Direct Methods for Sparse Linear Systems:

MATLAB sparse backslash

Tim Davis

davis@cise.ufl.edu

University of Florida

So what is a sparse matrix ... ?

So what is a sparse matrix ... ?

a matrix *"... that allows special techniques to take advantage of the large number of zero elements"* (Wilkinson)

sparse matrices arise in a wide range of applications ...

Sparse matrices arise in ...

computational fluid dynamics, finite-element methods, statistics, time/frequency domain circuit simulation, dynamic and static modeling of chemical processes, cryptography, magneto-hydrodynamics, electrical power systems, differential equations, quantum mechanics, structural mechanics (buildings, ships, aircraft, human body parts...), heat transfer, MRI reconstructions, vibroacoustics, linear and non-linear optimization, financial portfolios, semiconductor process simulation, economic modeling, oil reservoir modeling, astrophysics, crack propagation, Google page rank, 3D computer vision, cell phone tower placement, tomography, multibody simulation, model reduction, nano-technology, acoustic radiation, density functional theory, quadratic assignment, elastic properties of crystals, natural language processing, DNA electrophoresis, ...

Outline

- Sparse matrix algorithms: numerics plus graph theory

Outline

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up

Outline

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
  - L, x, b are all sparse
  - must know nonzero pattern of x to compute x efficiently
  - time: O(flops)

Outline

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
  - left-looking, partial pivoting
  - fill-reducing column ordering
  - relies on `x=L\b`, where `L`, `x`, `b` are all sparse
  - time: O(n+flops)

Outline

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)
  - up-looking and left-looking
  - fill-reducing symmetric ordering
  - relies on `x=L\b`, where `L`, `x`, b are all sparse
  - time: O(flops)

Outline

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)
- Supernodal and multifrontal methods (`x=A\b`)
  - cache-friendly dense matrix kernels (BLAS)
  - supernodal (left-looking)
  - multifrontal (right-looking)

... next: sparse matrix data structures

Sparse data structures

- compressed sparse column format (... many others)

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ...   Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

```
Ap: [0,                 3,                 6,         8,        10]
Ai: [0,  1,   3,   1,   2,   3,   0,   2,   1,   3   ]
Ax: [4.5,3.1,3.5,2.9,1.7,0.4,3.2,3.0,0.9,1.0]
```

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

```
Ap: [0,          3,                                ]
Ai: [0,   1,   3,                               ]
Ax: [4.5,3.1,3.5,                               ]
```

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

```
Ap: [            3,          6,            ]
Ai: [            1,  2,  3,              ]
Ax: [            2.9,1.7,0.4,            ]
```

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

```
Ap: [                             6,       8,          ]
Ai: [                             0,   2,            ]
Ax: [                             3.2,3.0,           ]
```

Sparse data structures

- compressed sparse column format
- column j is `Ai[Ap[j] ... Ap[j+1]-1]`, ditto in `Ax`
- Thus, `A(:,j)` is easy in MATLAB; `A(i,:)` hard

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix}$$

```
Ap: [                                         8,       10]
Ai: [                                         1,   3  ]
Ax: [                                         0.9,1.0]
```

Sparse lower triangular solve, `x=L\b`

Sparse lower triangular solve, x=L\b

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

Sparse lower triangular solve, x=L\b

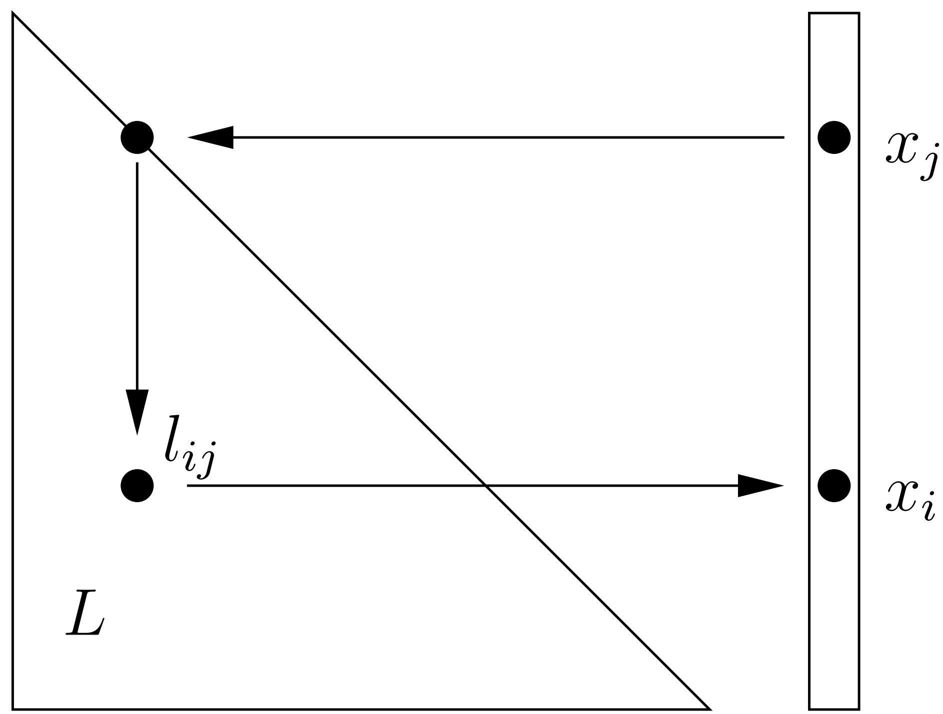
```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

- O(n+flops) time too high
- the problem:
  ```
  for j=1:n
    if (x(j) ≠ 0)
  ```
- need pattern of x before computing it

Sparse lower triangular solve, `x=L\b`

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

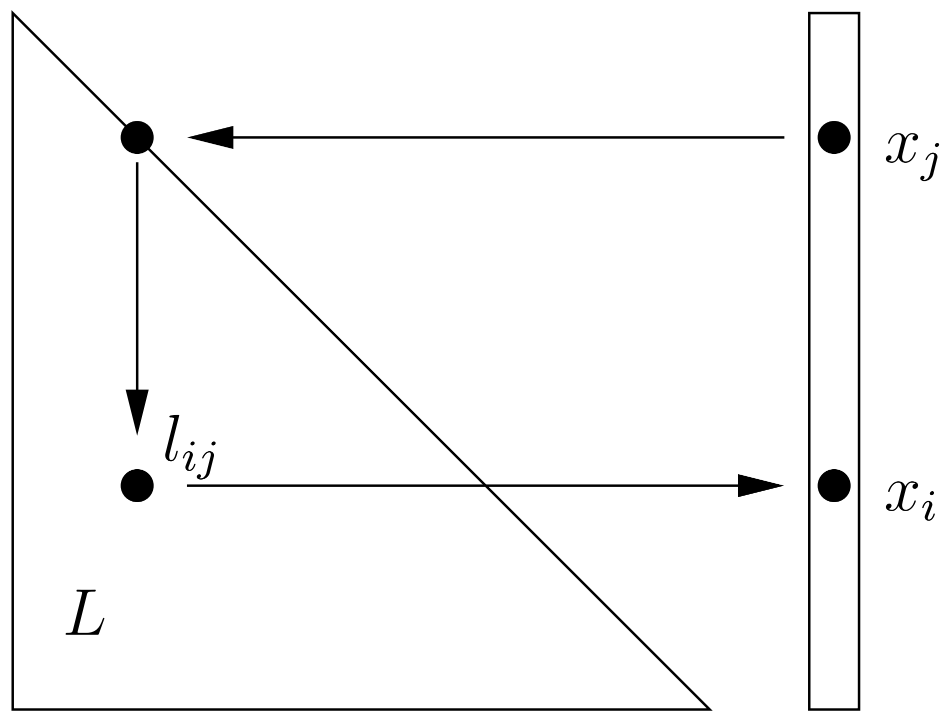
- $b_i \neq 0 \Rightarrow x_i \neq 0$



Sparse lower triangular solve, `x=L\b`

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

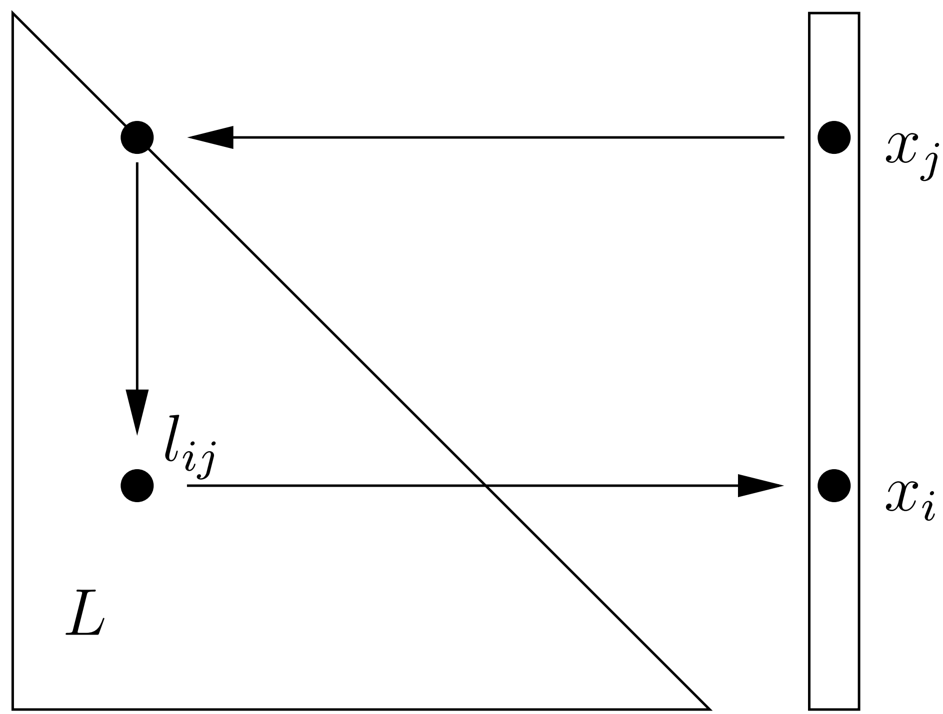
- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$



Sparse lower triangular solve, `x=L\b`

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

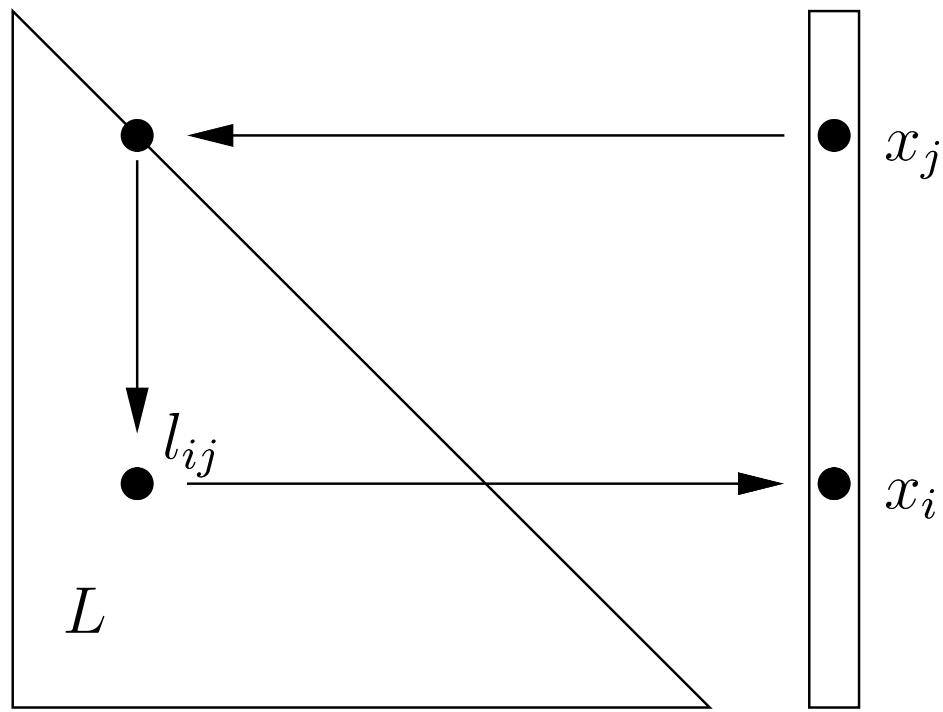
- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$
- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$



Sparse lower triangular solve, `x=L\b`

