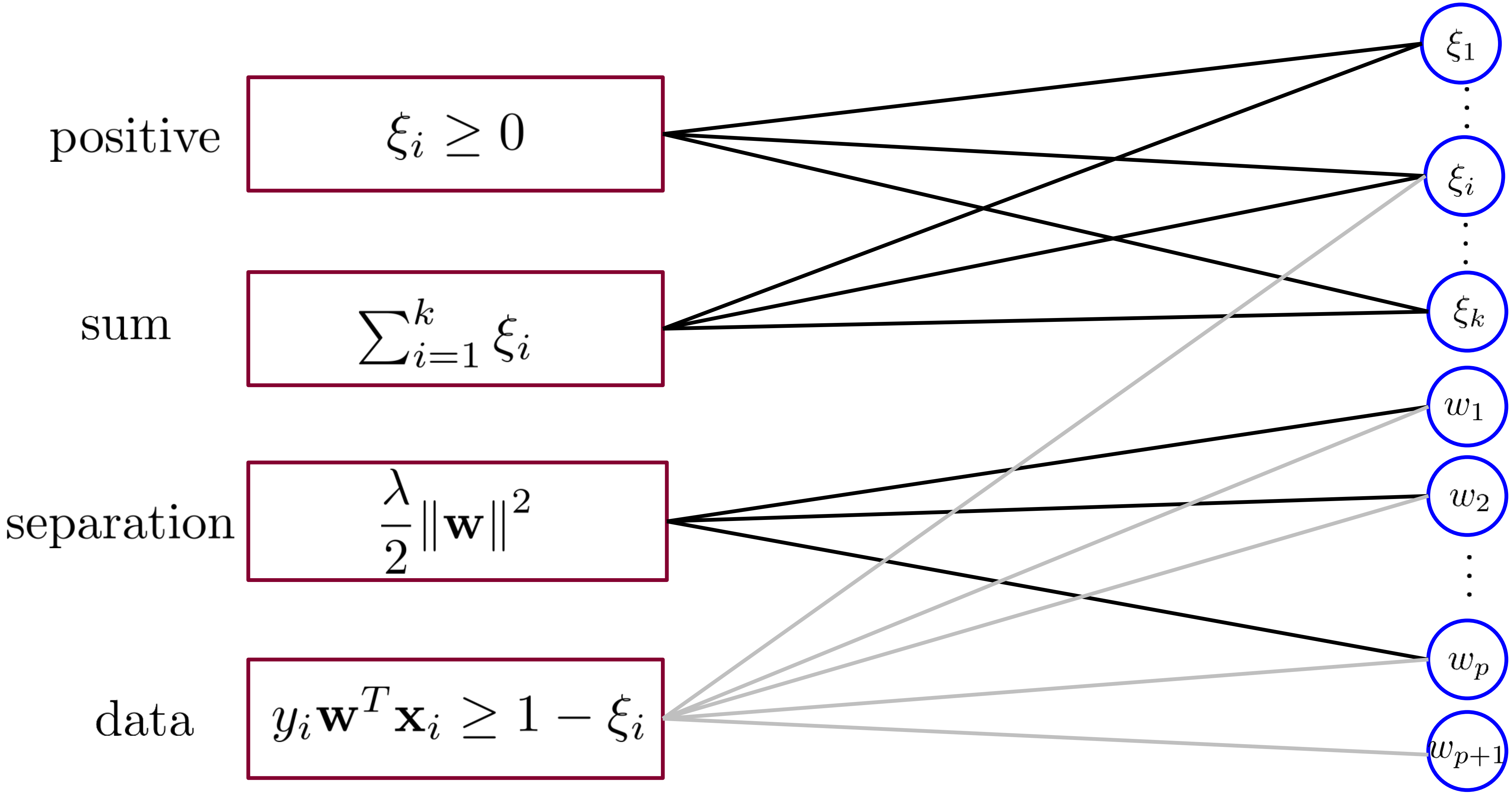
Support Vector Machine - ADMM



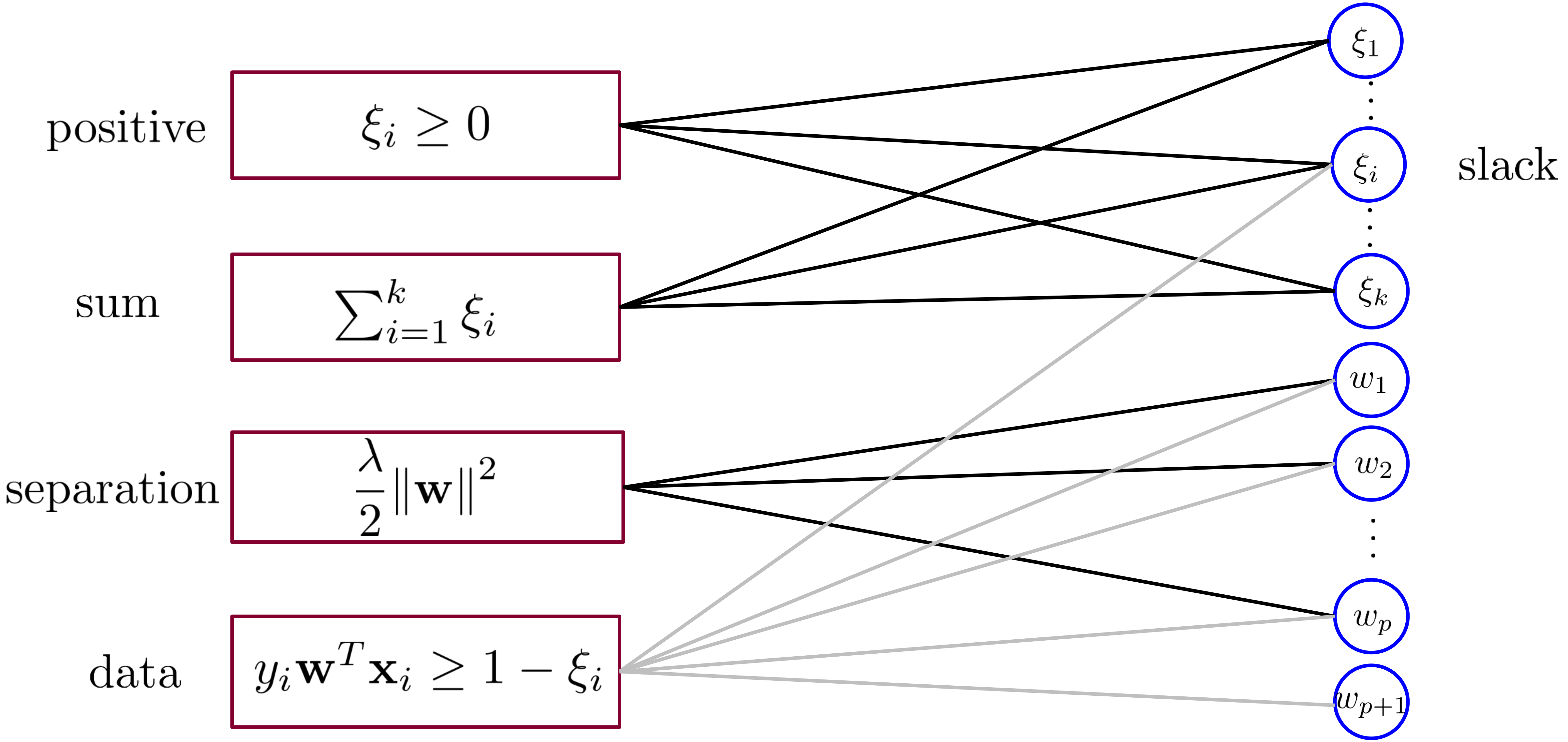
Support Vector Machine - ADMM



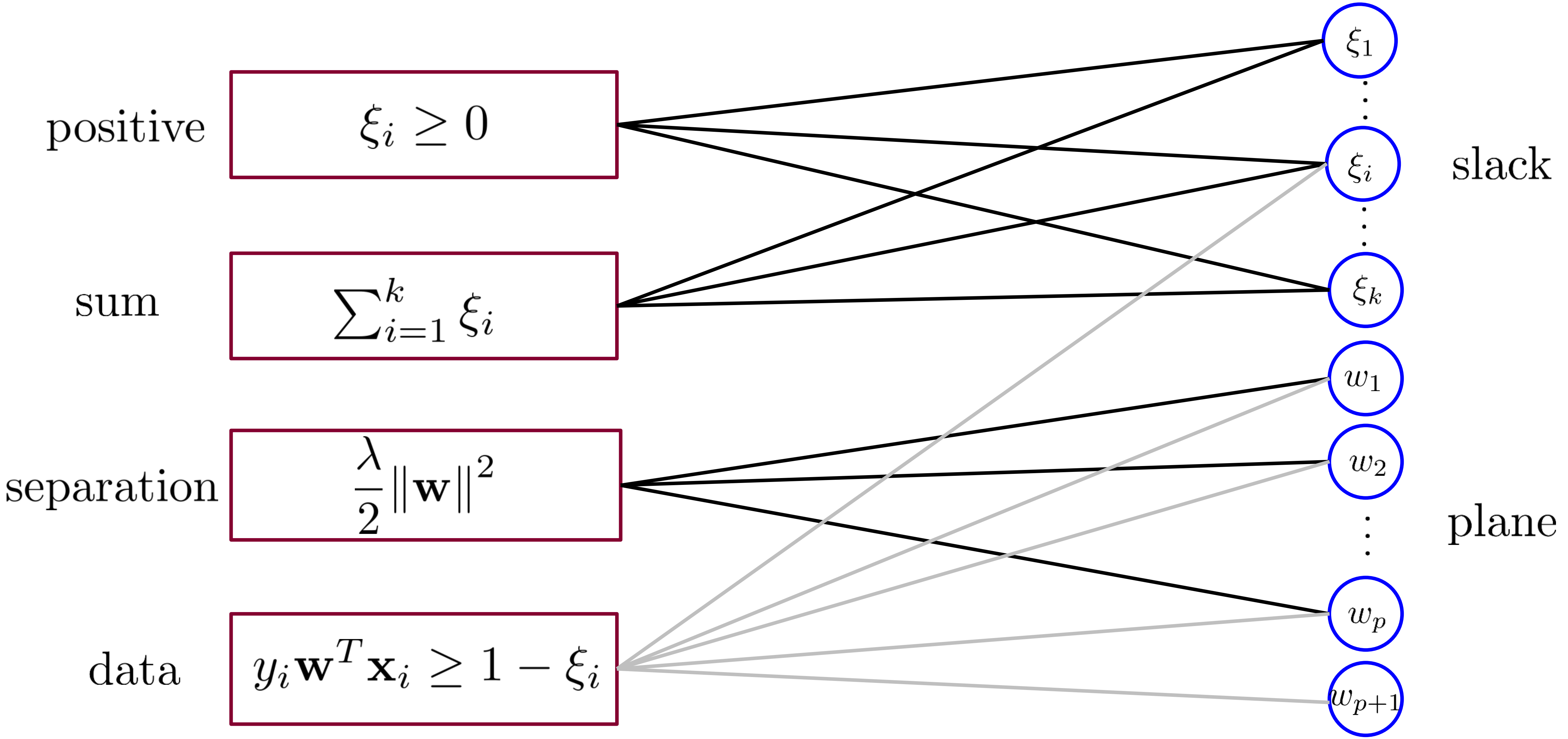
Support Vector Machine - ADMM



Support Vector Machine - ADMM



Support Vector Machine - ADMM



Support Vector Machine - Positive

$$\boxed{\xi_i \geq 0}$$

$$(x_1, \ldots, x_k) = \underset{s_1, \ldots, s_k}{\arg\min} \; \frac{\rho}{2}(s_1 - n_1)^2 + \ldots + \frac{\rho}{2}(s_k - n_k)^2$$
$$\text{subject to} \quad s_i \geq 0$$

$$\rho(s_i - n_i) = 0$$
$$\text{subject to} \quad s_i \geq 0$$

$$\longrightarrow \quad x_i = \max(0, n_i)$$

Support Vector Machine - Sum

$$\boxed{\sum_{i=1}^{k} \xi_i}$$

$$(x_1, \ldots, x_k) = \arg\min_{s_1,\ldots,s_k} \ (s_1 + \ldots + s_k) + \frac{\rho}{2}(s_1 - n_1)^2 + \ldots + \frac{\rho}{2}(s_k - n_k)^2$$

$$1 + \rho(s_i - n_i) = 0 \longrightarrow x_i = n_i - \frac{1}{\rho}$$

Support Vector Machine - Norm

$$\boxed{\frac{\lambda}{2}\|\mathbf{w}\|^2}$$

$$(x_1, \ldots, x_p) = \underset{s_1, \ldots, s_p}{\arg\min}\ \frac{\lambda}{2}({s_1}^2 + \ldots + {s_p}^2) + \frac{\rho}{2}(s_1 - n_1)^2 + \ldots + \frac{\rho}{2}(s_p - n_p)^2$$

$$\lambda s_i + \rho(s_i - n_i) = 0 \longrightarrow x_i = \frac{\rho n_i}{\lambda + \rho}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \;\; y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \quad y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

