On finding approximate supernodes for an efficient ILU(k) factorization

PMAA'06

Rennes

P. Hénon, P. Ramet, J. Roman

LaBRI, UMR CNRS 5800, Université Bordeaux I & ENSEIRB

Projet ScAlApplix, INRIA UR Futurs

Motivation of this work

- A popular choice as an algebraic preconditioner is an ILU(k) preconditioner (level-of-fill based inc. facto.)

- BUT

➢ Parallelization is not easy

➢ Scalar formulation does not take advantage of superscalar effect (i.e. BLAS)
=> Usually a low value of fill is used (k=0 or k=1)

Motivation of this work

ILU + Krylov Methods

Based on scalar implementation

Difficult to parallelize (mostly DD + Schwartz additive => # of iterations depends on the number of processors)

Low memory consumption

Precision ~ 10^-5

Direct methods

BLAS3 (mostly DGEMM)
Thread/SMP, Load Balance…

Parallelization job is done (MUMPS, PASTIX, SUPERLU…)

High memory consumption : very large 3D problems are out of their league (100 millions unknowns)

Great precision ~ 10^-18

We want a trade-off !

Motivation of this work

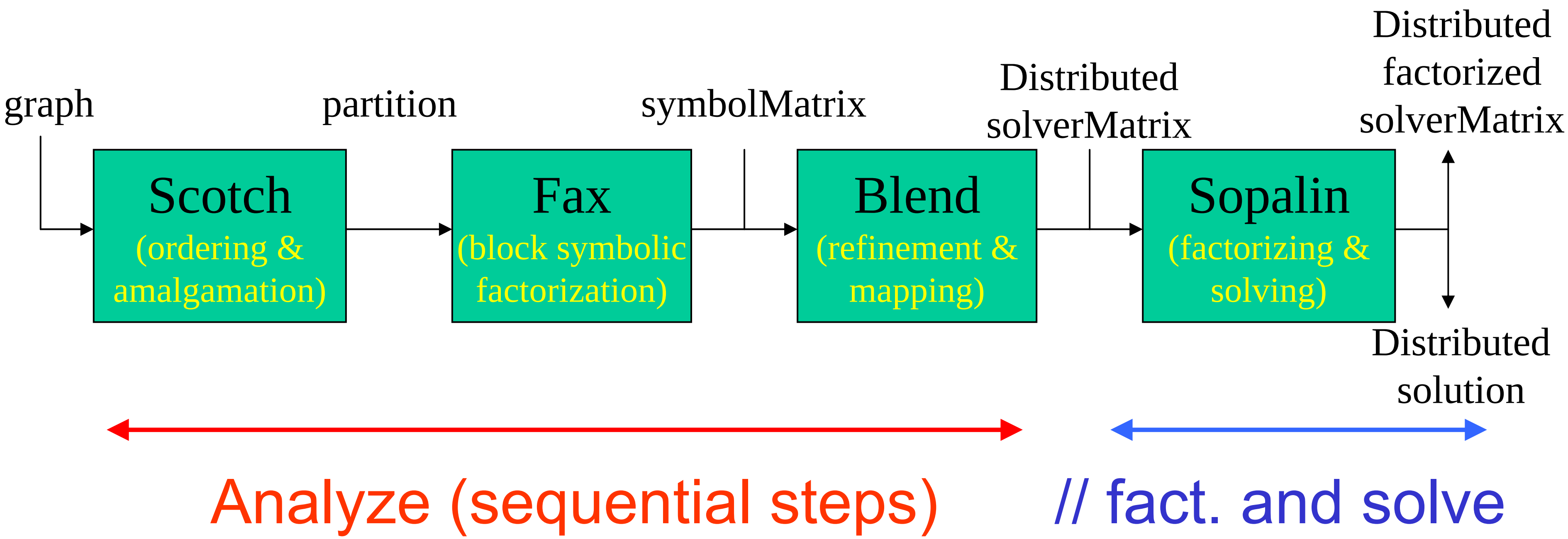
- Goal: we want to adapt a (supernodal) parallel direct solver (PaStiX) to build an incomplete block factorization and benefit from all the features that it provides:

  - Algorithmic is based on linear algebra kernels (BLAS)
  - Load-balancing and task scheduling are based on a fine modeling of computation and communication
  - Modern architecture management (SMP nodes) : hybrid Threads/MPI implementation

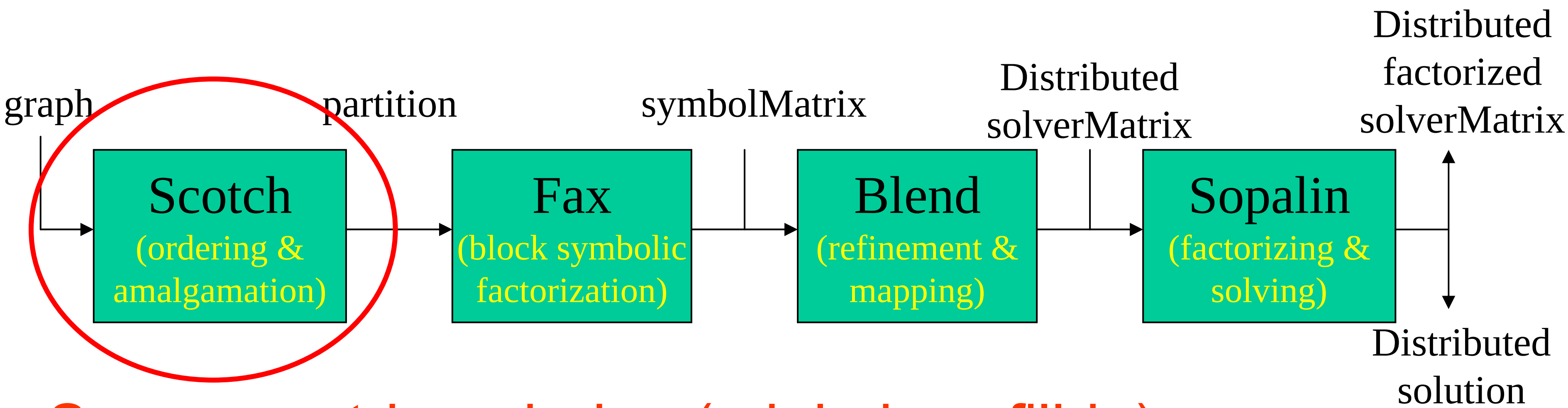
Outlines

- Which modifications in the direct solver?
- The symbolic incomplete factorization
- An algorithm to get dense blocks in ILU
- Experiments
- Conclusion

Direct solver chain (in PaStiX)