```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

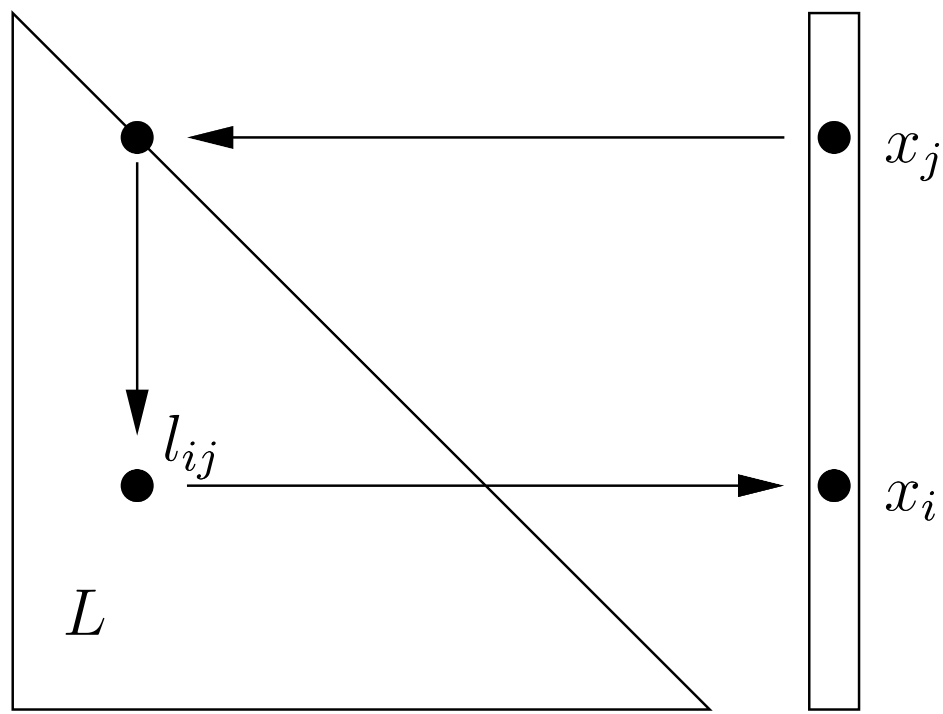
- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$
- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \,|\, b_i \neq 0\}$ and $\mathcal{X} = \{i \,|\, x_i \neq 0\}$



Sparse lower triangular solve, `x=L\b`

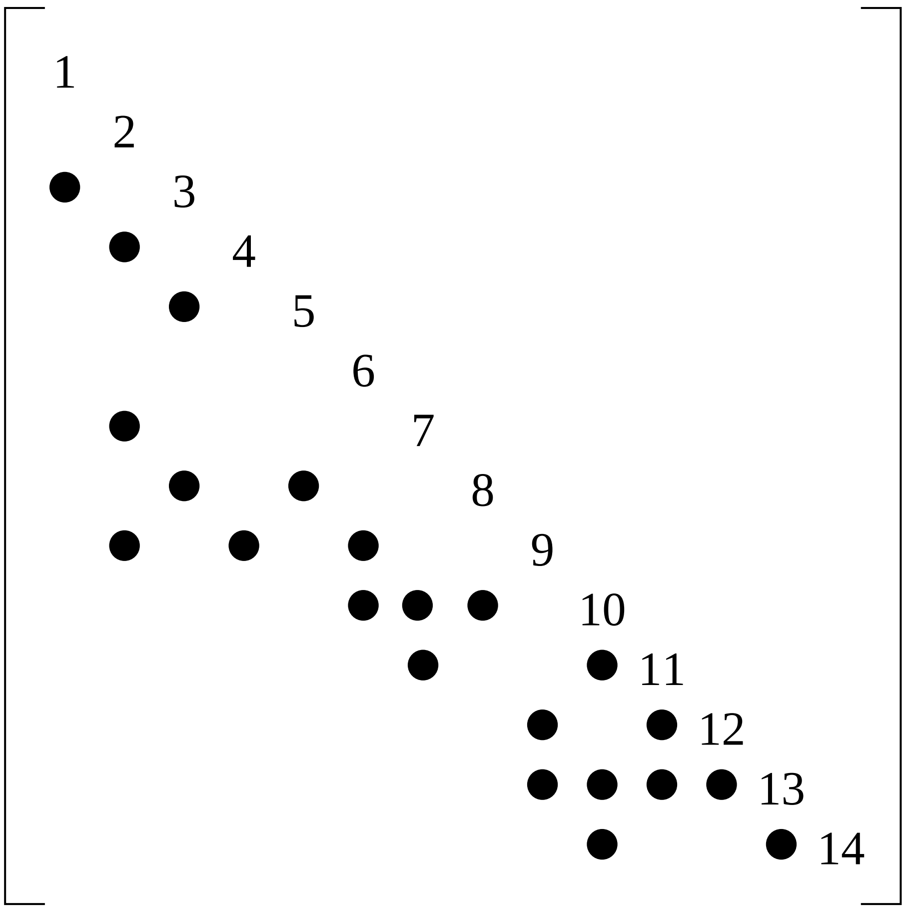
```
x = b
for j = 1:n
    if (x(j) ≠ 0)
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
    end
end
```

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$
- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \mid b_i \neq 0\}$ and $\mathcal{X} = \{i \mid x_i \neq 0\}$
- then $\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