ξ_i

$$\text{subject to } \; y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\ \min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \quad y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

w_1

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \quad y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix}$$

$$w_{p+1} \leftarrow s_{p+2}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \quad y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix} \quad N = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_{p+2} \end{bmatrix}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \quad y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix} \quad N = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_{p+2} \end{bmatrix} \quad Y = \begin{bmatrix} 1 \\ y_i \mathbf{x}_i \end{bmatrix}$$

Support Vector Machine - Data

$$\boxed{y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i}$$

$$(x_1, \ldots, x_{p+2}) = \underset{s_1, \ldots, s_{p+2}}{\arg\min} \frac{\rho}{2}(s_1 - n_1)^2 + \frac{\rho}{2}(s_2 - n_2)^2 + \ldots + \frac{\rho}{2}(s_{p+2} - n_{p+2})^2$$

$$\text{subject to} \;\; y_i [s_{2:p+2}]^T \mathbf{x}_i \geq 1 - s_1$$

$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{p+2} \end{bmatrix} \quad N = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_{p+2} \end{bmatrix} \quad Y = \begin{bmatrix} 1 \\ y_i \mathbf{x}_i \end{bmatrix}$$

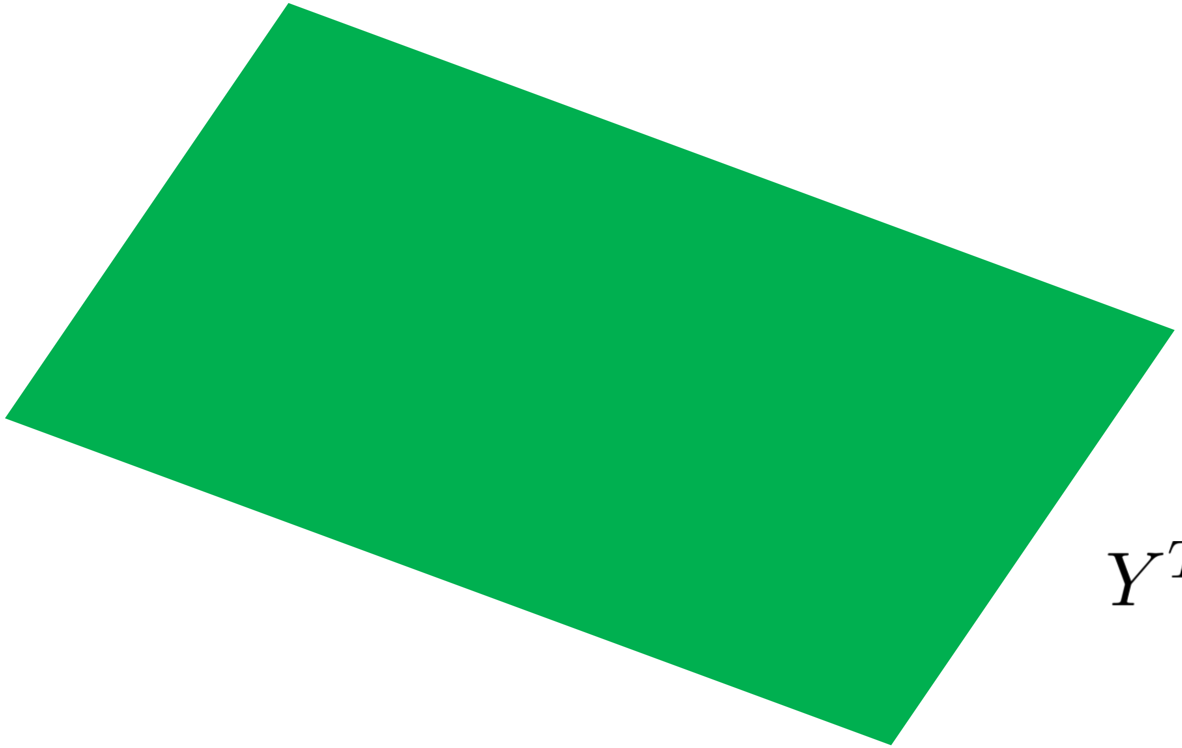
$$\boxed{\begin{aligned} X &= \underset{S}{\arg\min} \|S - N\|^2 \\ &\text{subject to} \;\; Y^T S \geq 1 \end{aligned}}$$