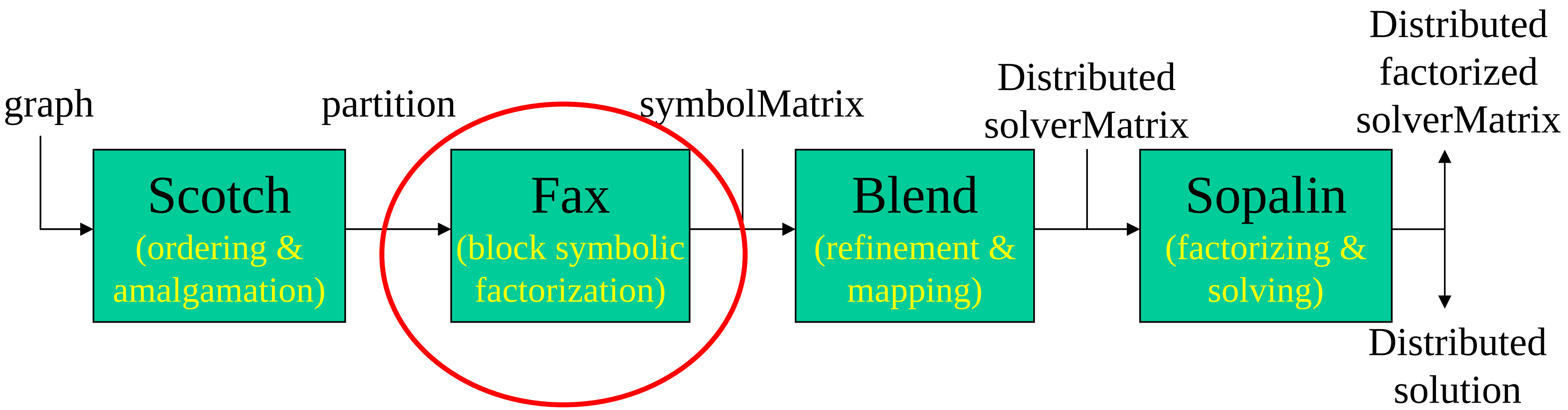
Direct solver chain (in PaStiX)



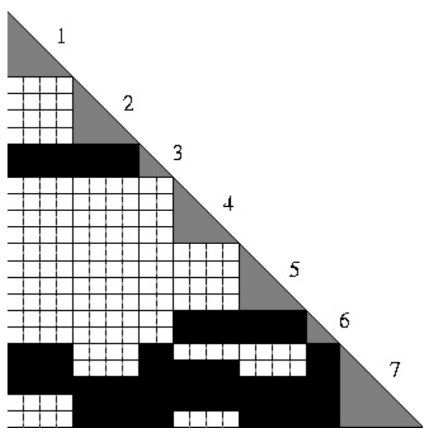
Sparse matrix ordering (minimizes fill-in)

- Scotch: an hybrid algorithm
  - incomplete Nested Dissection
  - the resulting subgraphs being ordered with an Approximate Minimum Degree method under constraints (HAMD)

Direct solver chain (in PaStiX)