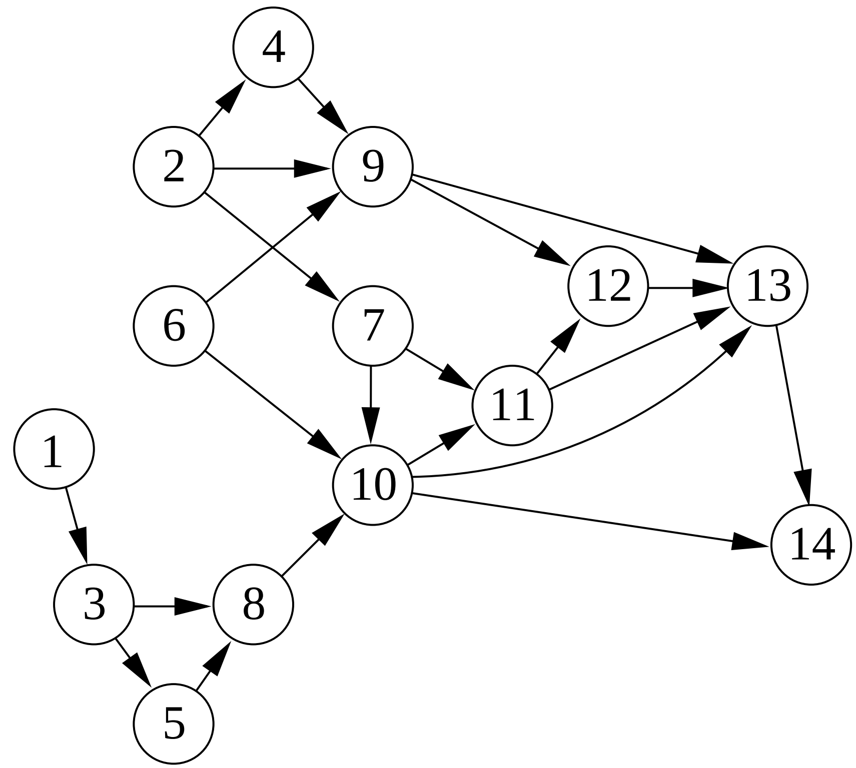


Sparse lower triangular solve, x=L\b

Lower triangular matrix L

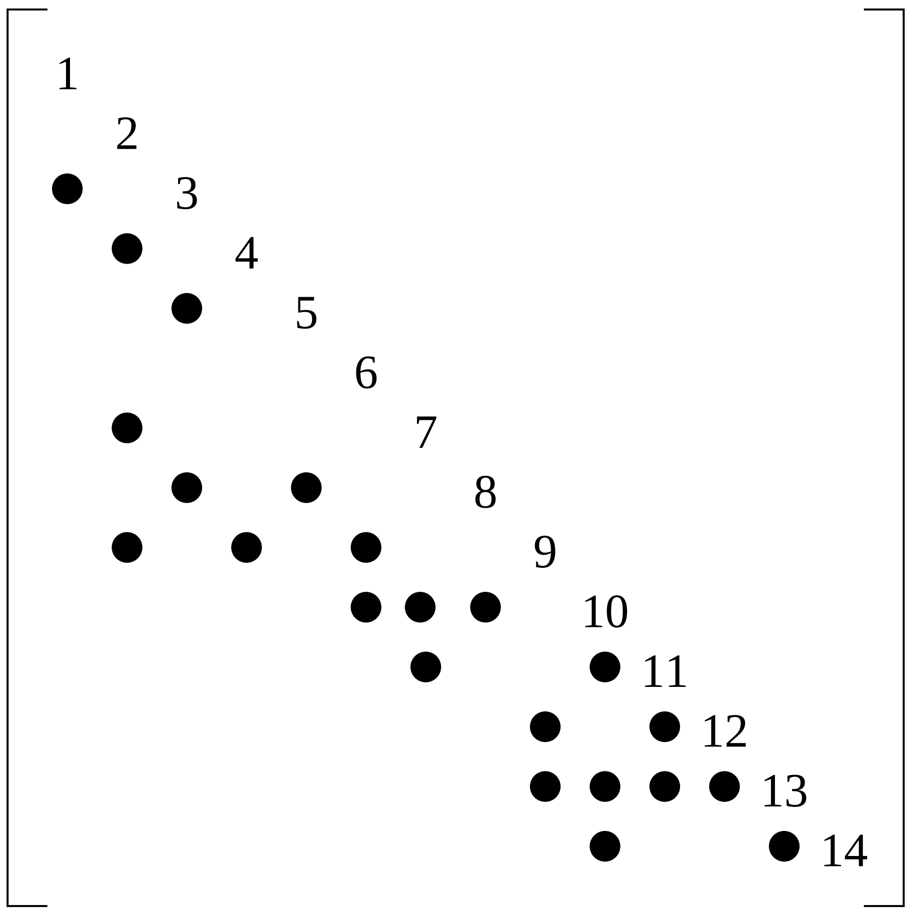


Graph G_L

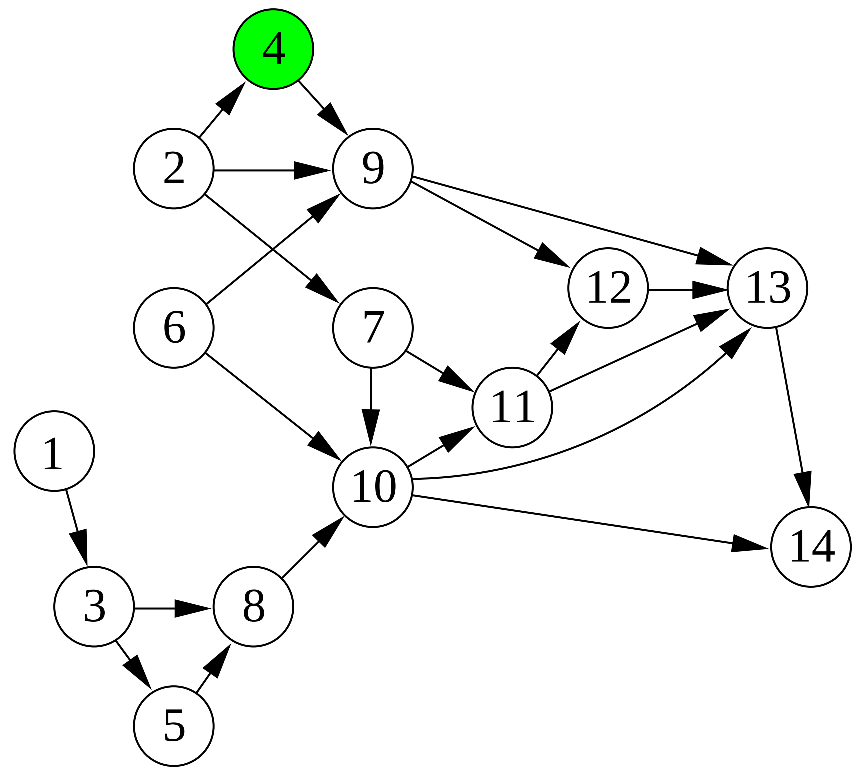


Sparse lower triangular solve, x=L\b

Lower triangular matrix L



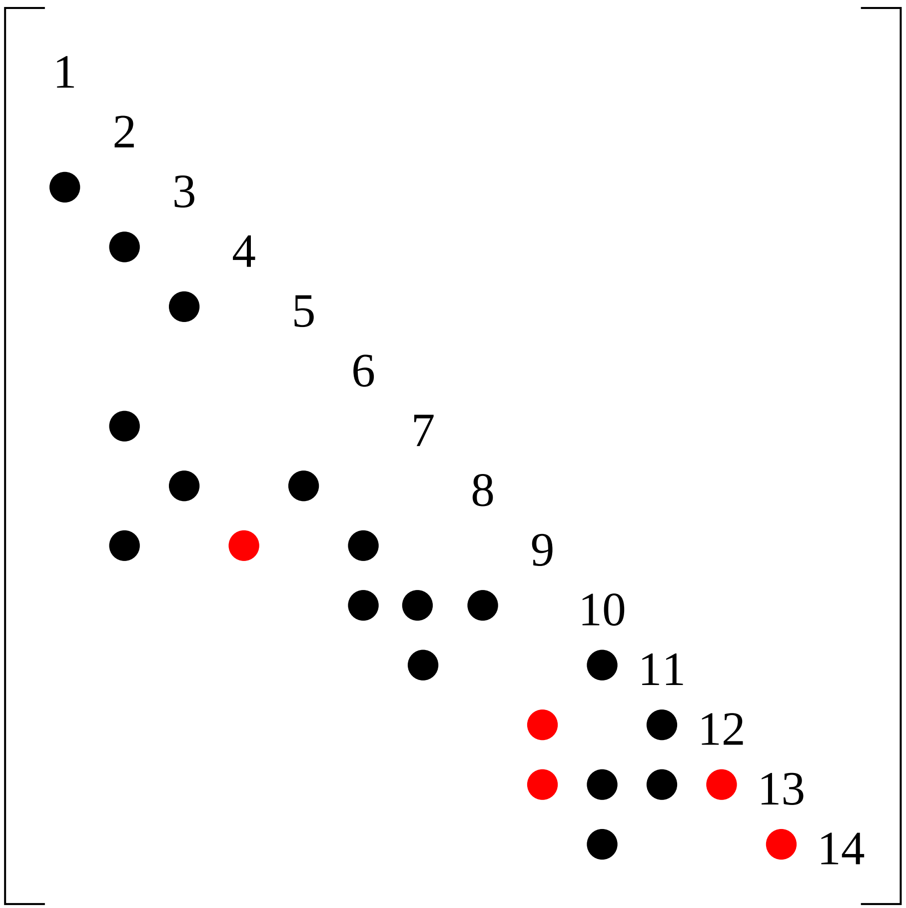
Graph G_L



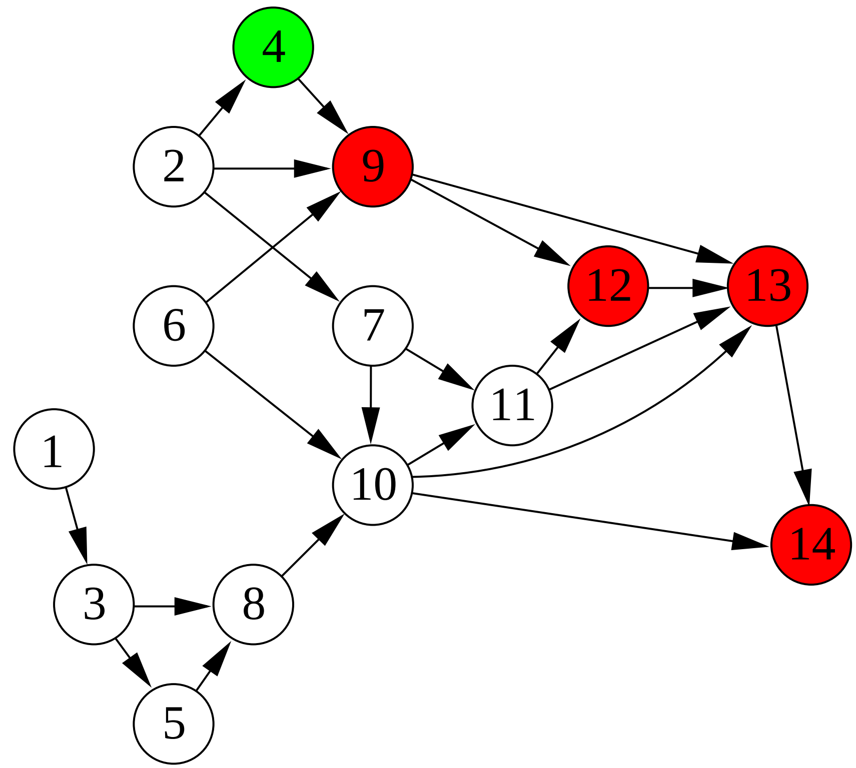
If $\mathcal{B} = \{4\}$

Sparse lower triangular solve, x=L\b

Lower triangular matrix L



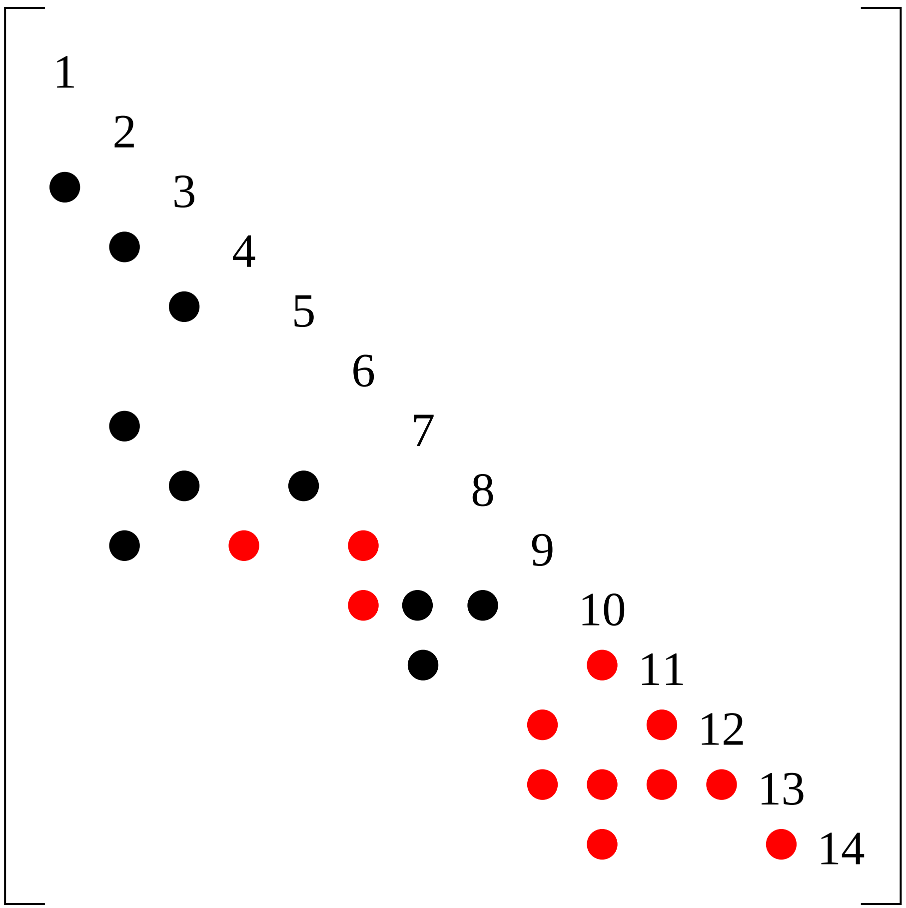
Graph G_L



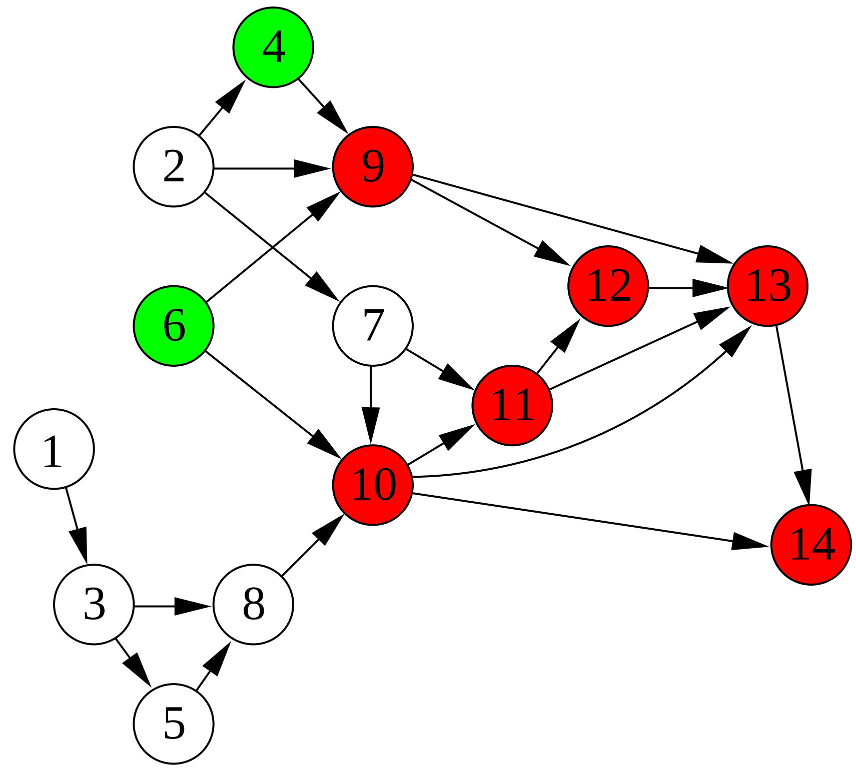
If $\mathcal{B} = \{4\}$
then $\mathcal{X} = \{4, 9, 12, 13, 14\}$

Sparse lower triangular solve, x=L\b

Lower triangular matrix L



Graph G_L



If $\mathcal{B} = \{4, 6\}$
then $\mathcal{X} = \{6, 10, 11, 4, 9, 12, 13, 14\}$

Sparse lower triangular solve, `x=L\b`

```
function x = lsolve(L,b)
    x = b
    for j = 1:n
        if (x(j) ≠ 0)
            x(j+1:n) = x(j+1:n) - L(j+1:n,j)*x(j)
```

Time: $O(n +$ flops$)$, need $\mathcal{X}$ to get $O($flops$)$

Sparse lower triangular solve, `x=L\b`

function `x = lsolve(L,b)`

$\mathcal{X} = \text{Reach}(L, \mathcal{B})$

```
x = b
```

for each j in $\mathcal{X}$

```
    x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
```

Sparse lower triangular solve, `x=L\b`

function `x = lsolve(L,b)`

```
    X = Reach(L, B)
    x = b
    for each j in X
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)
```

function $\mathcal{X} = \text{Reach}(\mathtt{L}, \mathcal{B})$

```
    for each i in B do
        if (node i is unmarked) dfs(i)
```

function dfs(j)

```
    mark node j
    for each i in L_j do
        if (node i is unmarked) dfs(i)
    push j onto stack for X
```

Sparse lower triangular solve, `x=L\b`

```
function x = lsolve(L,b)
    𝒳 = Reach(L,ℬ)
    x = b
    for each j in 𝒳
        x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)

function 𝒳 = Reach(L,ℬ)
    for each i in ℬ do
        if (node i is unmarked) dfs(i)

function dfs(j)
    mark node j
    for each i in ℒ_j do
        if (node i is unmarked) dfs(i)
    push j onto stack for 𝒳
```

$\boxed{\text{Total time: } O(\text{flops})}$

Sparse lower triangular solve, `x=L\b`

function `x = lsolve(L,b)`
    $\mathcal{X} =$ Reach$(L, \mathcal{B})$
    `x = b`
    for each j in $\mathcal{X}$
        `x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)`

function $\mathcal{X} =$ Reach$(\mathtt{L}, \mathcal{B})$
    for each i in $\mathcal{B}$ do
        if (node i is unmarked) dfs(i)

function dfs(j)
    mark node j
    for each i in $\mathcal{L}_j$ do
        if (node i is unmarked) dfs(i)
    push j onto stack for $\mathcal{X}$

$\boxed{\text{which can be less than } n}$

Sparse LU (Gilbert/Peierls)

- left-looking. kth step computes kth column of $\mathbf{L}$ and $\mathbf{U}$

Sparse LU (Gilbert/Peierls)