Support Vector Machine - Data

$$X = \arg\min_{S} \|S - N\|^2$$
$$\text{subject to } \; Y^T S \geq 1$$

Support Vector Machine - Data

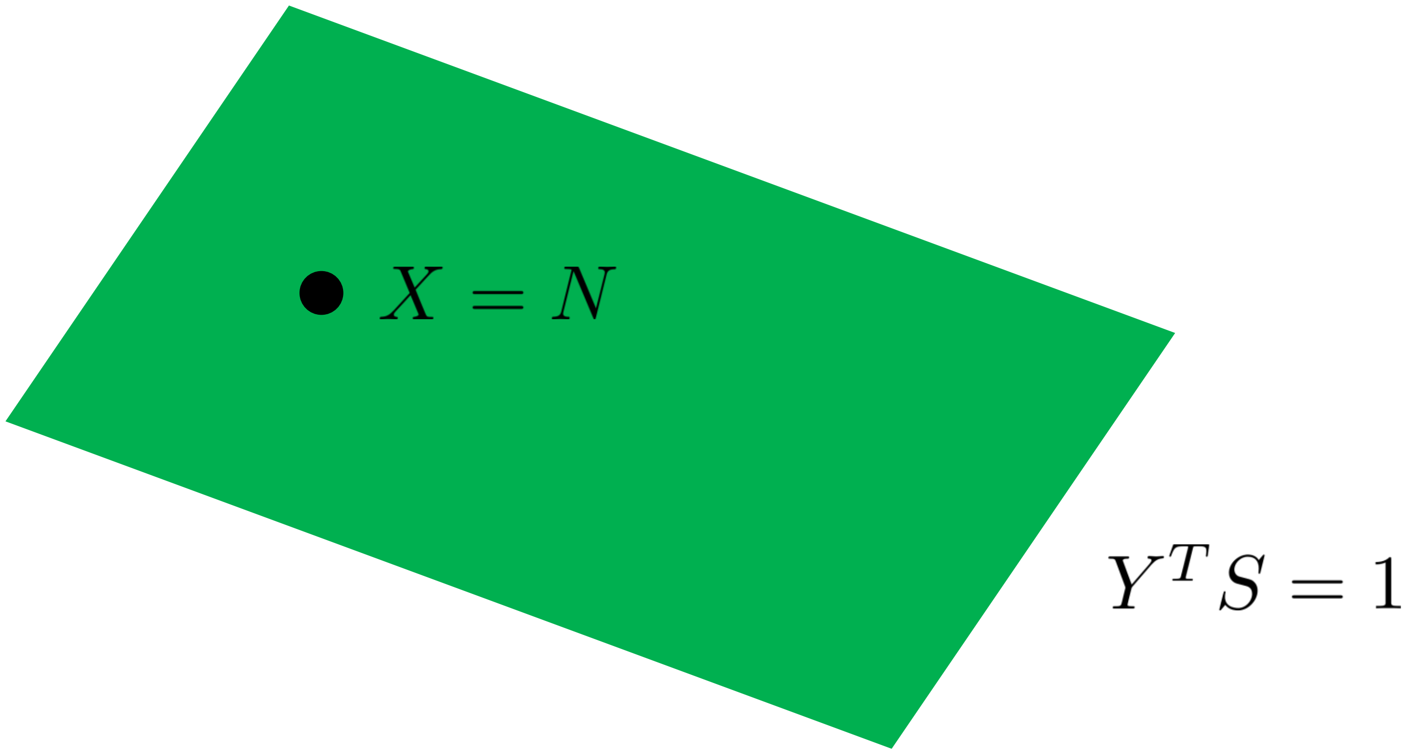
$$X = \arg\min_{S} \|S - N\|^2$$
$$\text{subject to} \quad Y^T S \geq 1$$



$$Y^T S = 1$$

Support Vector Machine - Data

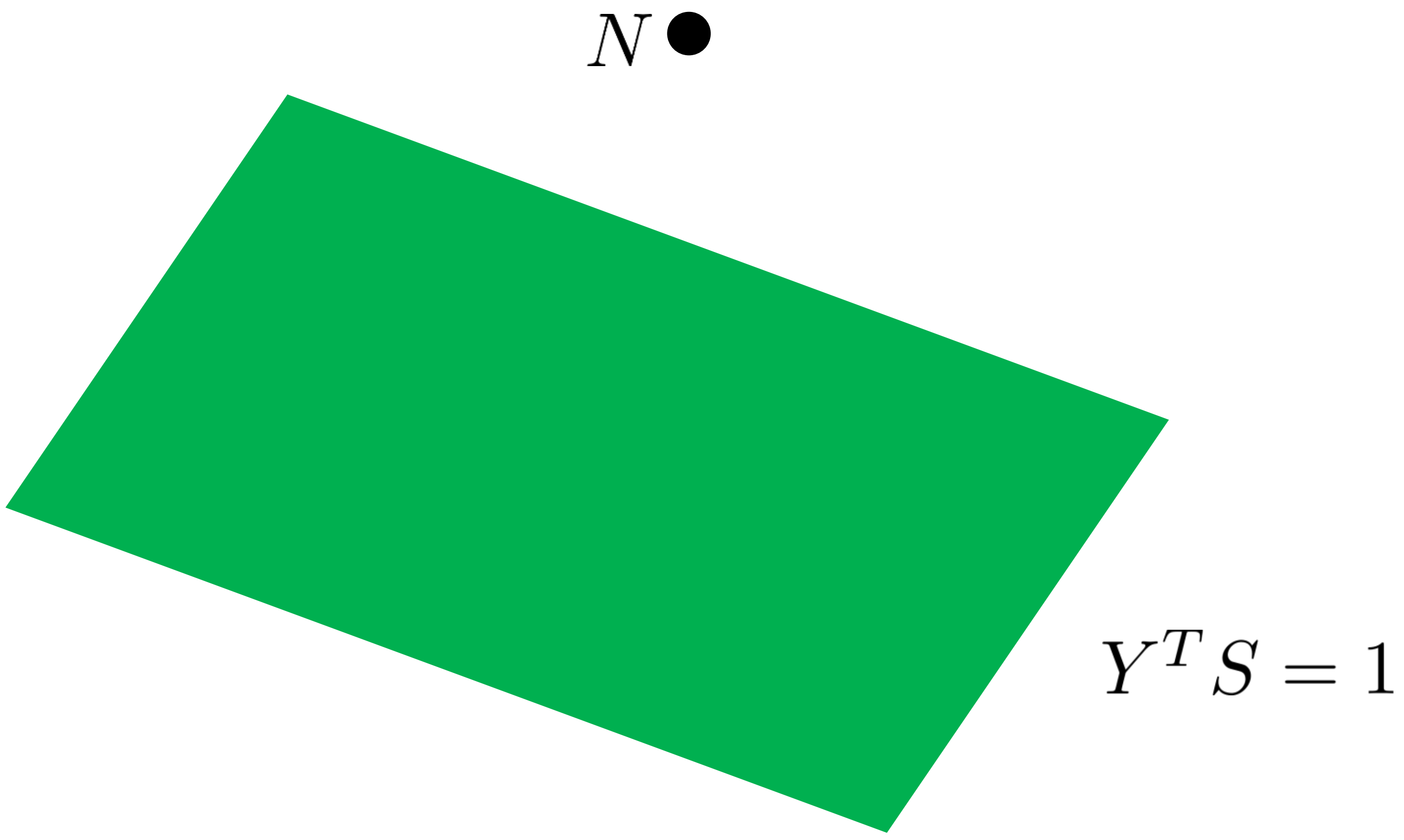
$$X = \arg\min_S \|S - N\|^2$$
$$\text{subject to } \; Y^T S \geq 1$$