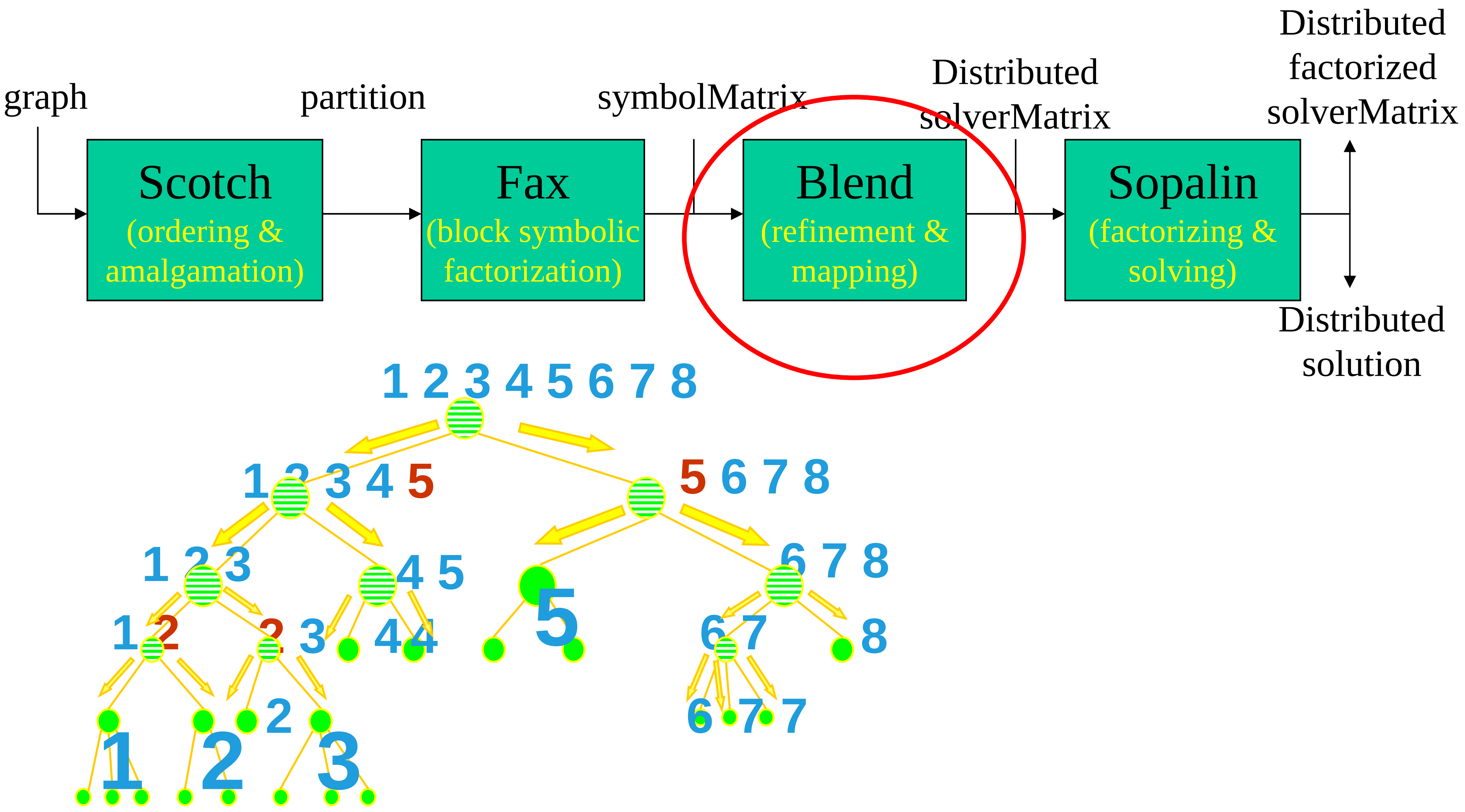


The symbolic block factorization

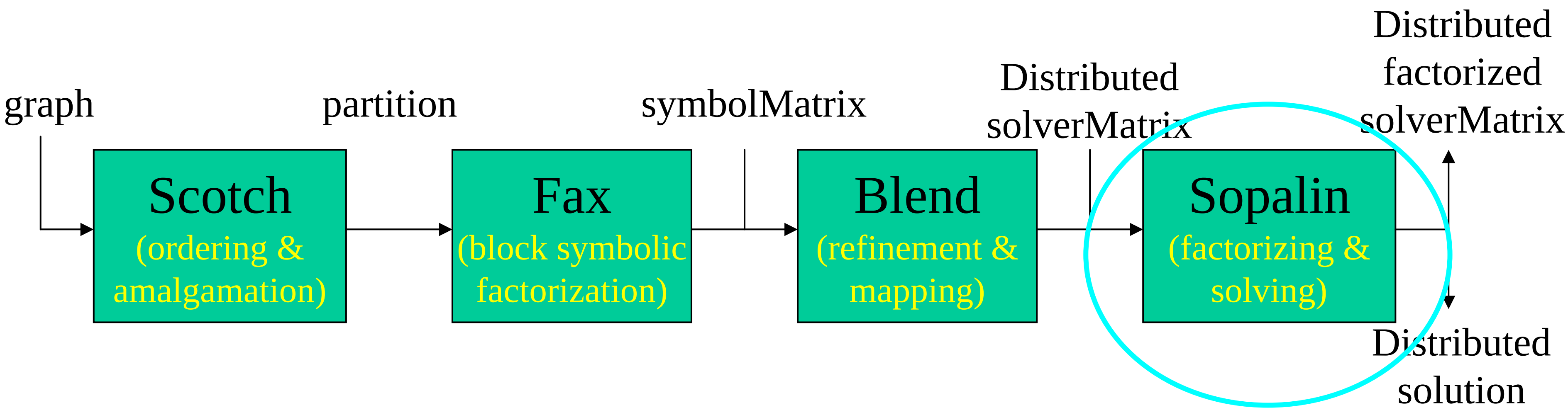


- ***Q(G,P)→Q(G,P)*=Q(G*,P) => linear in number of blocks!***
- Dense block structures ⇒ only a extra few pointers to store the matrix

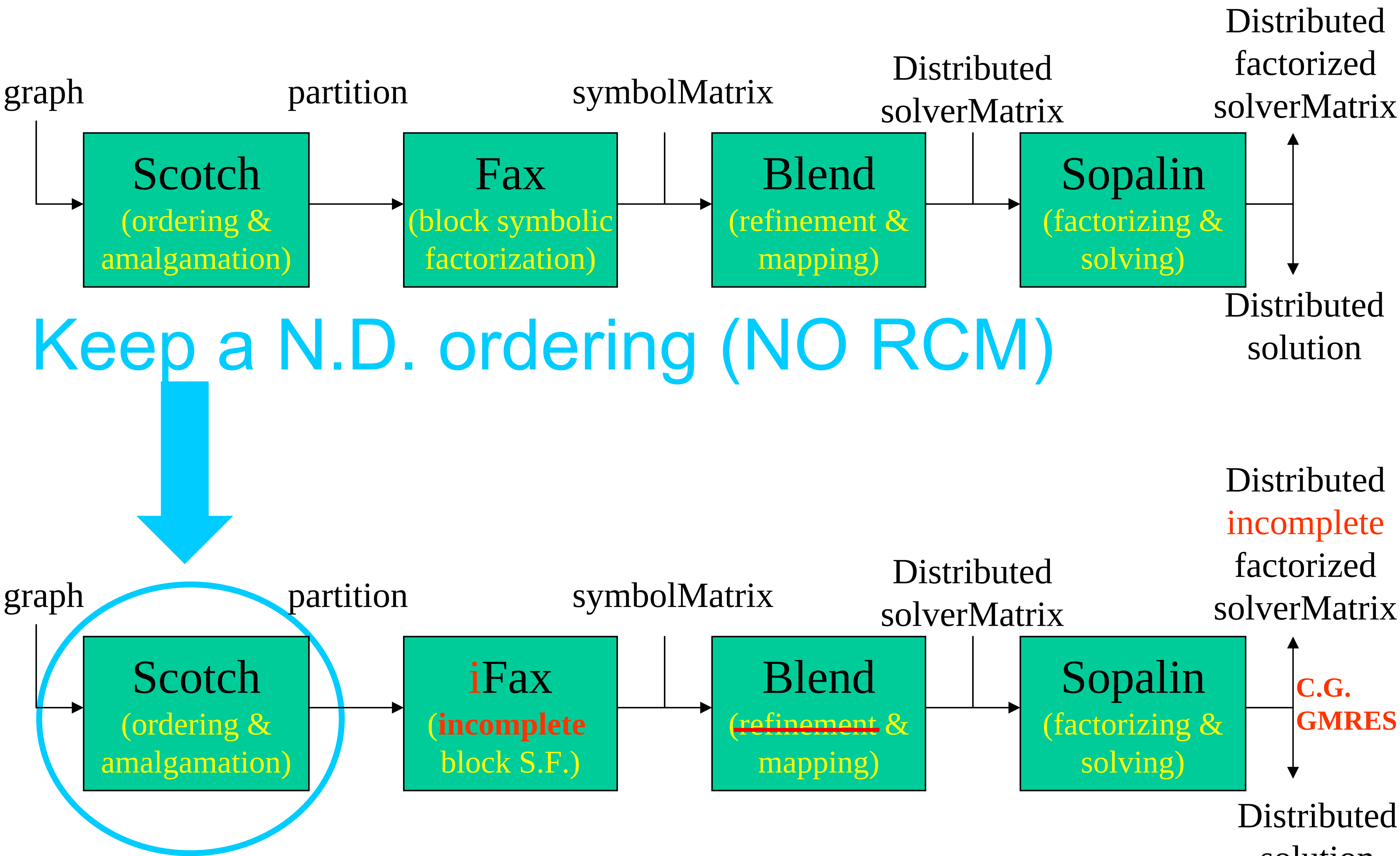
Direct solver chain (in PaStiX)