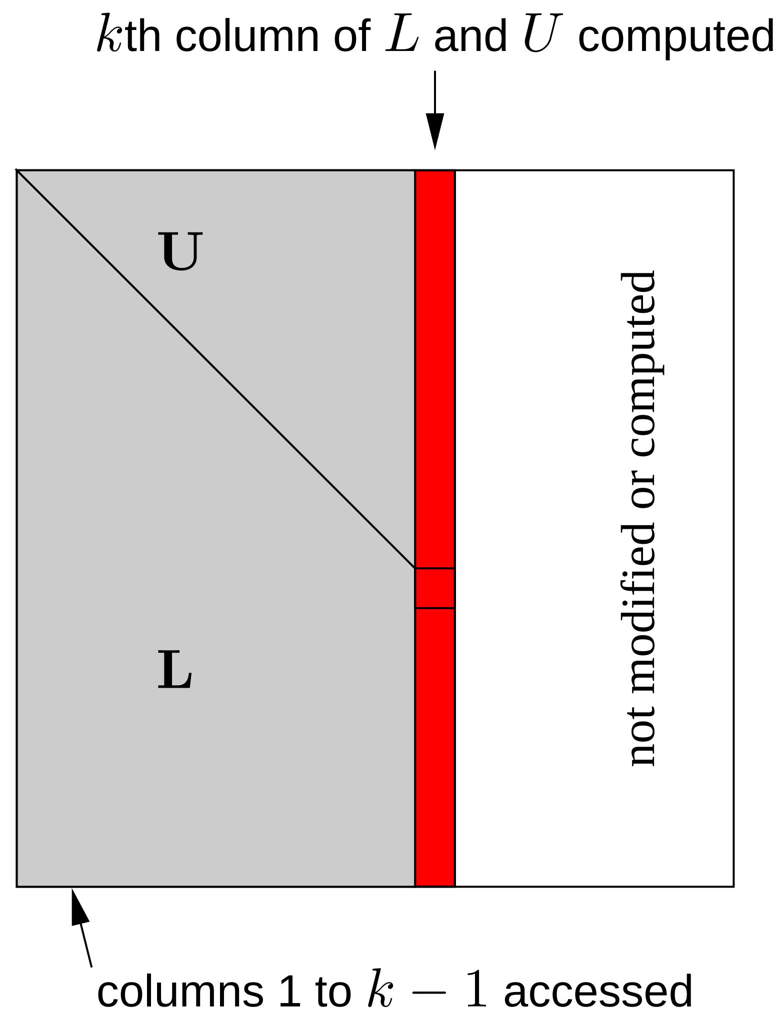
- left-looking. kth step computes kth column of $\mathbf{L}$ and $\mathbf{U}$

```
L = speye(n)
U = speye(n)
for k = 1:n
   x = L \ A(:,k)
   U(1:k,k) = x(1:k)
   L(k:n,k) = ...
      x(k:n) / U(k,k)
end
```

Sparse LU (Gilbert/Peierls)

- left-looking. kth step computes kth column of $\mathbf{L}$ and $\mathbf{U}$

```
L = speye(n)
U = speye(n)
for k = 1:n
   x = L \ A(:,k)
   U(1:k,k) = x(1:k)
   L(k:n,k) = ...
      x(k:n) / U(k,k)
end
```



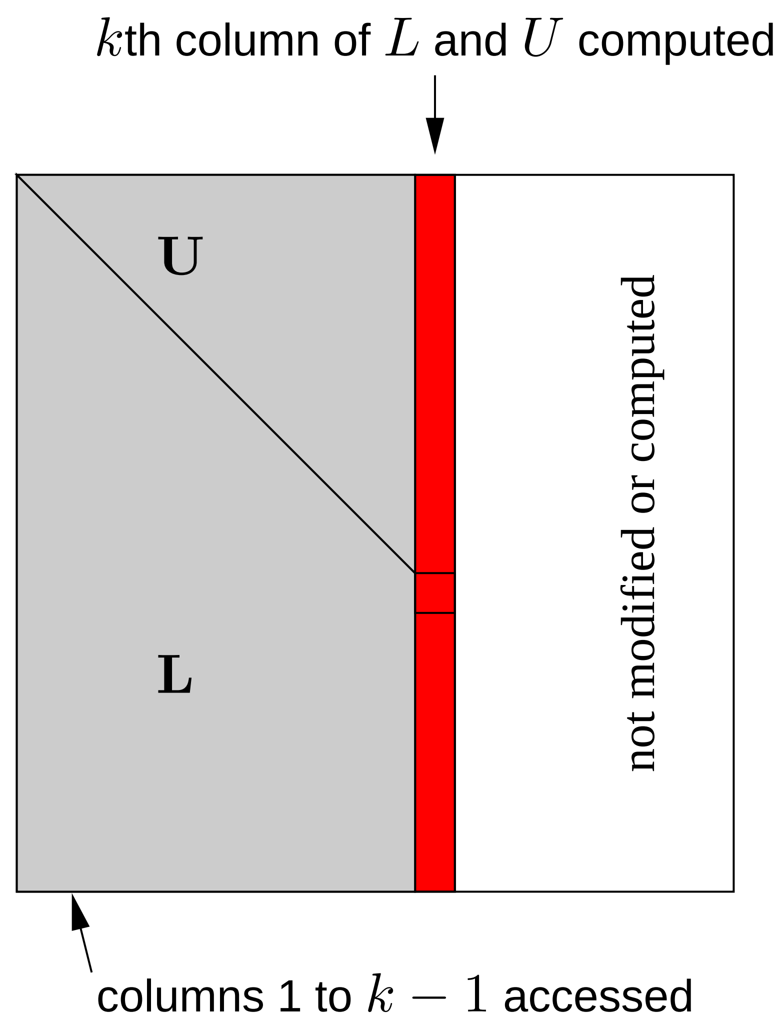
Sparse LU (Gilbert/Peierls)

- left-looking. kth step computes kth column of $\mathbf{L}$ and $\mathbf{U}$

```
L = speye(n)
U = speye(n)
for k = 1:n
    x = L \ A(:,k)
    U(1:k,k) = x(1:k)
    L(k:n,k) = ...
        x(k:n) / U(k,k)
end
```

$LU = PAQ$

- P: partial pivoting on x
- Q: fill-reducing column pre-ordering



Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} L_{11} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & l_{12} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & a_{22} \end{bmatrix}$$

Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} \mathbf{L_{11}} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} \mathbf{L_{11}^T} & l_{12} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A_{11}} & a_{12} \\ a_{12}^T & a_{22} \end{bmatrix}$$

1. factorize $\mathbf{L_{11}}\mathbf{L_{11}^T} = \mathbf{A_{11}}$

Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} \mathbf{L_{11}} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & \mathbf{l_{12}} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & \mathbf{a_{12}} \\ a_{12}^T & a_{22} \end{bmatrix}$$

1. factorize $L_{11} L_{11}^T = A_{11}$
2. solve $\mathbf{L_{11} l_{12}} = \mathbf{a_{12}}$ for l_{12}

Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} L_{11} & \\ \mathbf{l_{12}^T} & \mathbf{l_{22}} \end{bmatrix} \begin{bmatrix} L_{11}^T & \mathbf{l_{12}} \\ & \mathbf{l_{22}} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & \mathbf{a_{22}} \end{bmatrix}$$

1. factorize $L_{11}L_{11}^T = A_{11}$
2. solve $L_{11}l_{12} = a_{12}$ for l_{12}
3. $\mathbf{l_{22}} = \sqrt{\mathbf{a_{22}} - \mathbf{l_{12}^T}\mathbf{l_{12}}}$

Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} L_{11} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & l_{12} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & a_{22} \end{bmatrix}$$

1. factorize $L_{11}L_{11}^T = A_{11}$
2. solve $L_{11}l_{12} = a_{12}$ for l_{12}
3. $l_{22} = \sqrt{a_{22} - l_{12}^T l_{12}}$

for k = 1 to n

solve $L_{11}l_{12} = a_{12}$ for l_{12}

$l_{22} = \sqrt{a_{22} - l_{12}^T l_{12}}$

Sparse Cholesky, $LL^T = A$

$$\begin{bmatrix} L_{11} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & l_{12} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & a_{22} \end{bmatrix}$$

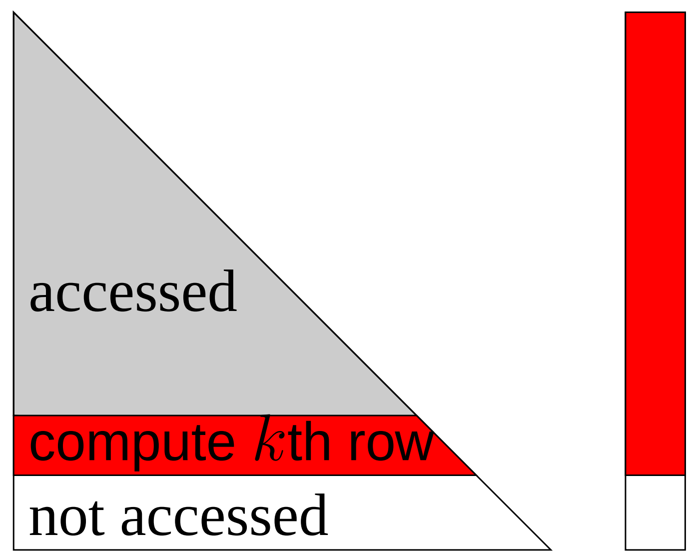
1. factorize $L_{11}L_{11}^T = A_{11}$
2. solve $L_{11}l_{12} = a_{12}$ for l_{12}
3. $l_{22} = \sqrt{a_{22} - l_{12}^T l_{12}}$

for k = 1 to n

  solve $L_{11}l_{12} = a_{12}$ for l_{12}

  $l_{22} = \sqrt{a_{22} - l_{12}^T l_{12}}$

an up-looking method

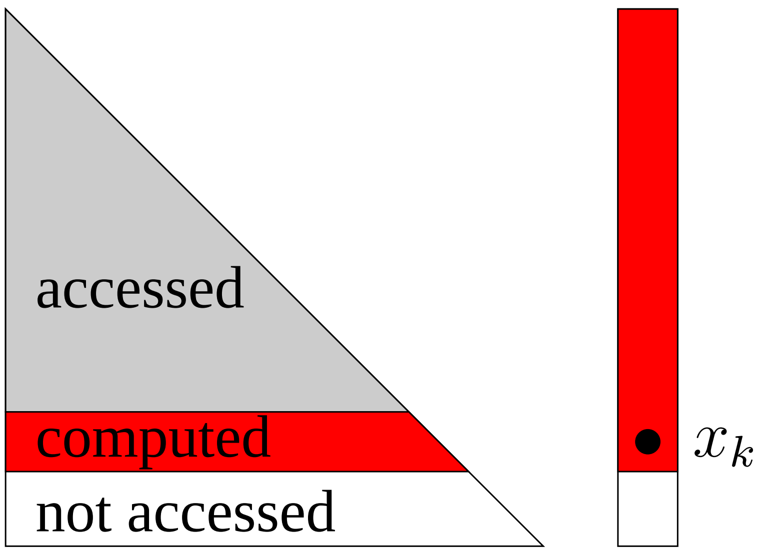


Sparse Cholesky: etree

- elimination tree
- arises in many direct methods
  - Compute nonzero pattern of `x=L\b` for a Cholesky `L` in time $O(|x|)$, the number of nonzeros in `x`
  - ...

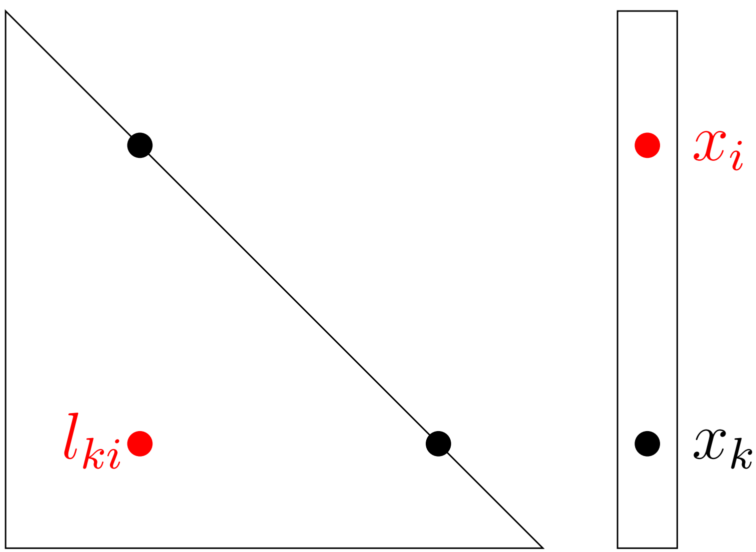
Sparse Cholesky: etree

Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



Sparse Cholesky: etree

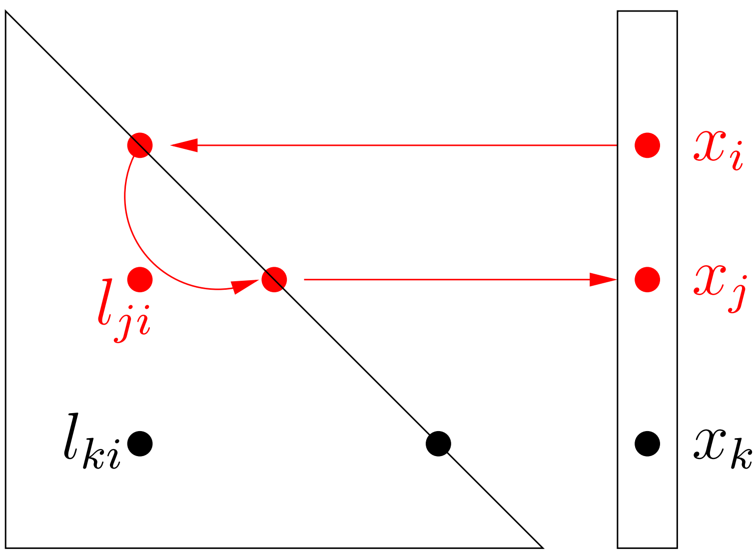
Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$