Support Vector Machine - Data

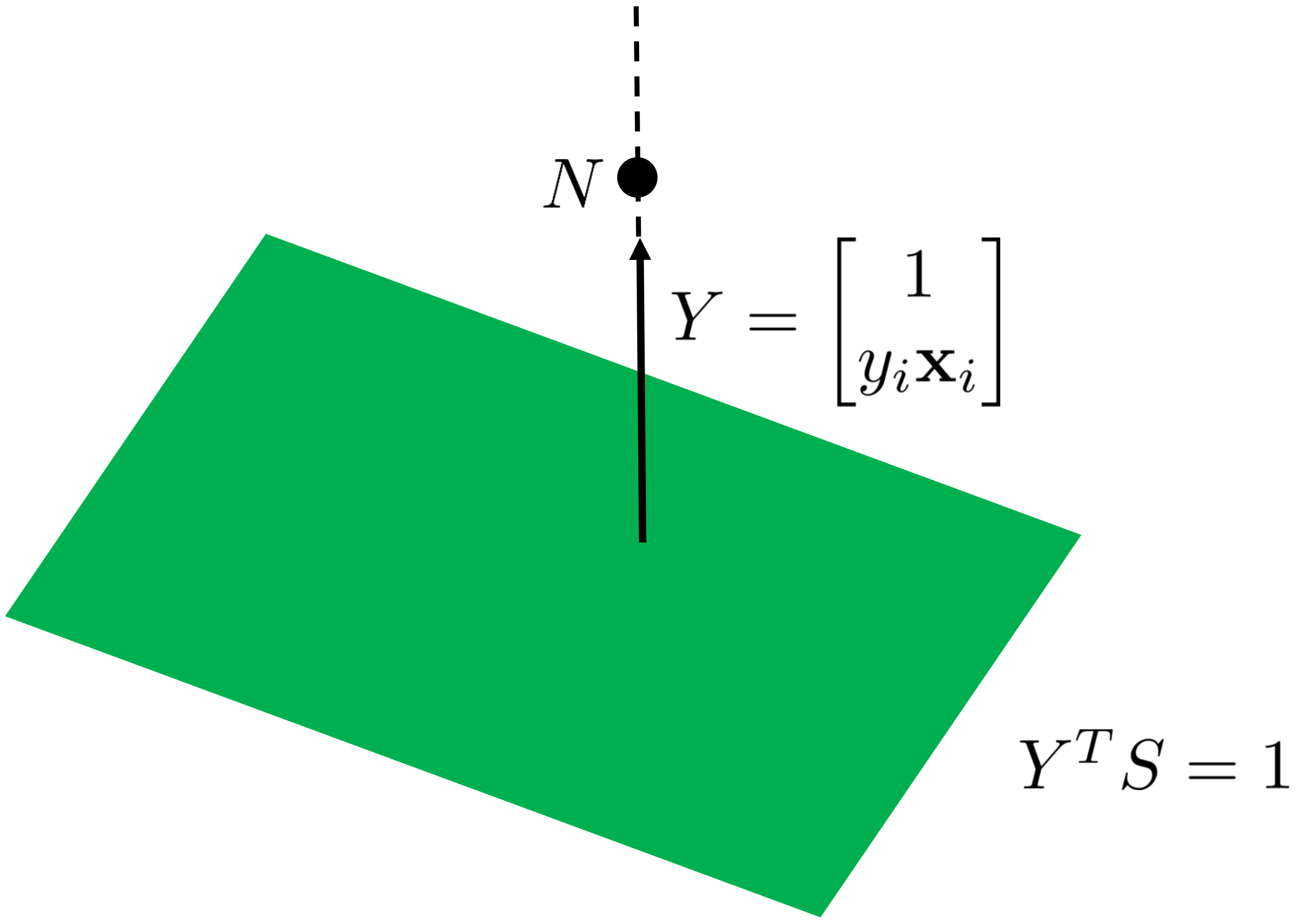
$$X = \arg\min_{S} \|S - N\|^2$$

$$\text{subject to} \quad Y^T S \geq 1$$



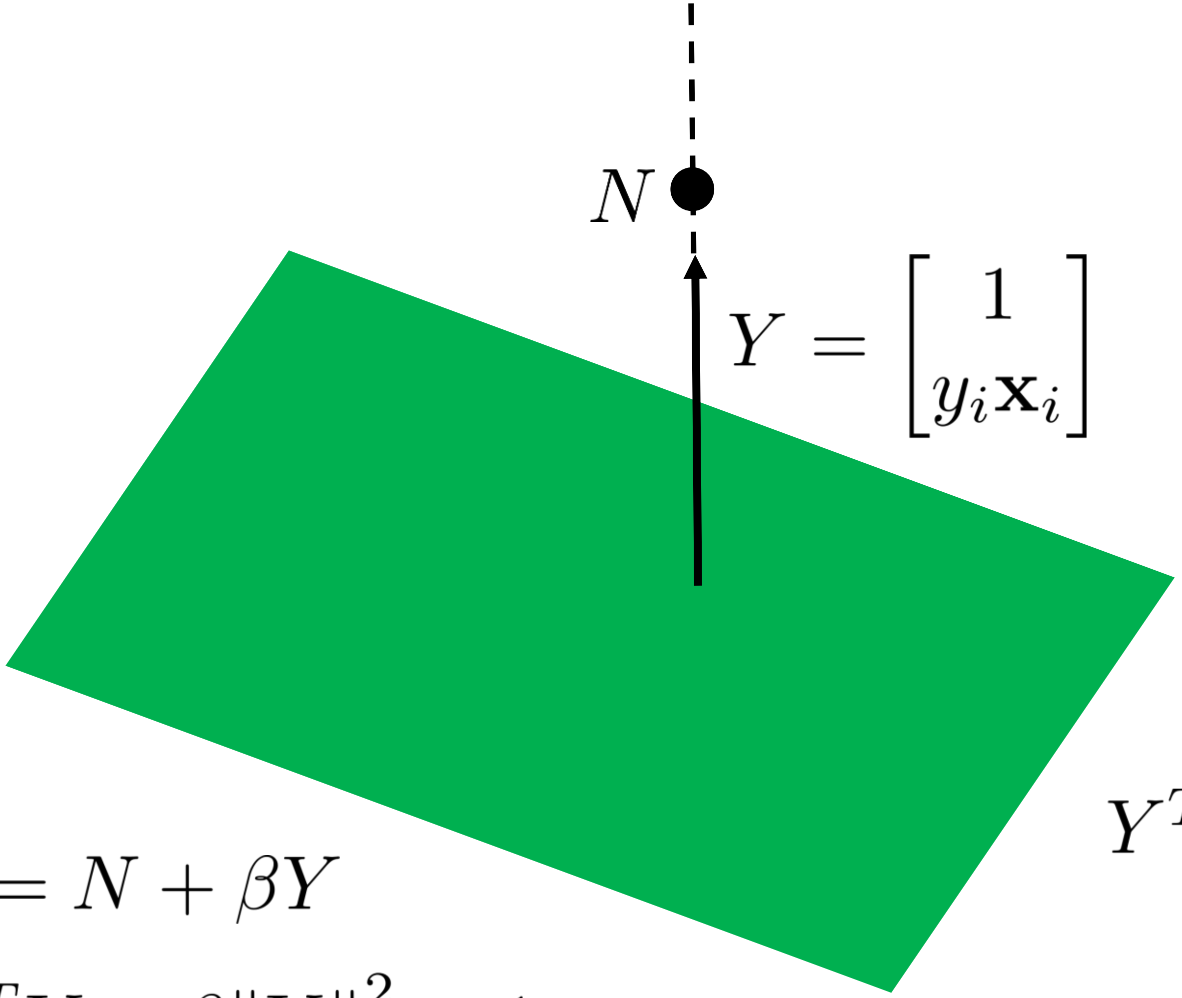
Support Vector Machine - Data

$$X = \arg\min_{S} \|S - N\|^2$$
$$\text{subject to} \quad Y^T S \geq 1$$



Support Vector Machine - Data

$$X = \arg\min_{S} \|S - N\|^2$$
$$\text{subject to} \quad Y^T S \geq 1$$



$$Y = \begin{bmatrix} 1 \\ y_i \mathbf{x}_i \end{bmatrix}$$

$$Y^T S = 1$$

$$S = N + \beta Y$$

$$N^T Y + \beta \|Y\|^2 = 1$$

Support Vector Machine - pos

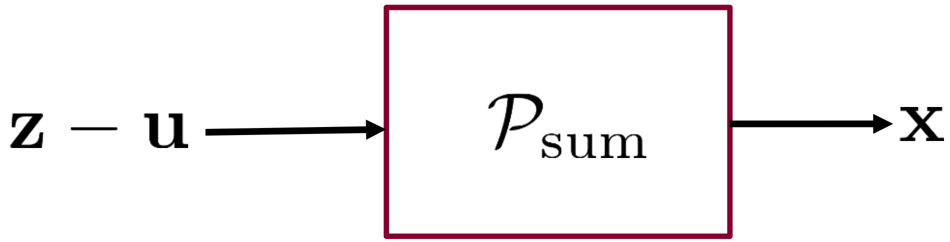
$$\mathbf{z} - \mathbf{u} \longrightarrow \boxed{\mathcal{P}_{\text{pos}}} \longrightarrow \mathbf{x}$$

```
function [X] = P_pos(Z_minus_U)

    X = max(Z_minus_U,0);

end
```

Support Vector Machine - pos

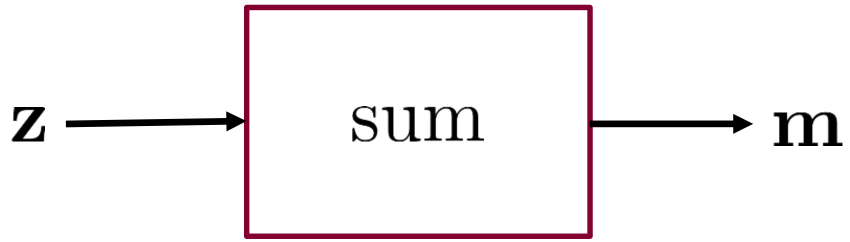