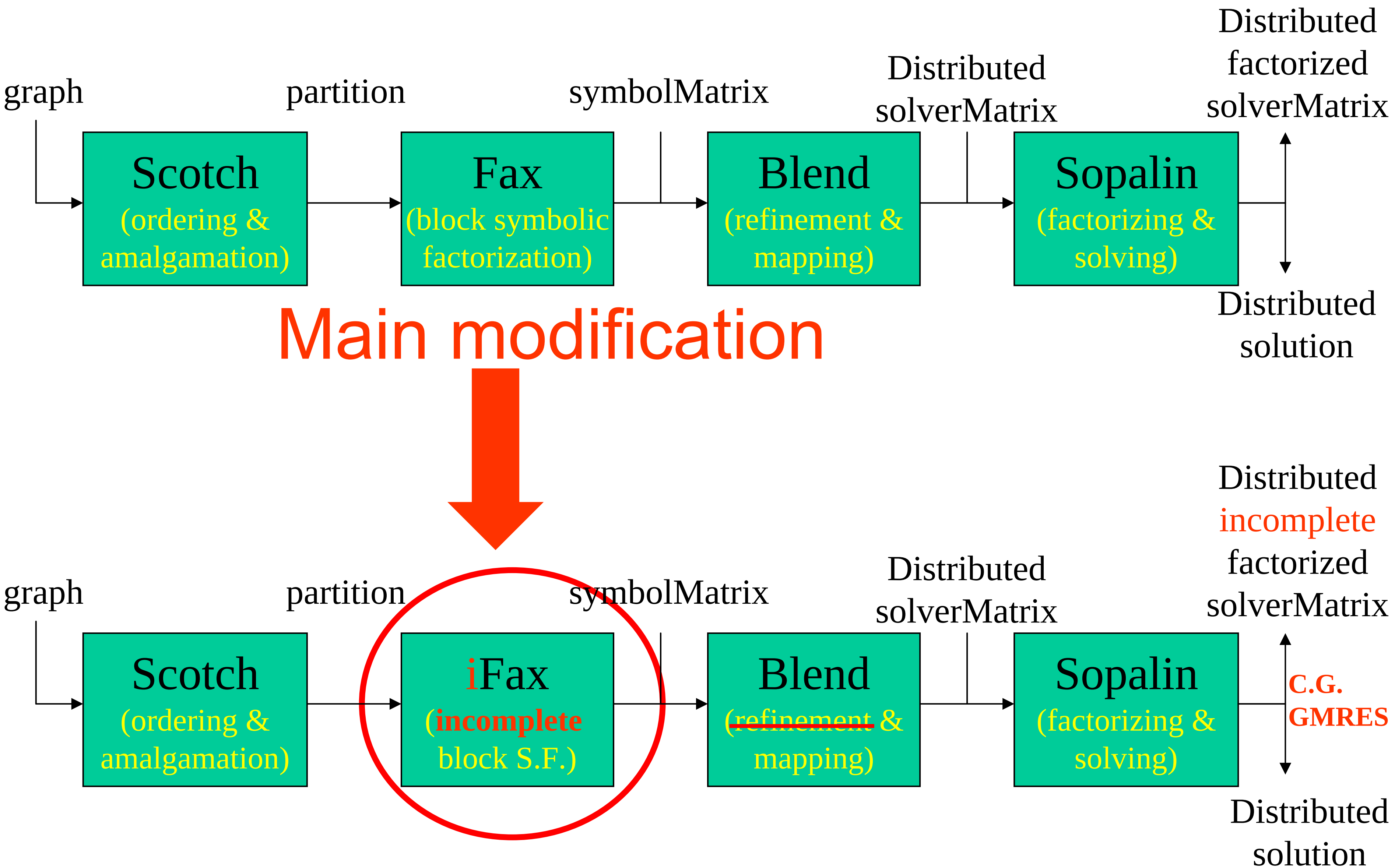


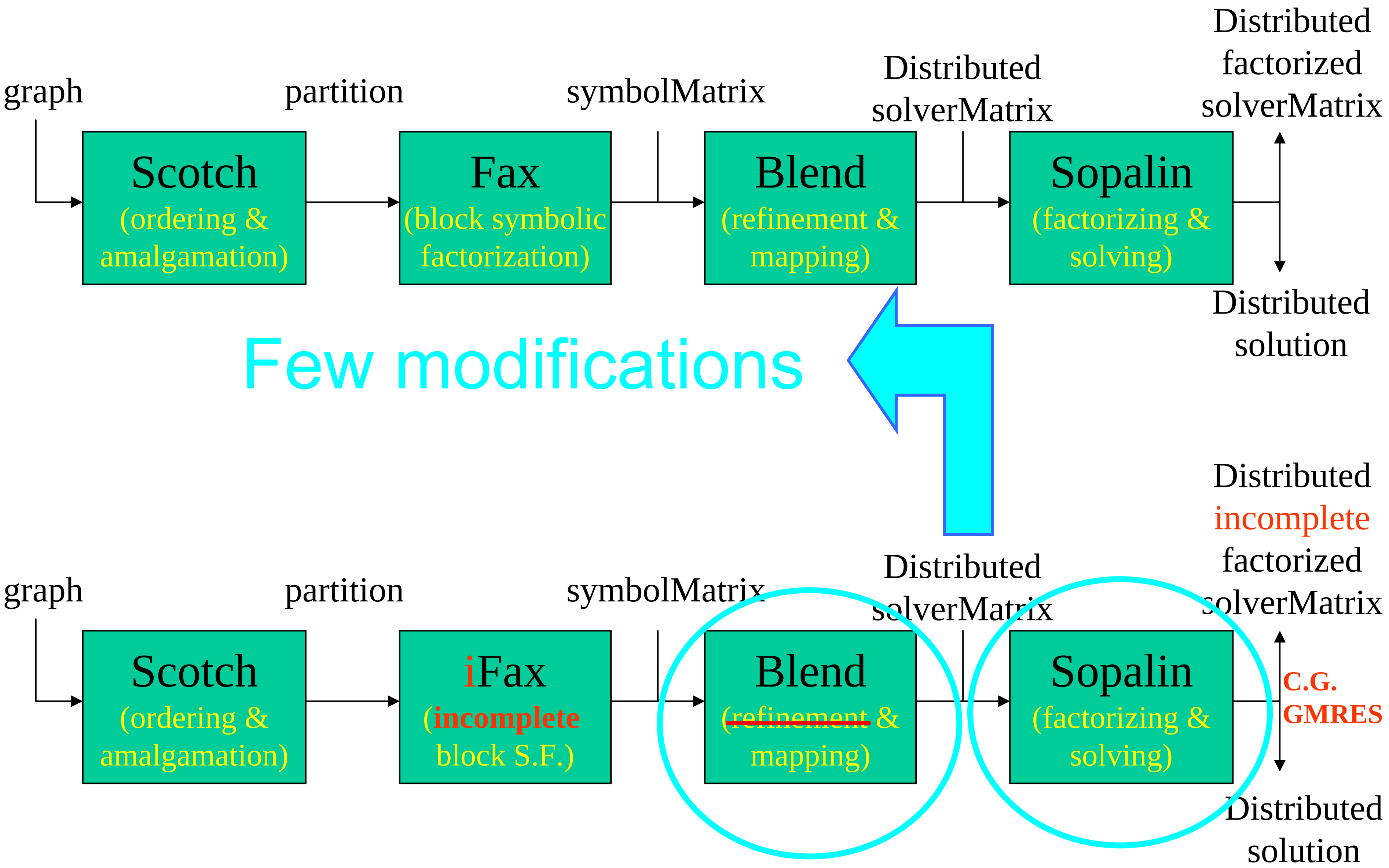
Direct solver chain (in PaStiX)



- Modern architecture management (SMP nodes) : hybrid Threads/MPI implementation (all processors in the same SMP node work directly in share memory
  - Less MPI communication and lower the parallel memory overcost


graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch
(ordering & amalgamation)
Fax
(block symbolic factorization)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
Distributed solution
Keep a N.D. ordering (NO RCM)
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch
(ordering & amalgamation)
iFax
(incomplete block S.F.)
Blend
(~~refinement~~ & mapping)
Sopalin
(factorizing & solving)
C.G.
GMRES
Distributed solution


graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch
(ordering & amalgamation)
Fax
(block symbolic factorization)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
Distributed solution
Main modification
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch
(ordering & amalgamation)
iFax
(incomplete block S.F.)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
C.G.
GMRES
Distributed solution


graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch (ordering & amalgamation)
Fax (block symbolic factorization)
Blend (refinement & mapping)
Sopalin (factorizing & solving)
Distributed solution
Few modifications
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch (ordering & amalgamation)
iFax (incomplete block S.F.)
Blend (~~refinement~~ & mapping)
Sopalin (factorizing & solving)
C.G.
GMRES
Distributed solution