Sparse Cholesky: etree

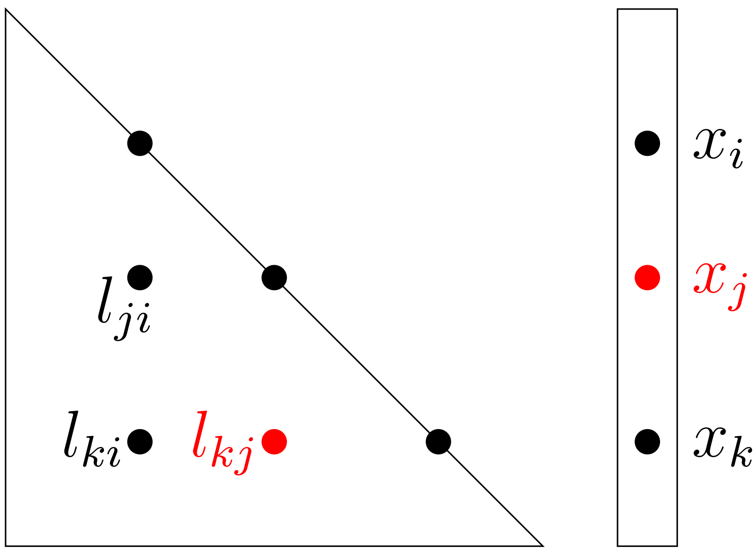
Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$
- ($l_{ji} \neq 0$ and $x_i \neq 0$) $\Rightarrow x_j \neq 0$

Sparse Cholesky: etree

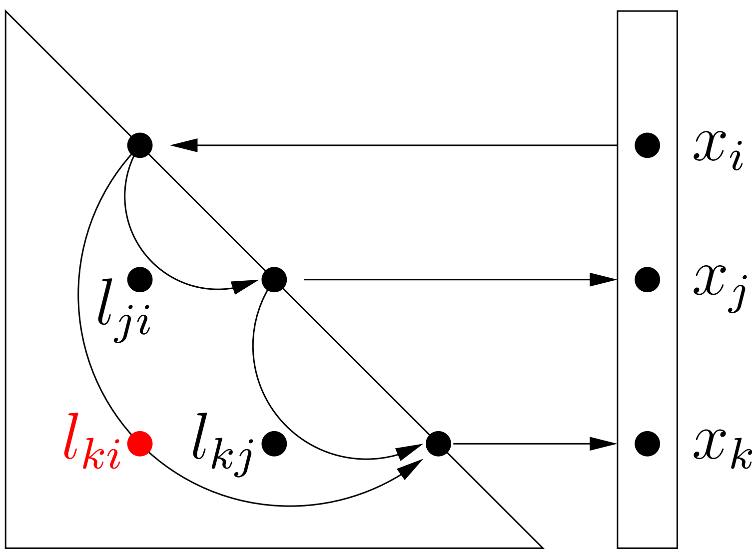
Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$
- ($l_{ji} \neq 0$ and $x_i \neq 0$) $\Rightarrow x_j \neq 0$
- $l_{kj} \neq 0 \Leftrightarrow x_j \neq 0$

Sparse Cholesky: etree

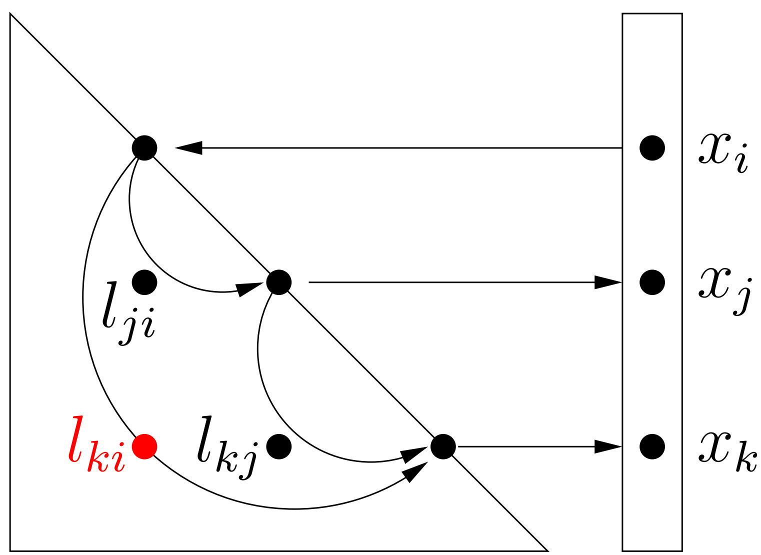
Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$
- ($l_{ji} \neq 0$ and $x_i \neq 0$) $\Rightarrow x_j \neq 0$
- $l_{kj} \neq 0 \Leftrightarrow x_j \neq 0$
- Thus, l_{ki} redundant for $\mathcal{X} =$ Reach$(\mathcal{B})$.

Sparse Cholesky: etree

Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:

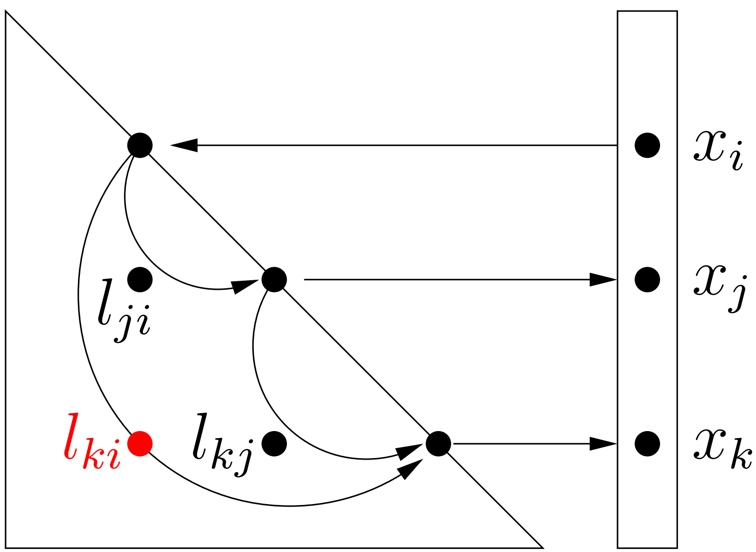


- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$
- ($l_{ji} \neq 0$ and $x_i \neq 0$) $\Rightarrow x_j \neq 0$
- $l_{kj} \neq 0 \Leftrightarrow x_j \neq 0$
- Thus, l_{ki} redundant for $\mathcal{X} = \text{Reach}(b)$.

- parent$(i) = \min\{j > i \mid l_{ji} \neq 0\}$; other edges redundant

Sparse Cholesky: etree

Elimination tree $\mathcal{T}$: pruning the graph of L.
Consider computing kth row of L:



- $l_{ki} \neq 0 \Leftrightarrow x_i \neq 0$
- ($l_{ji} \neq 0$ and $x_i \neq 0$) $\Rightarrow x_j \neq 0$
- $l_{kj} \neq 0 \Leftrightarrow x_j \neq 0$
- Thus, l_{ki} redundant for $\mathcal{X} = \text{Reach}(b)$.

- parent$(i) = \min\{j > i \mid l_{ji} \neq 0\}$; other edges redundant
- $\mathcal{L}_{k*} = \text{Reach}(A_{1:k,k})$ in $O(|\mathcal{L}_{k*}|)$ time

Sparse Cholesky: etree

$$A$$

$$\begin{bmatrix}
1 & & & & & \bullet & \bullet & & & & \\
& 2 & \bullet & & & & & \bullet & & & \\
& \bullet & 3 & & & & & & & \bullet & \bullet \\
& & & 4 & & \bullet & & & & \bullet & \\
& & & & 5 & & & \bullet & & & \bullet \\
\bullet & & & \bullet & & 6 & & & \bullet & \bullet & \\
\bullet & & & & & & 7 & & & & \bullet \\
& \bullet & & & \bullet & & & 8 & & \bullet & \bullet \\
& & & & & \bullet & & & 9 & & \\
& & \bullet & \bullet & & \bullet & & \bullet & & 10 & \bullet \\
& & \bullet & & \bullet & & \bullet & \bullet & & \bullet & 11
\end{bmatrix}$$

Sparse Cholesky: etree

A

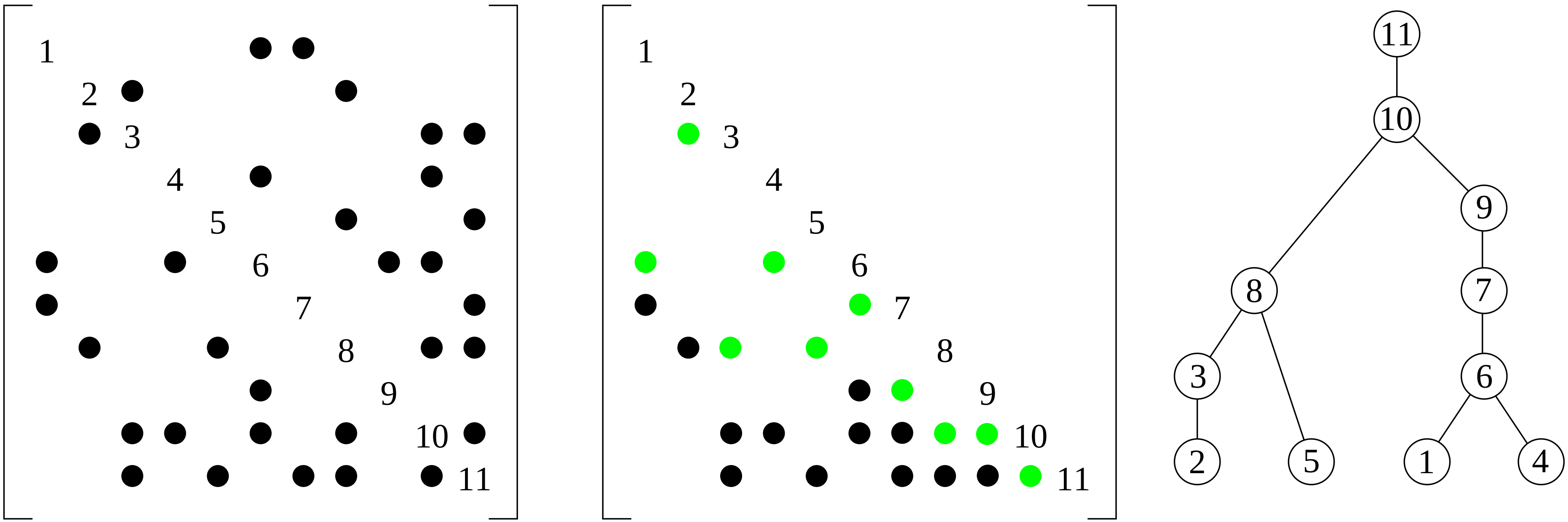
$$\begin{bmatrix}
1 & & & & & \bullet & \bullet & & & & \\
& 2 & \bullet & & & & & \bullet & & & \\
& \bullet & 3 & & & & & & & \bullet & \bullet \\
& & & 4 & & \bullet & & & & \bullet & \\
& & & & 5 & & & \bullet & & & \bullet \\
\bullet & & & \bullet & & 6 & & & \bullet & \bullet & \\
\bullet & & & & & & 7 & & & & \bullet \\
& \bullet & & & \bullet & & & 8 & & \bullet & \bullet \\
& & & & & \bullet & & & 9 & & \\
& & \bullet & \bullet & & \bullet & & \bullet & & 10 & \bullet \\
& & \bullet & & \bullet & & \bullet & \bullet & & \bullet & 11
\end{bmatrix}$$

Cholesky factor L of A

$$\begin{bmatrix}
1 & & & & & & & & & & \\
& 2 & & & & & & & & & \\
& \bullet & 3 & & & & & & & & \\
& & & 4 & & & & & & & \\
& & & & 5 & & & & & & \\
\bullet & & & \bullet & & 6 & & & & & \\
\bullet & & & & & {\color{red}\bullet} & 7 & & & & \\
& \bullet & {\color{red}\bullet} & & \bullet & & & 8 & & & \\
& & & & & \bullet & {\color{red}\bullet} & & 9 & & \\
& & \bullet & \bullet & & \bullet & {\color{red}\bullet} & \bullet & {\color{red}\bullet} & 10 & \\
& & \bullet & & \bullet & & \bullet & \bullet & {\color{red}\bullet} & \bullet & 11
\end{bmatrix}$$