```
function [M, new_U] = F_pos(Z , U)
    % Compute internal updates
    X = P_pos( Z - U );

    new_U = U + (X  -  Z);

    % Compute outgoing messages
    M = new_U + X;

end
```

Support Vector Machine - sum




```
function [X] = P_sum(Z_minus_U)

    global rho

    X =  Z_minus_U  - (1 / rho);

end
```

Support Vector Machine - pos




```
function [M, new_U] = F_pos(Z , U)
    % Compute internal updates
    X = P_pos( Z - U );

    new_U = U + (X  -  Z);

    % Compute outgoing messages
    M = new_U + X;

end
```

Support Vector Machine - separation

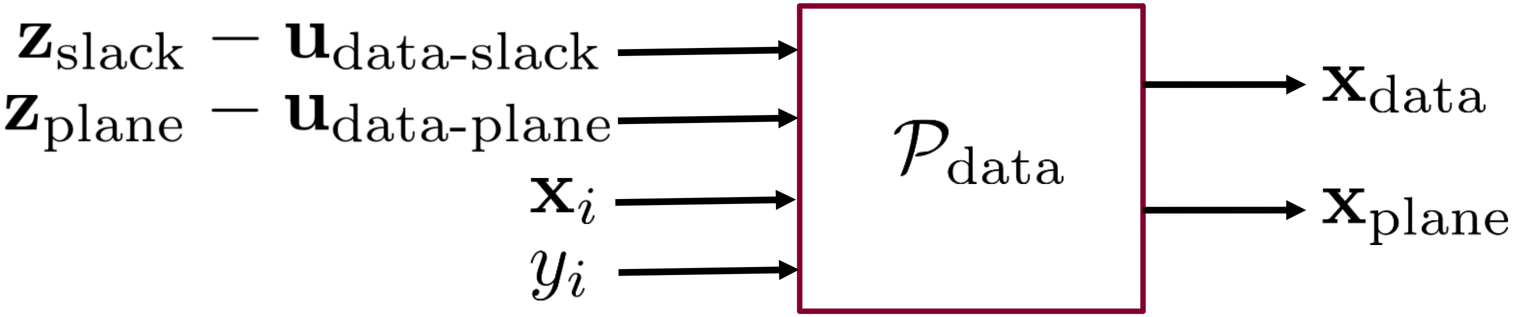
$$\mathbf{z} - \mathbf{u} \longrightarrow \boxed{\mathcal{P}_{\text{separation}}} \longrightarrow \mathbf{x}$$

```
function [X] = P_separation(Z_minus_U)

        global rho
        global lambda

    X = (rho/(lambda + rho)) * Z_minus_U ;

end
```