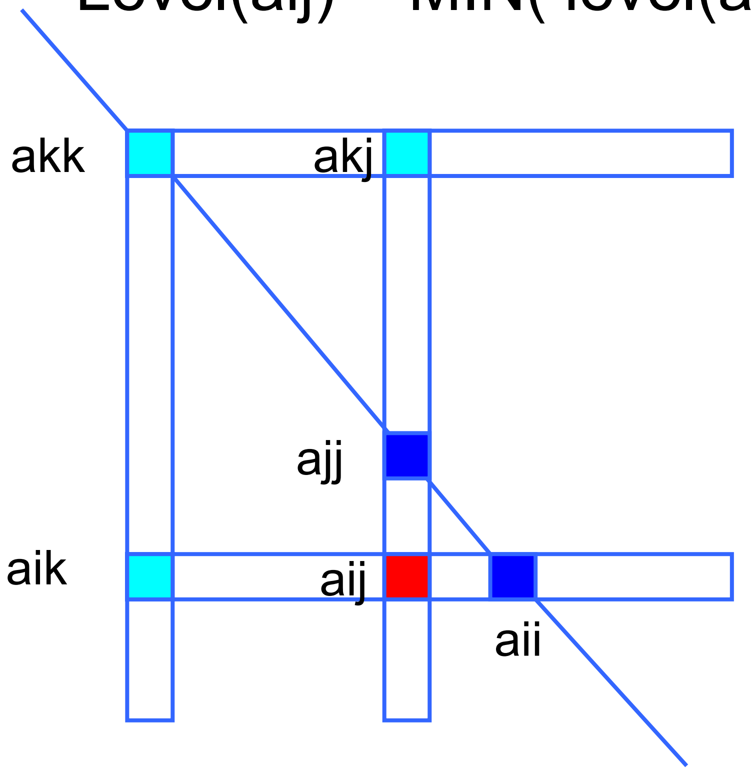
Outlines

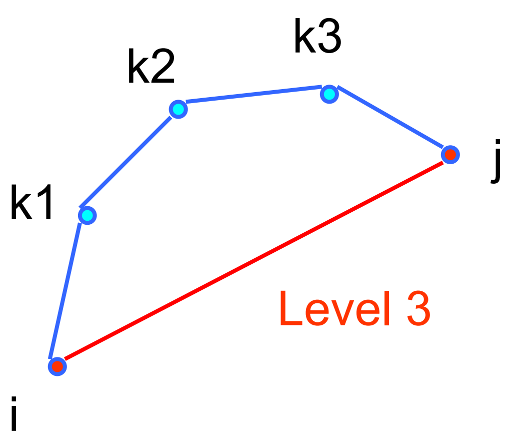
- Which modifications in the direct solver?
- ➢ The symbolic incomplete factorization
- An algorithm to get dense blocks in ILU
- Experiments
- Conclusion

Level based ILU(k)

- Scalar formulation of the level-of-fill:
  Non zero entries of A have a level 0.
  Consider the elimination of the kth unknowns during the fact. then:

  Level(aij) = MIN(level(aij) , level(aik)+level(akj)+1)





k1, k2, k3 < i and j

Level based ILU(k)

- The scalar incomplete factorization have the same asymptotical complexity than the Inc. Fact.

- BUT: it requires much less CPU time

- D. Hysom and A. Pothen gives a practical algorithm that can be easily // (based on the search of elimination^{kj} paths of length <= k+1) [Level Based Incompleted Factorization: Graphs model and Algorithm (2002)]

Level based ILU(k)

- In a FEM method a mesh node corresponds to several Degrees Of Freedom (DOF) and in this case we can use the node graph instead of the adj. graph of A i.e. :

  $Q(G,P) \rightarrow Q(G,P)^k = Q(G^k,P)$ P = *partition of mesh nodes*

- This means the symbolic factorization will have a complexity in the respect of the number of nodes whereas the factorization has a complexity in respect to the number of DOF.

Outlines

- Which modifications in the direct solver?
- The symbolic incomplete factorization
- ➢ An algorithm to get dense blocks in ILU
- Experiments
- Conclusion

How to build a dense block structure in ILU(k) factors ?

- First step: find the exact supernode partition in the ILU(k) NNZ pattern

- In most cases, this partition is too refined (dense blocks are usually too small for BLAS3)

- Idea: we allow some extra fill-in in the symbolic factor to build a better block partition

➢ Ex: How can we make bigger dense blocks if we allow 20% more fill-in ?

How to build a dense block structure in ILU(k) factors ?

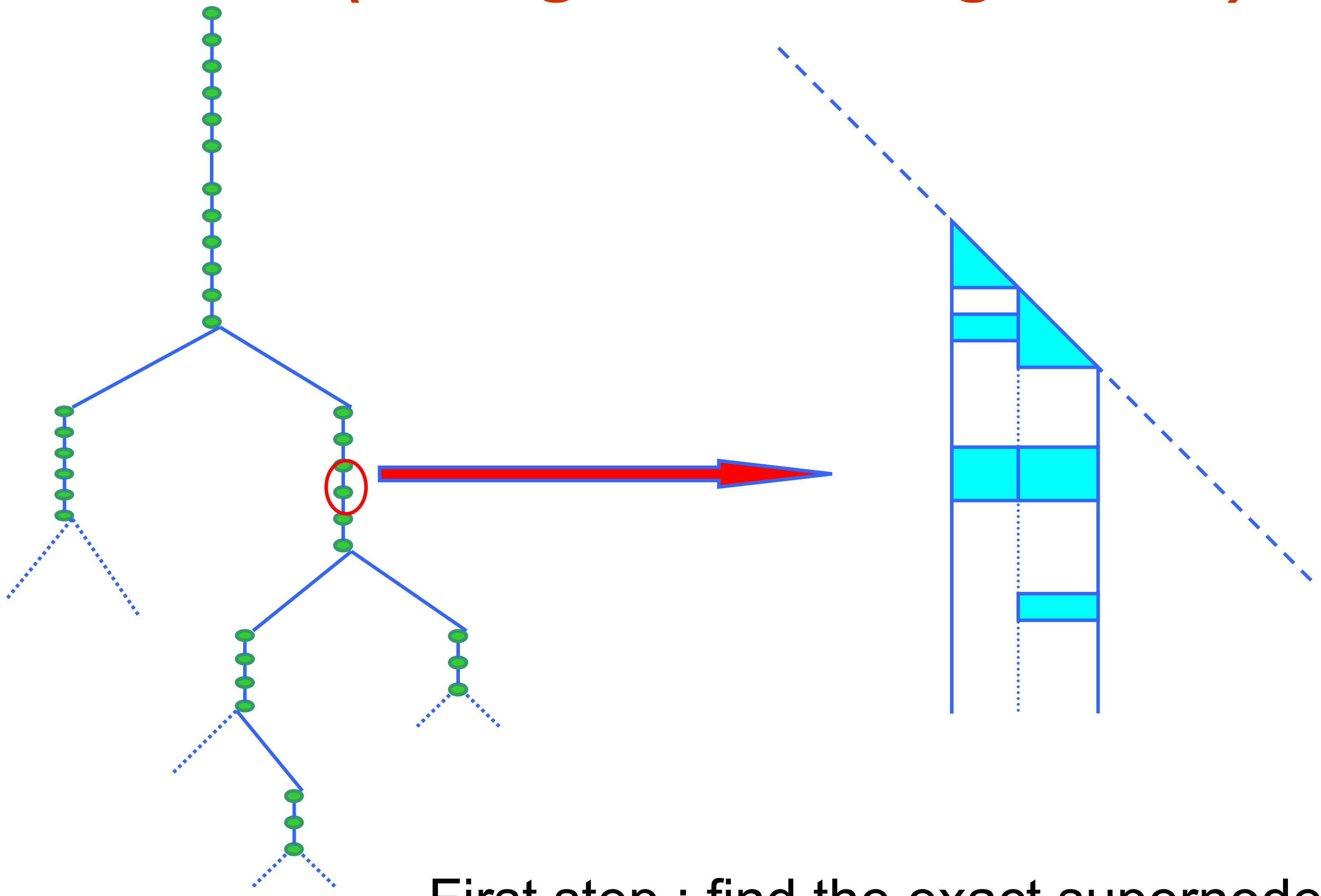
- We imposed some constraints:

➢ any permutation that groups columns with similar NNZ pattern should not affect G^k

➢ any permutation should not destroy the elimination tree structure

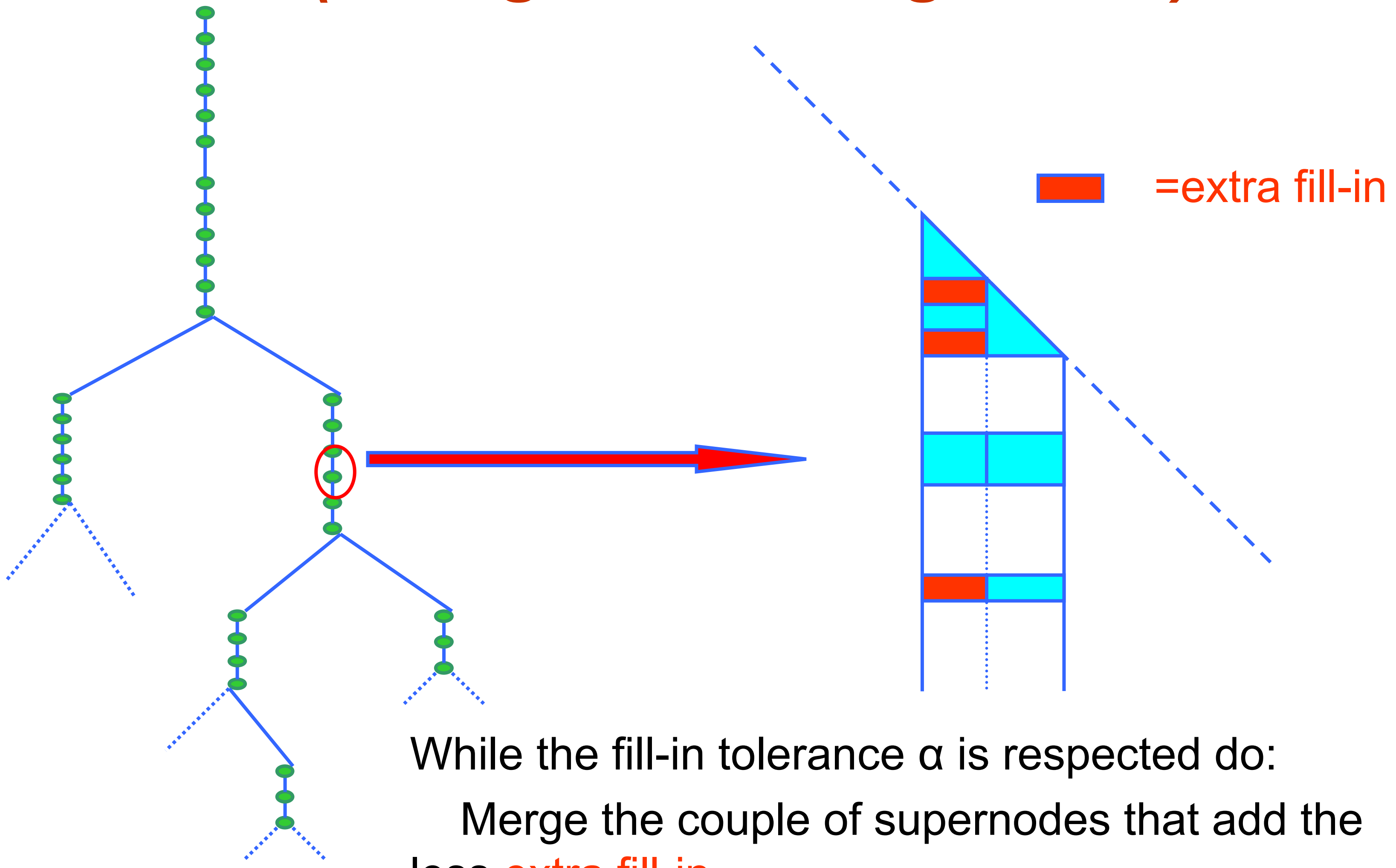
=> We impose the rule « merge only with your father… » for the supernode

Finding an approximated Supernodes Partition (amalgamation algorithm)



First step : find the exact supernode partition

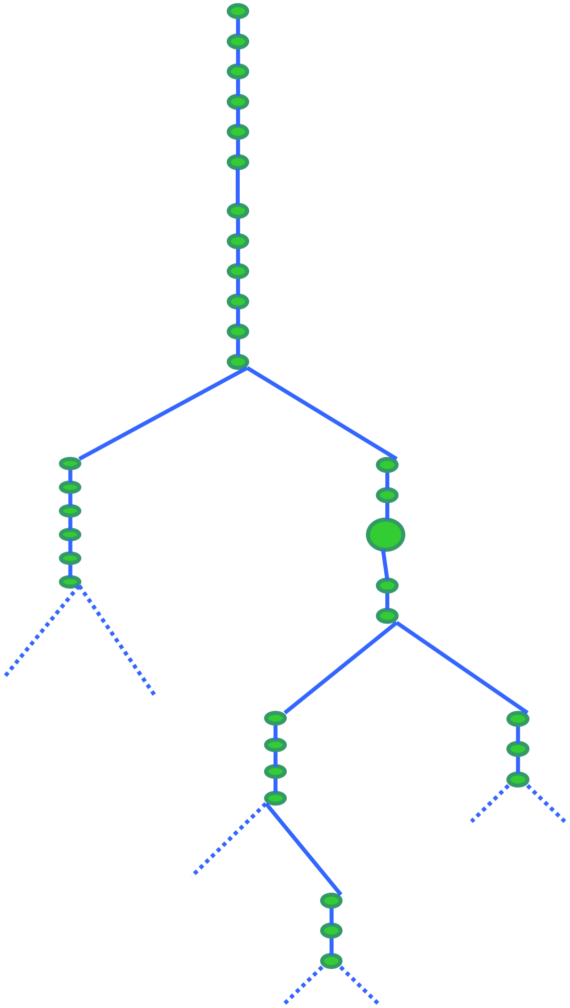
Finding an approximated Supernodes Partition (amalgamation algorithm)