Sparse Cholesky: etree

A | Cholesky factor L of A | elimination tree



Sparse Cholesky: overview

- Symbolic analysis:

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$
  - some methods find $\mathcal{L}$: $O(|L|)$ or less

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$
  - some methods find $\mathcal{L}$: $O(|L|)$ or less
- Numeric factorization:

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$
  - some methods find $\mathcal{L}$: $O(|L|)$ or less
- Numeric factorization:
  - up-looking

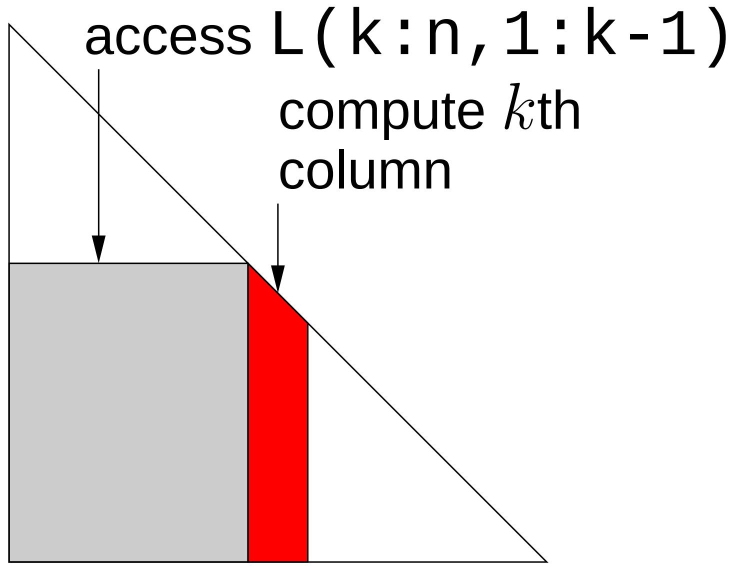
Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$
  - some methods find $\mathcal{L}$: $O(|L|)$ or less
- Numeric factorization:
  - up-looking
  - left-looking, supernodal

Sparse Cholesky: overview

- Symbolic analysis:
  - fill-reducing ordering, $\bar{A} = PAP^T = LL^T$
  - etree of $\bar{A}$: nearly $O(|A|)$
  - depth-first postordering of etree: $O(n)$
  - column counts of L: nearly $O(|A|)$
  - some methods find $\mathcal{L}$: $O(|L|)$ or less
- Numeric factorization:
  - up-looking
  - left-looking, supernodal
  - right-looking, multifrontal

Sparse Cholesky: left-looking



for $k = 1$ to n

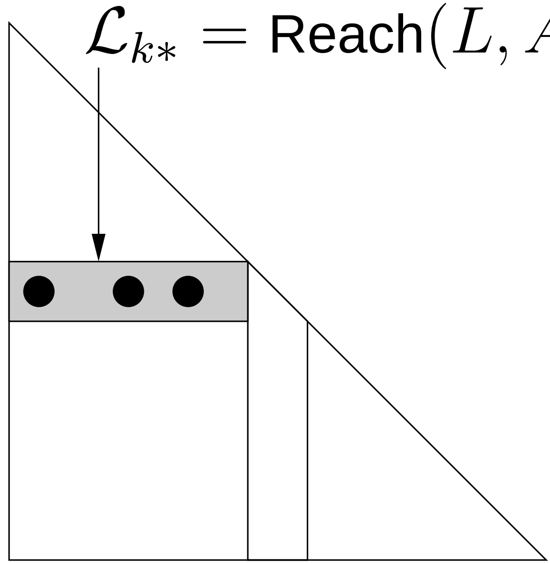
```
    x = A(k:n,k)
```

    for each j in Reach$(L, A(1:k,k))$

```
        x(k:n) = x(k:n) - L(k:n,j) * L(k,j)
    L(k,k) = sqrt(x(k))
    L(k+1:n,k) = x(k) / L(k,k)
```

Sparse Cholesky: left-looking

$\mathcal{L}_{k*} = \text{Reach}(L, A(1:k,k))$



for $k = 1$ to n

```
    x = A(k:n,k)
```

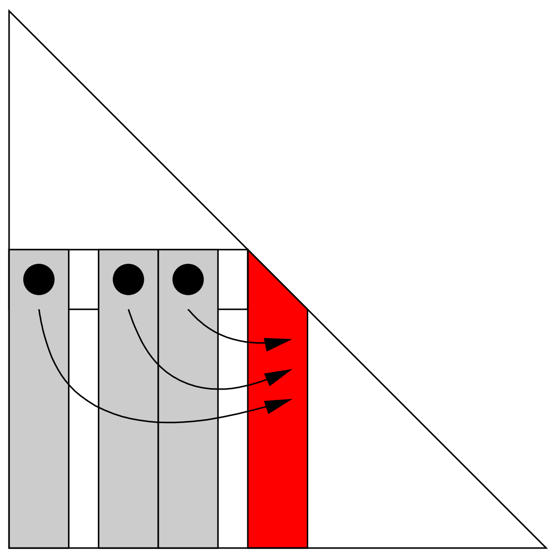
    for each j in $\text{Reach}(L, A(1:k,k))$

        ...

    ...

    ...

Sparse Cholesky: left-looking



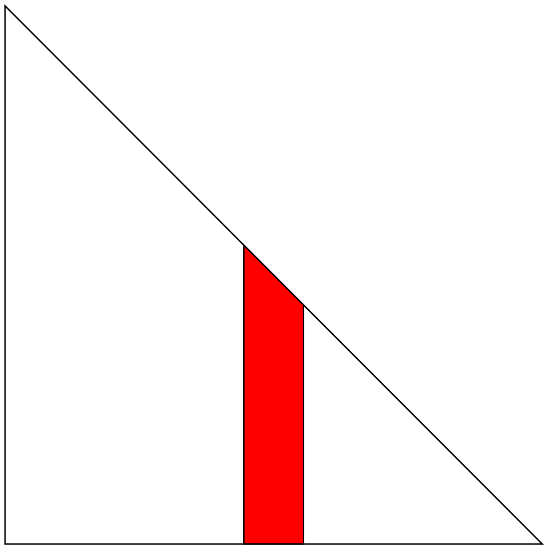
for $k = 1$ to n

```
x = A(k:n,k)
```

for each j in Reach$(L, A(1:k,k))$

```
    x(k:n) = x(k:n) - L(k:n,j) * L(k,j)
...
...
```

Sparse Cholesky: left-looking



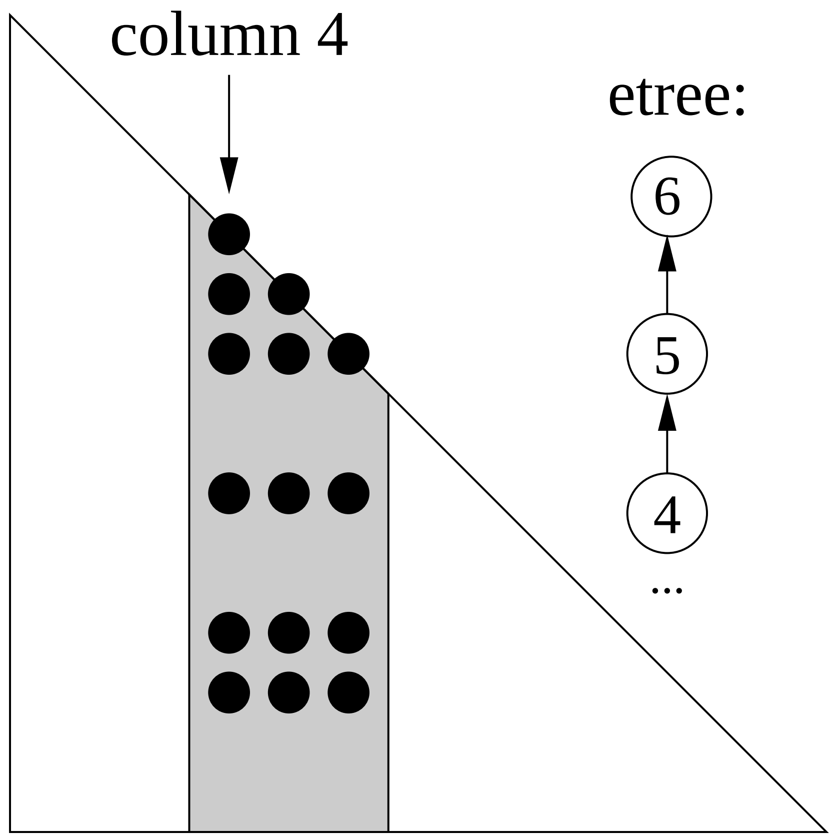
for $k = 1$ to n

```
    x = A(k:n,k)
```

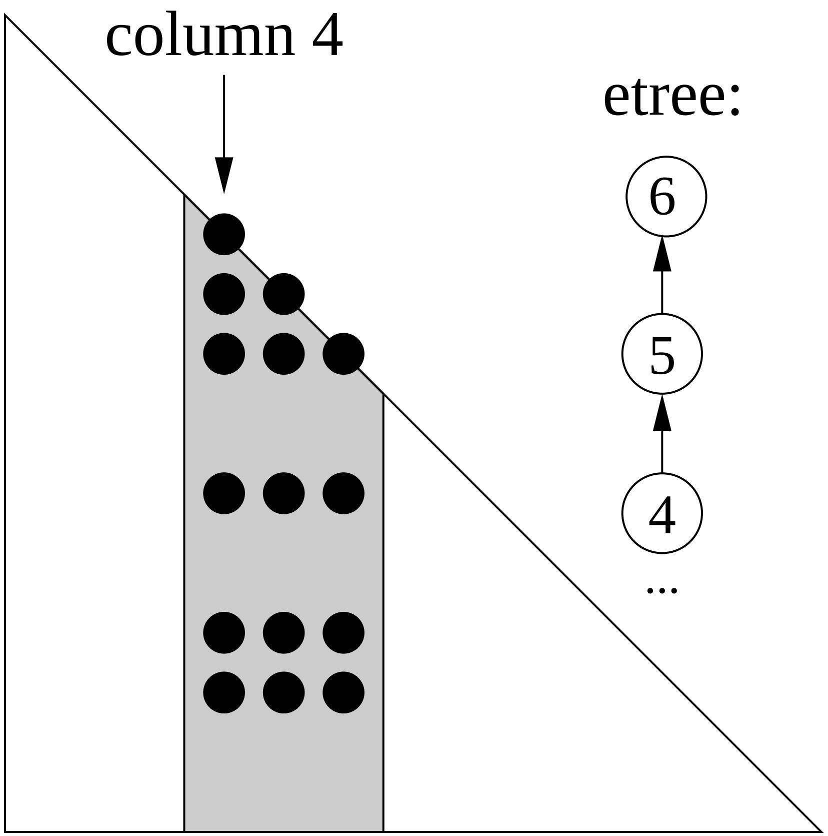
    for each j in Reach$(L, A(1:k,k))$

```
        x(k:n) = x(k:n) - L(k:n,j) * L(k,j)
    L(k,k) = sqrt(x(k))
    L(k+1:n,k) = x(k) / L(k,k)
```