Support Vector Machine - separation

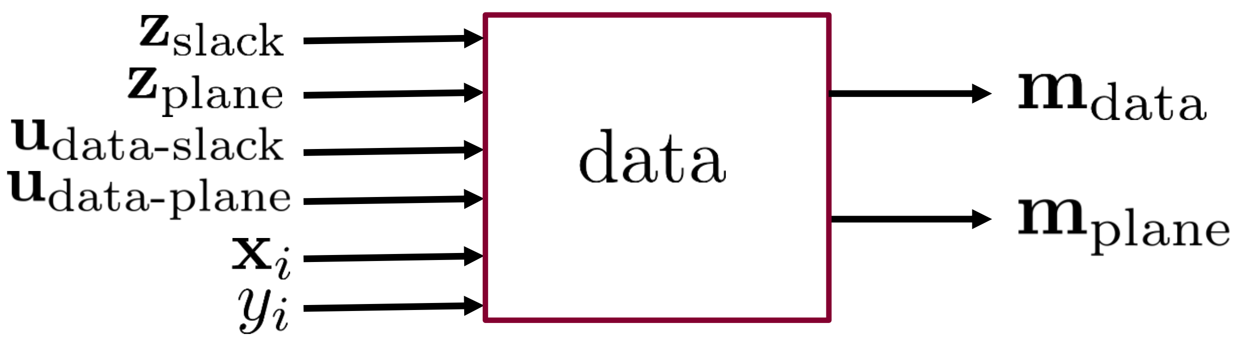

```
function [M, new_U] = F_separation(Z, U)

    % Compute internal updates
    X = P_separation( Z - U );

    new_U = U + (X  -  Z);

    % Compute outgoing messages
    M = new_U + X;

end
```

Support Vector Machine - data




```
function [X_data, X_plane] = P_data(Z_slack_minus_U_data_slack,Z_plane_minus_U_data_plane,x_i,y_i)

if (y_i*Z_plane_minus_U_data_plane'*x_i >= 1 - Z_slack_minus_U_data_slack)

    X_data = Z_slack_minus_U_data_slack; X_plane = Z_plane_minus_U_data_plane;
else
    beta = ((1-[1;y_i*x_i]'*[Z_slack_minus_U_data_slack;Z_plane_minus_U_data_plane])/([1;y_i.*x_
         i]'*[1;y_i*x_i]));

    X_data = Z_slack_minus_U_data_slack + beta;
    X_plane = Z_plane_minus_U_data_plane + beta*y_i*x_i;

end
```

Support Vector Machine - data

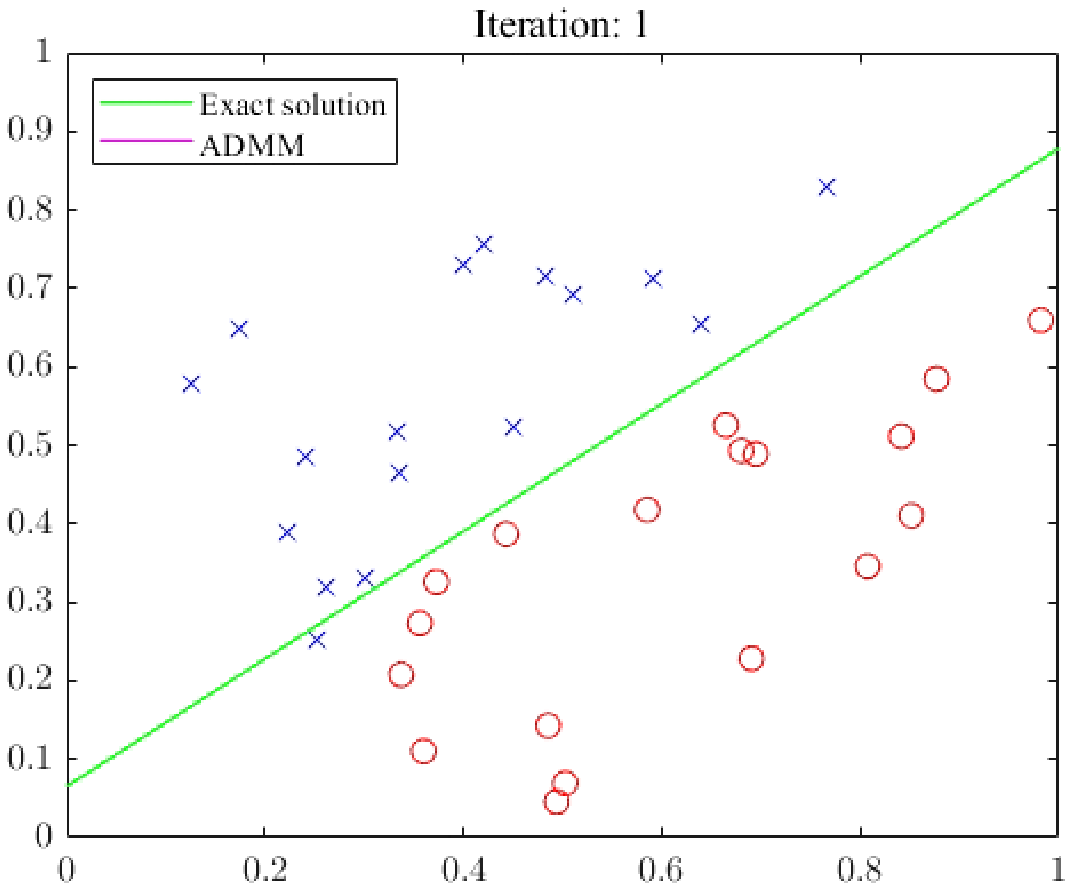



```
function [M_data,M_plane, new_U_data,new_U_plane] = F_data(Z_slack, Z_plane,U_data_slack,U_data_plane,
x_i, y_i)

    % Compute internal updates
    [X_data, X_plane] = P_data( Z_slack - U_data_slack , Z_plane - U_data_plane ,  x_i, y_i);

    new_U_data = U_data_slack + (X_data  -  Z_slack);
    new_U_plane = U_data_plane + (X_plane  -  Z_plane);

    % Compute outgoing messages
    M_plane = new_U_plane + X_plane;
    M_data = new_U_data + X_data;

end
```

```matlab
n = 10; p = 4000; y = sign(randn(n,1)); x = randn(p,n); x = [x;ones(1,n)];% Create random data
global rho; rho = 1; global lambda; lambda = 0.1; %Initialization
U_pos  = randn(n,1); U_sum  = randn(n,1); U_norm  = randn(p,1); U_data  = randn(p+2,n);
M_pos  = randn(n,1); M_sum  = randn(n,1); M_norm  = randn(p,1); M_data  = randn(p+2,n);
Z_slack = randn(n,1); Z_plane = randn(p+1,1);
%ADMM iterations
for t = 1:1000
    [M_pos, U_pos] = F_pos(Z_slack , U_pos); % POSITIVE SLACK
    [M_sum, U_sum] = F_sum(Z_slack , U_sum); % SLACK SUM COST
    [M_norm, U_norm] = F_separation(Z_plane(1:p) , U_norm); % SEPARATION COST
    for i = 1:n  % DATA CONSTRAINT
       [M_data(1,i), M_data(2:end,i),U_data(1,i),U_data(2:end,i)] = F_data( Z_slack(i),Z_plane,
        U_data(1,i),U_data(2:end,i),x(:,i),y(i));
    end
    % Z updates
    Z_slack = M_pos + M_sum;
    for i = 1:n
        Z_slack(i) = Z_slack(i) + M_data(1,i);
    end
    Z_slack = Z_slack / 3; Z_plane(1:p) = M_norm;
    for i = 1:p
        for j = 1:n
            Z_plane(i) = Z_plane(i) + M_data(i+1,j);
        end
    end
    Z_plane(1:p) = Z_plane(1:p) / (n+1);
    for i = 1:n
        Z_plane(p+1) = Z_plane(p+1) + M_data(p+2,i);
    end
    Z_plane(p+1) = Z_plane(p+1)/n;
end
```