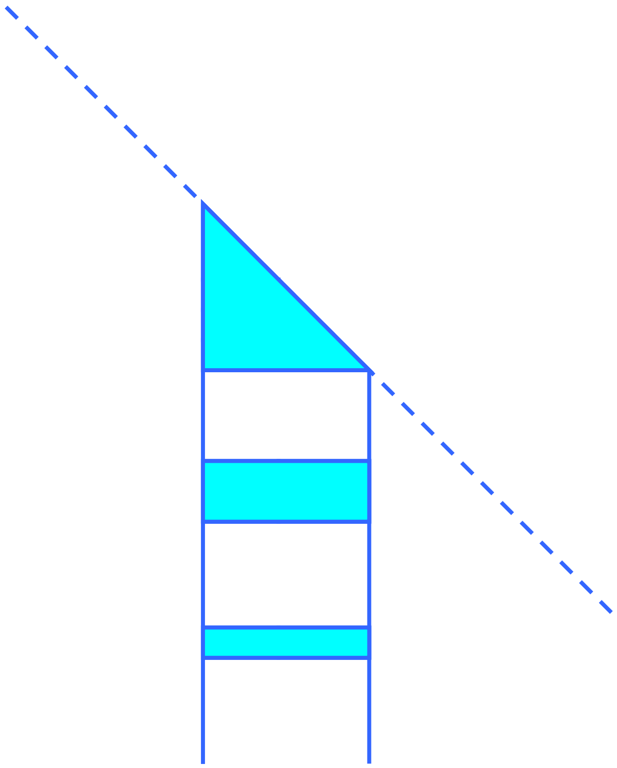


While the fill-in tolerance α is respected do:

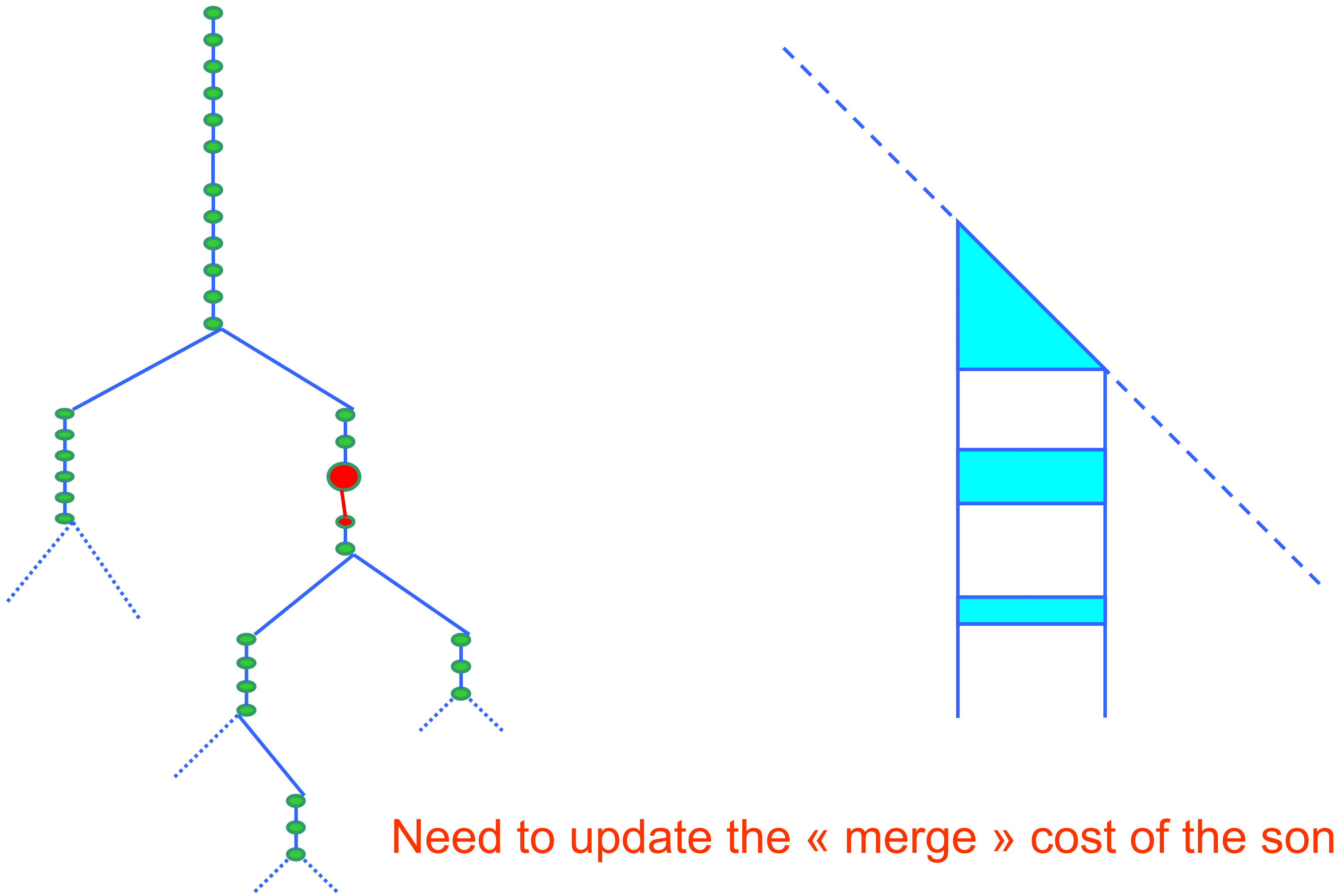
Merge the couple of supernodes that add the less extra fill-in

Finding an approximated Supernodes Partition (amalgamation algorithm)