Sparse Cholesky: supernodal

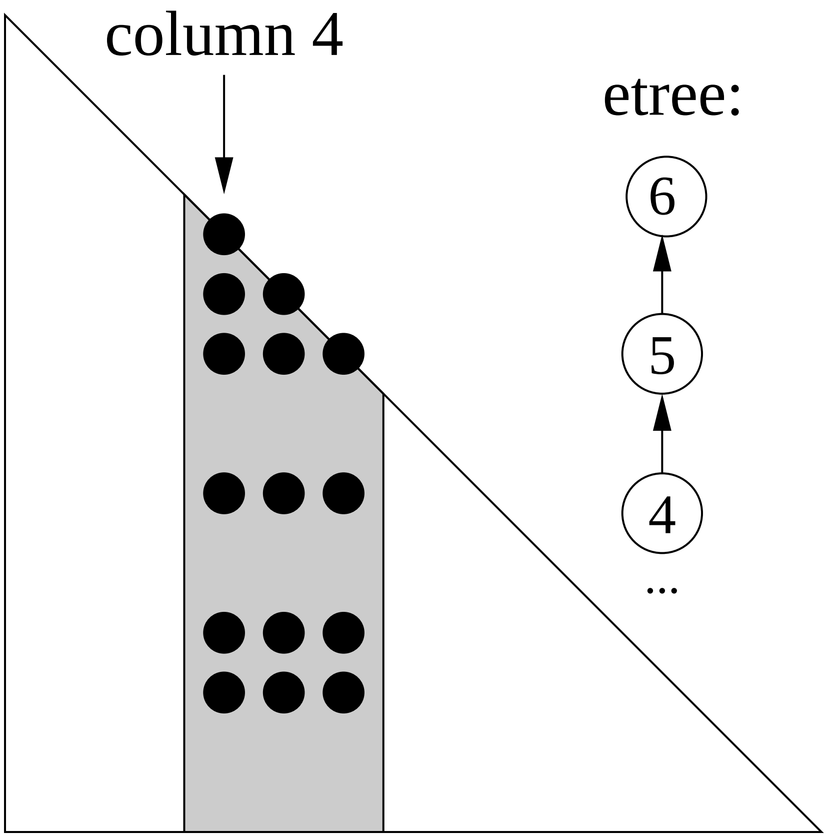


Sparse Cholesky: supernodal



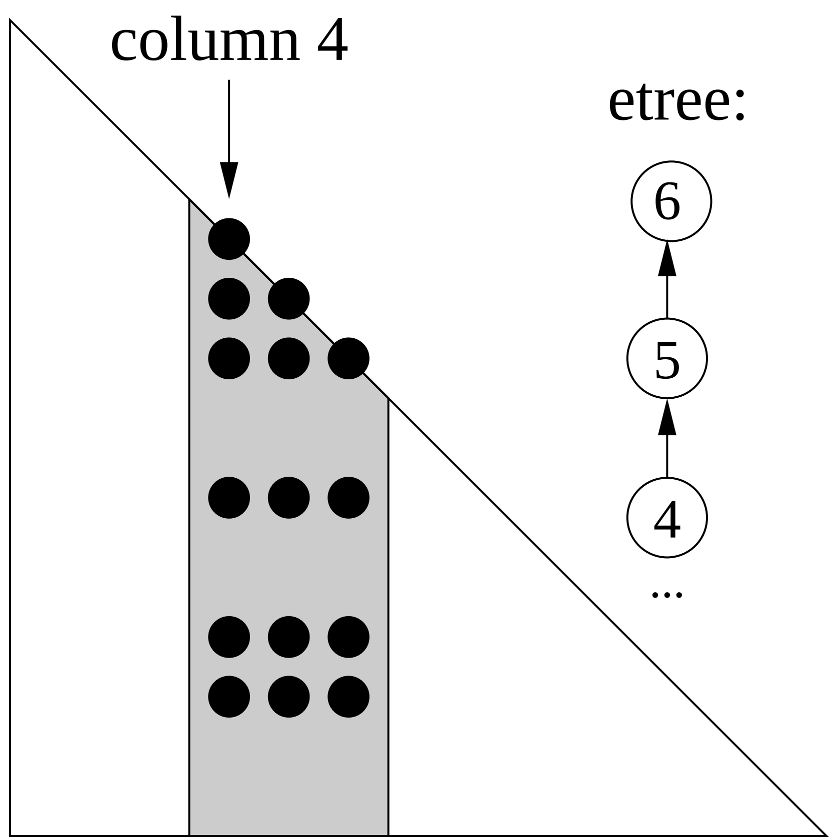
- Adjacent columns of L often have identical pattern

Sparse Cholesky: supernodal



- Adjacent columns of L often have identical pattern
- a chain in the elimination tree

Sparse Cholesky: supernodal

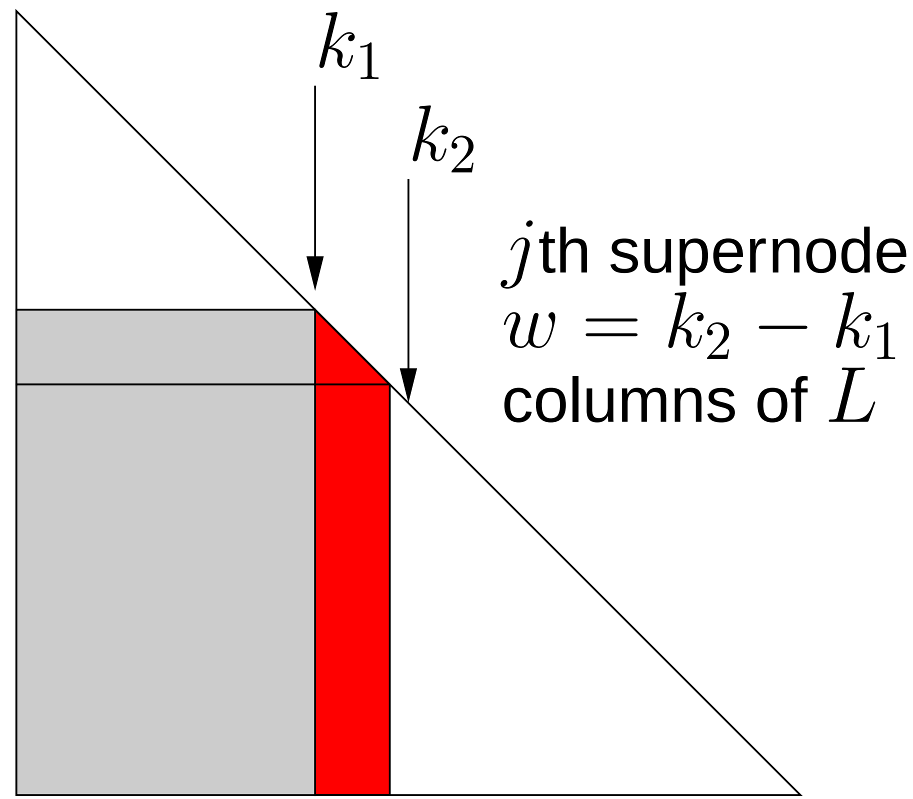


- Adjacent columns of L often have identical pattern
- a chain in the elimination tree
- can exploit dense submatrix operations

Sparse Cholesky: supernodal

block left-looking

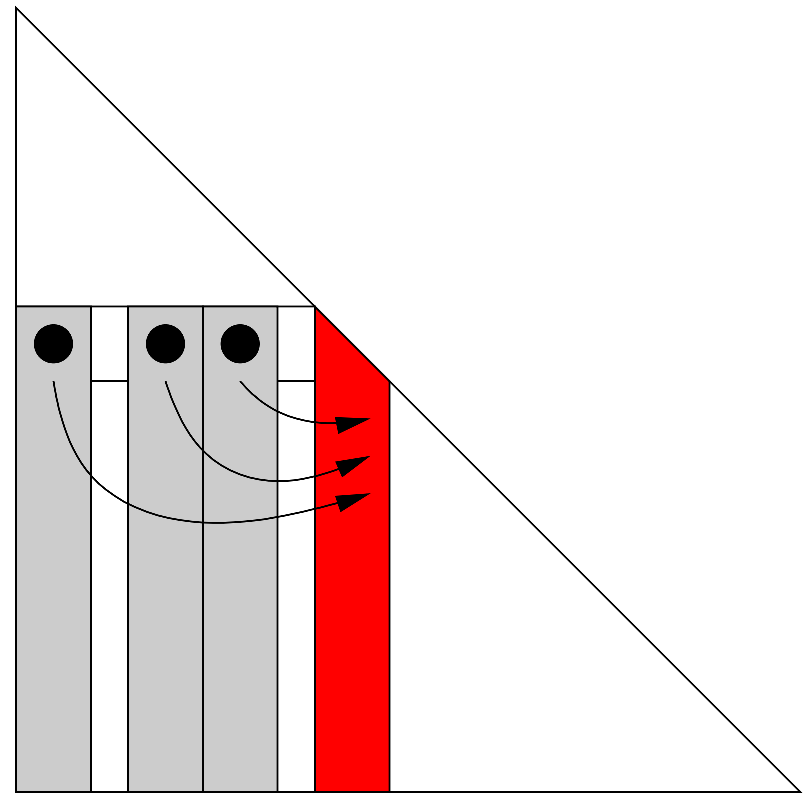
- for jth supernode:



Sparse Cholesky: supernodal

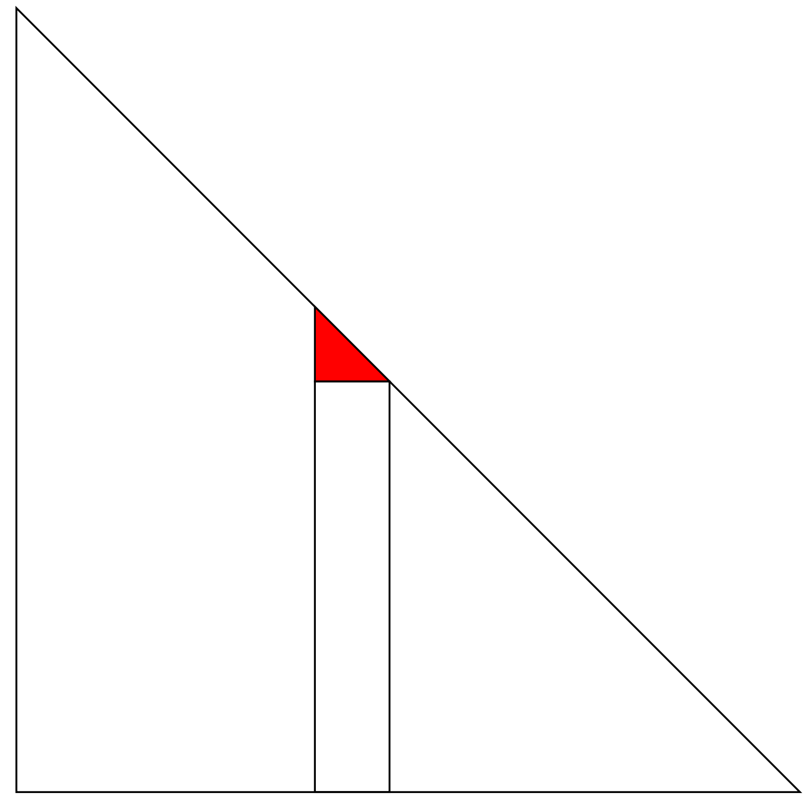
block left-looking

- for jth supernode:
- (1) sparse block matrix multiply



Sparse Cholesky: supernodal

block left-looking

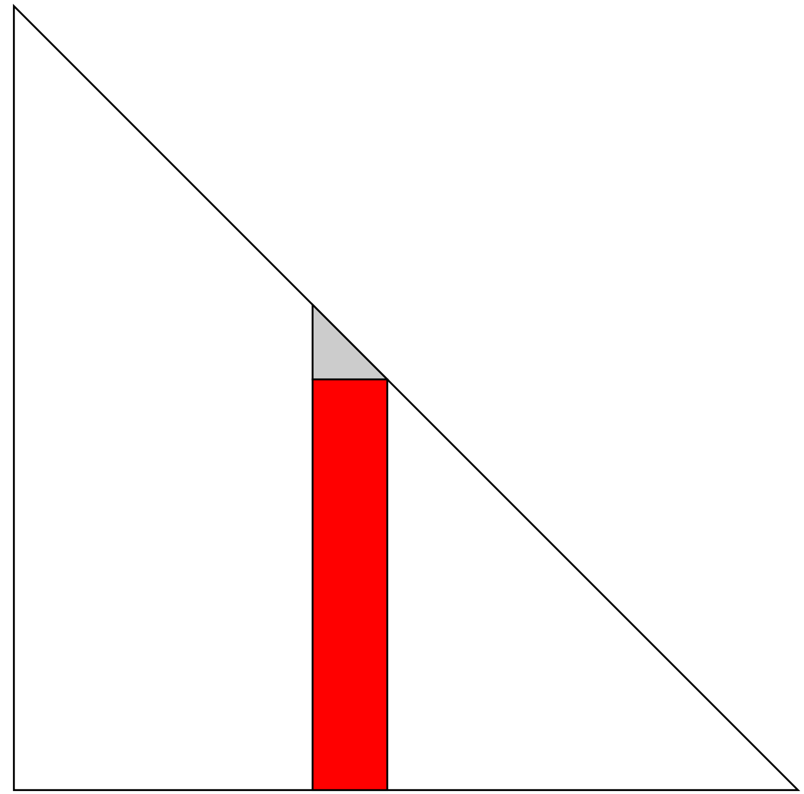


- for jth supernode:
- (1) sparse block matrix multiply
- (2) dense Cholesky

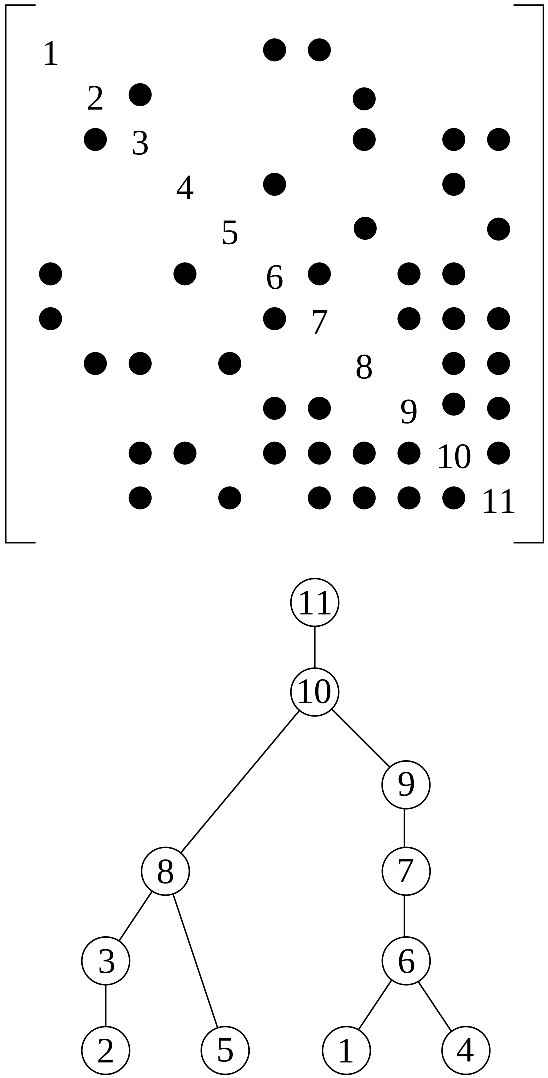
Sparse Cholesky: supernodal

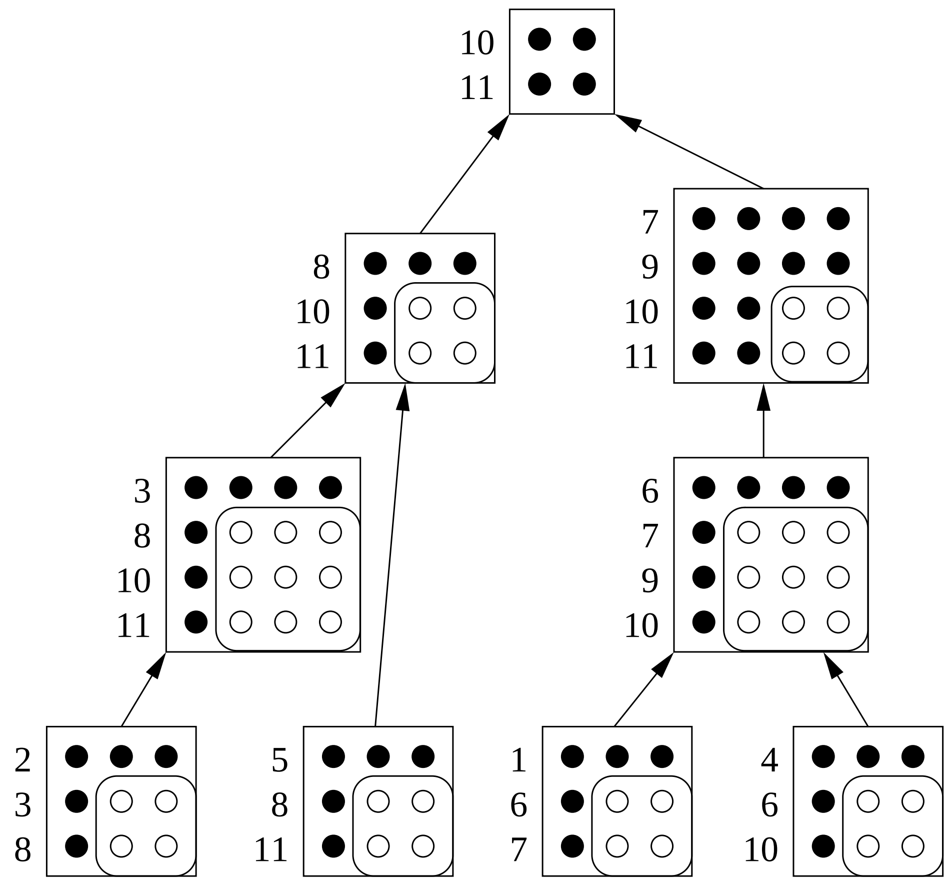
block left-looking

- for jth supernode:
- (1) sparse block matrix multiply
- (2) dense Cholesky
- (3) dense block $Lx = b^T$ solve



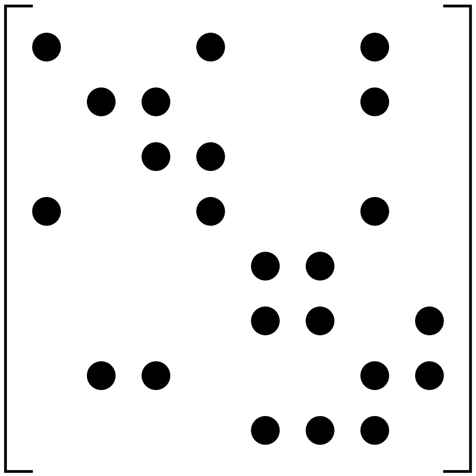
Sparse LU: multifrontal



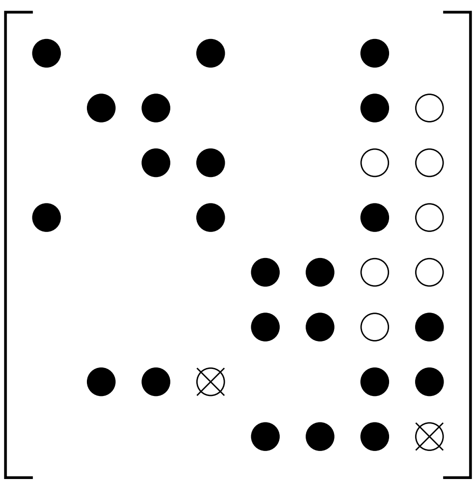


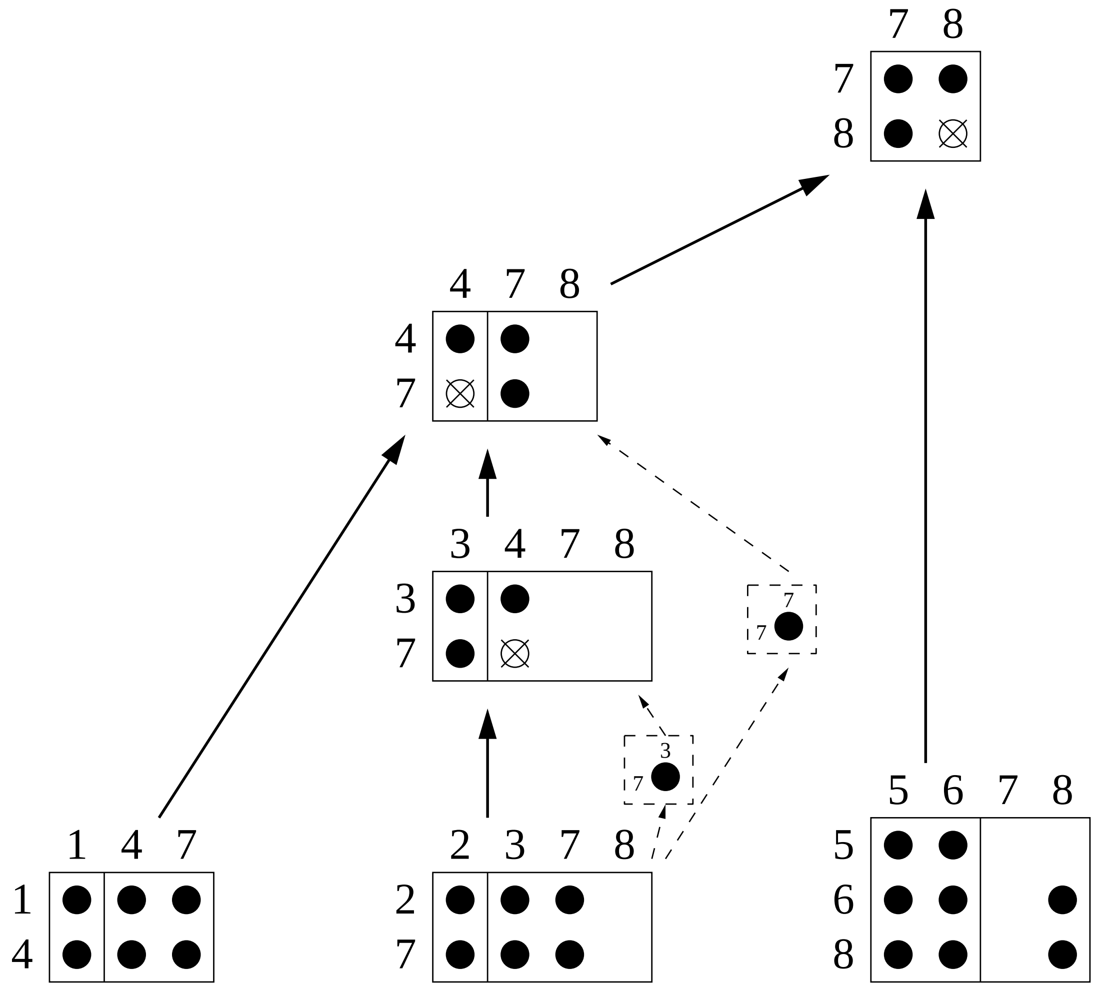
Sparse LU: UMFPACK

A



LU factors





x=A\b **when A is sparse**

x=A\b when A is sparse

- if A diagonal: scale each row

x=A\b when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK

x=A\b when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK
- if A lower or upper triangular: forward/backsolve

x=A\b when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK
- if A lower or upper triangular: forward/backsolve
- if A rectangular: Givens-based QR

`x=A\b` when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK
- if A lower or upper triangular: forward/backsolve
- if A rectangular: Givens-based QR
- if `A=A'` and `all(diag(A)>0)`: try `chol`, supernodal or up-looking (CHOLMOD)

`x=A\b` when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK
- if A lower or upper triangular: forward/backsolve
- if A rectangular: Givens-based QR
- if `A=A'` and `all(diag(A)>0)`: try `chol`, supernodal or up-looking (CHOLMOD)
- else: `lu`, either multifrontal (UMFPACK) or left-looking (GPLU)

`x=A\b` when A is sparse

- if A diagonal: scale each row
- if A banded: LAPACK
- if A lower or upper triangular: forward/backsolve
- if A rectangular: Givens-based QR
- if `A=A'` and `all(diag(A)>0)`: try `chol`, supernodal or up-looking (CHOLMOD)
- else: `lu`, either multifrontal (UMFPACK) or left-looking (GPLU)
- http://www.cise.ufl.edu/research/sparse

Postscript

- Up-coming book: *Direct Methods for Sparse Linear Systems*, SIAM, Sept. 2006.

Postscript

- Up-coming book: *Direct Methods for Sparse Linear Systems*, SIAM, Sept. 2006.
- Sparse Cholesky update/downdate
  - Given $A = LL^T$, compute $\overline{LL}^T = A \pm ww^T$
  - *"among the most important algorithms in linear algebra"*, Wilkinson
  - time proportional to number of entries that change
  - columns of L that change: sparsity pattern of `x=L\w`

Direct methods for sparse linear systems

```
x = A\b
```

Direct methods for sparse linear systems

```
x = A\b
```

- Sparse matrix algorithms: numerics plus graph theory

Direct methods for sparse linear systems

```
x = A\b
```

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up

Direct methods for sparse linear systems

```
x = A\b
```

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)

Direct methods for sparse linear systems

`x = A\b`

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)

Direct methods for sparse linear systems

`x = A\b`

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)

Direct methods for sparse linear systems

`x = A\b`

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)
- Supernodal and multifrontal methods (`x=A\b`)

Direct methods for sparse linear systems

`x = A\b`

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)
- Supernodal and multifrontal methods (`x=A\b`)
- Sparse Cholesky update/downdate (`cholupdate`)

Direct methods for sparse linear systems

`x = A\b`

- Sparse matrix algorithms: numerics plus graph theory
- Goal: sparse matrix methods from the ground up
- Lower triangular solve (`x=L\b`)
- Sparse LU factorization (`[L,U,P]=lu(A)`)
- Sparse Cholesky factorization (`L=chol(A)'`)
- Supernodal and multifrontal methods (`x=A\b`)
- Sparse Cholesky update/downdate (`cholupdate`)
- Up-coming SIAM book: September 2006