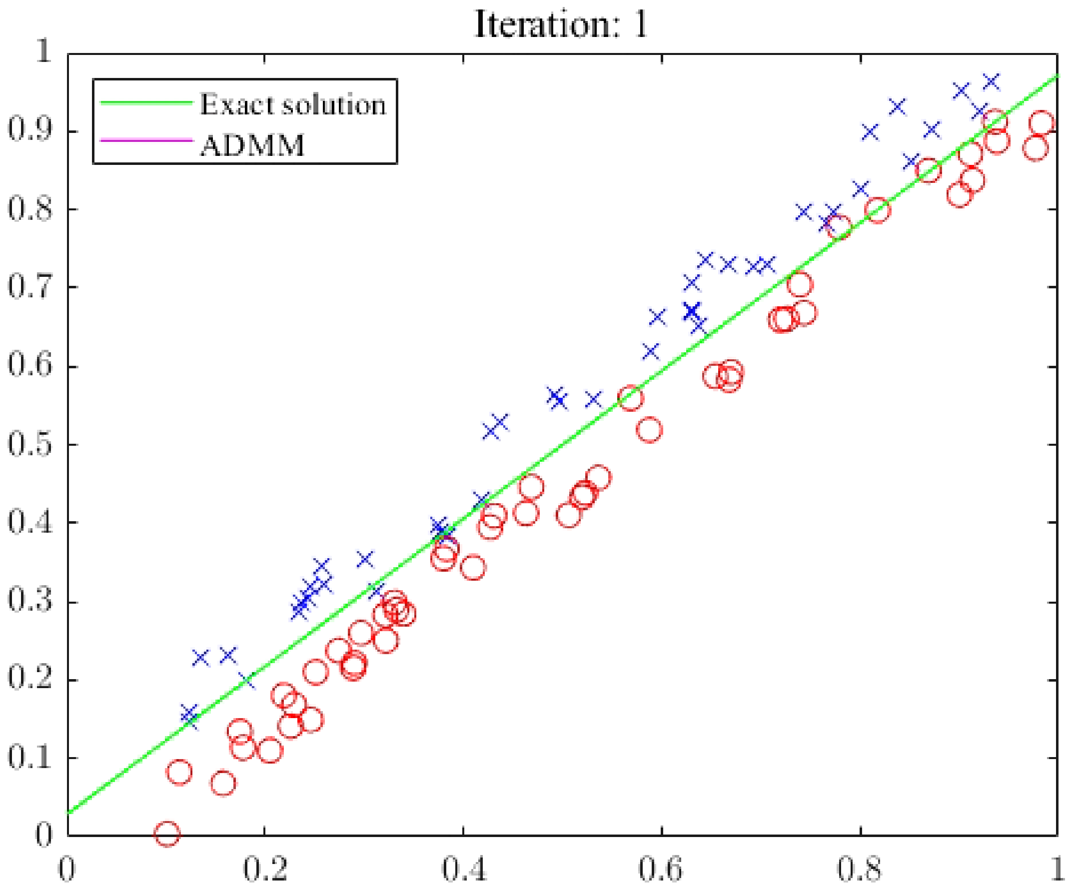
Support Vector Machine

Support Vector Machine



Support Vector Machine

Support Vector Machine



Please cite this tutorial by citing:


```
@article{safavi2018admmtutorial,
  title={Networks and large scale optimization: a short, hands-on, tutorial on ADMM},
  note={Open Data Science Conference},
  author={Safavi, Sam and Bento, Jos{\’e}},
  year={2018}
}

@inproceedings{hao2016testing,
  title={Testing fine-grained parallelism for the ADMM on a factor-graph},
  author={Hao, Ning and Oghbaee, AmirReza and Rostami, Mohammad and Derbinsky, Nate and Bento, Jos{\'e}},
  booktitle={Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International},
  pages={835--844},
  year={2016},
  organization={IEEE}
}

@inproceedings{francca2016explicit,
  title={An explicit rate bound for over-relaxed ADMM},
  author={Fran{\c{c}}a, Guilherme and Bento, Jos{\'e}},
  booktitle={Information Theory (ISIT), 2016 IEEE International Symposium on},
  pages={2104--2108},
  year={2016},
  organization={IEEE}
}

@article{derbinsky2013improved,
  title={An improved three-weight message-passing algorithm},
  author={Derbinsky, Nate and Bento, Jos{\'e} and Elser, Veit and Yedidia, Jonathan S},
  journal={arXiv preprint arXiv:1305.1961},
  year={2013}
}

@article{bento2018complexity,
  title={On the Complexity of the Weighted Fussed Lasso},
  author={Bento, Jos{\’e} and Furmaniak, Ralph and Ray, Surjyendu},
  journal={arXiv preprint arXiv:1801.04987},
  year={2018}
}
```


Code, link to slides and video available at

https://github.com/bentoayr/ADMM-tutorial

or

http://jbento.info