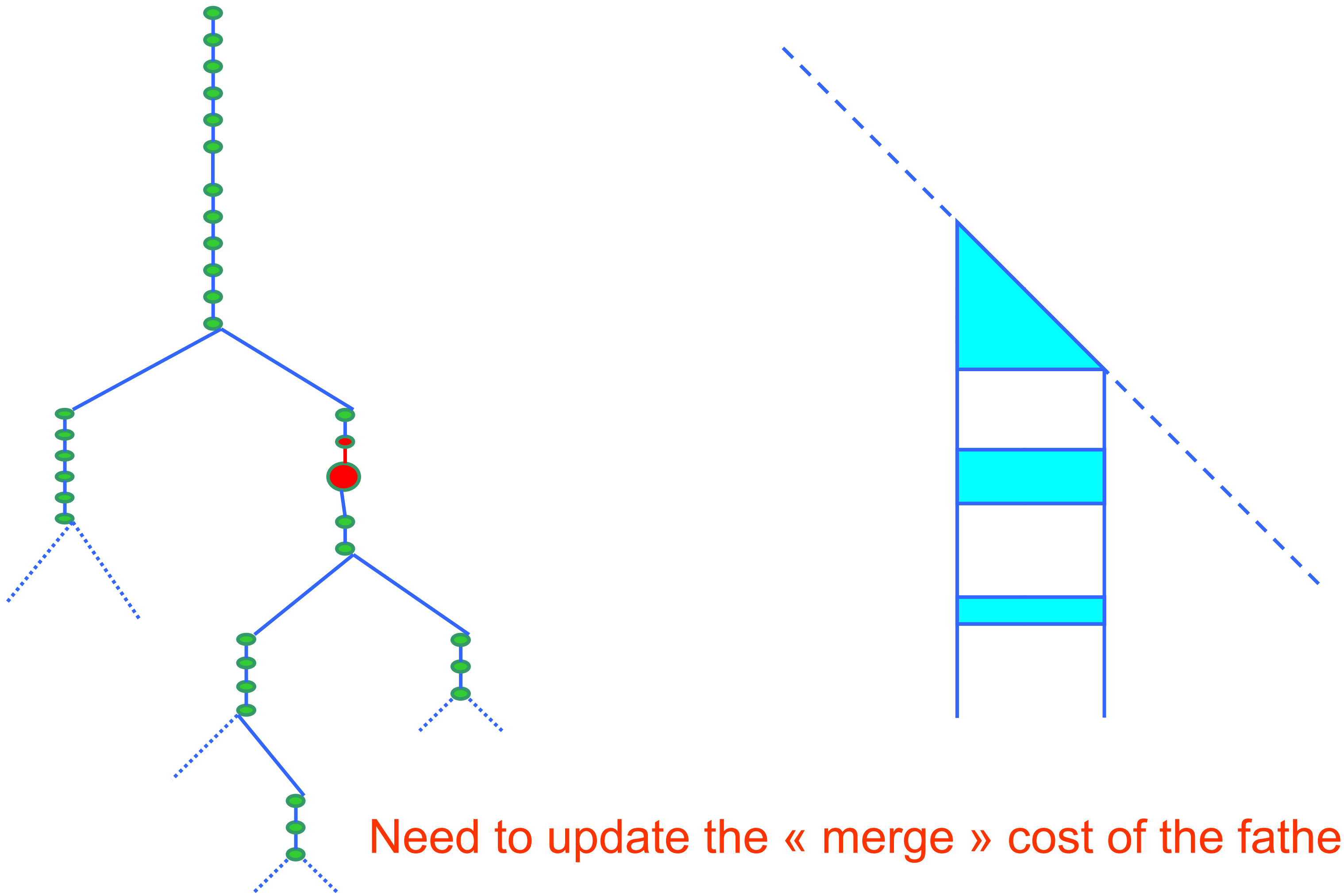


Finding an approximated Supernodes Partition (amalgamation algorithm)



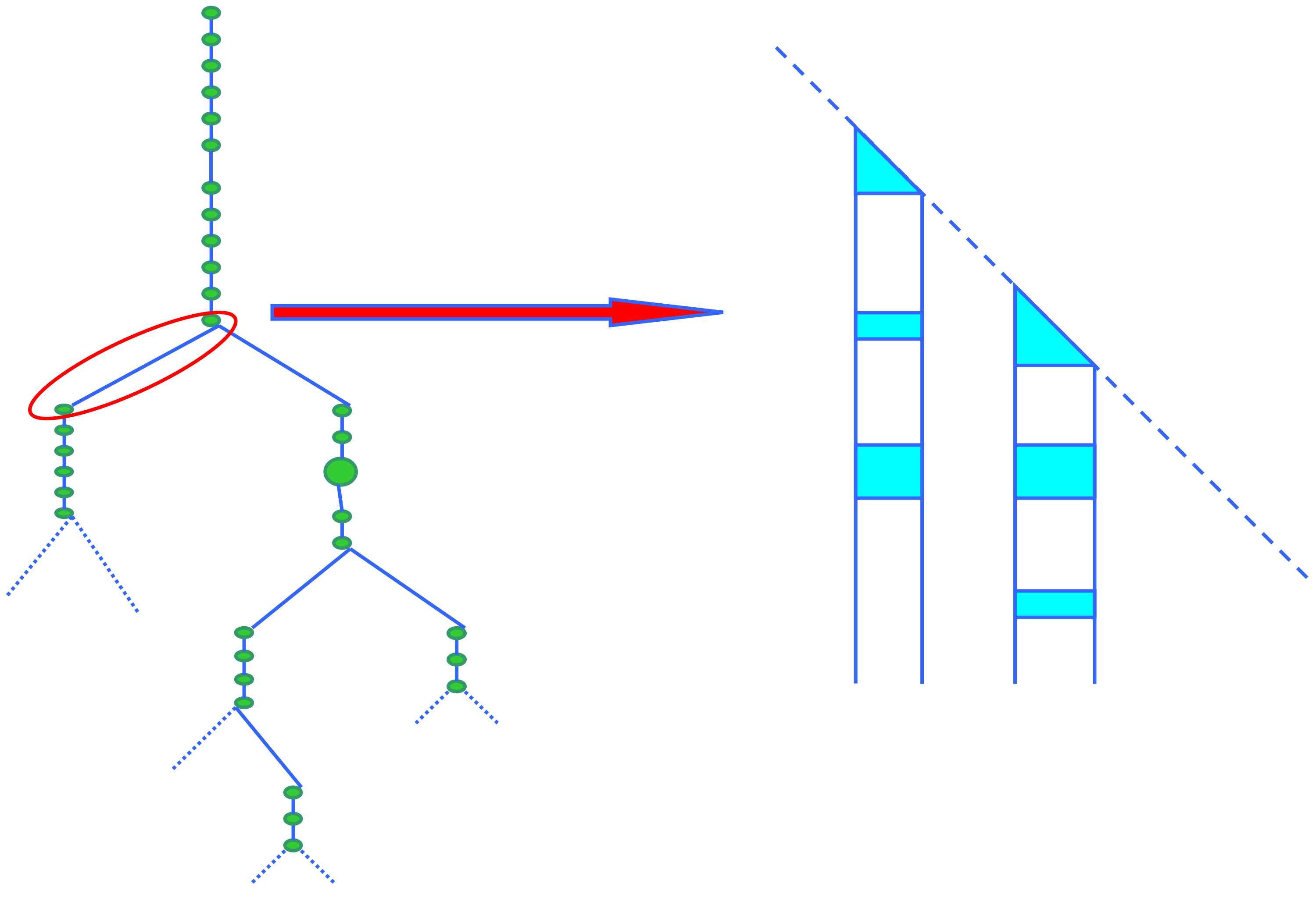
Need to update the « merge » cost of the son

Finding an approximated Supernodes Partition (amalgamation algorithm)

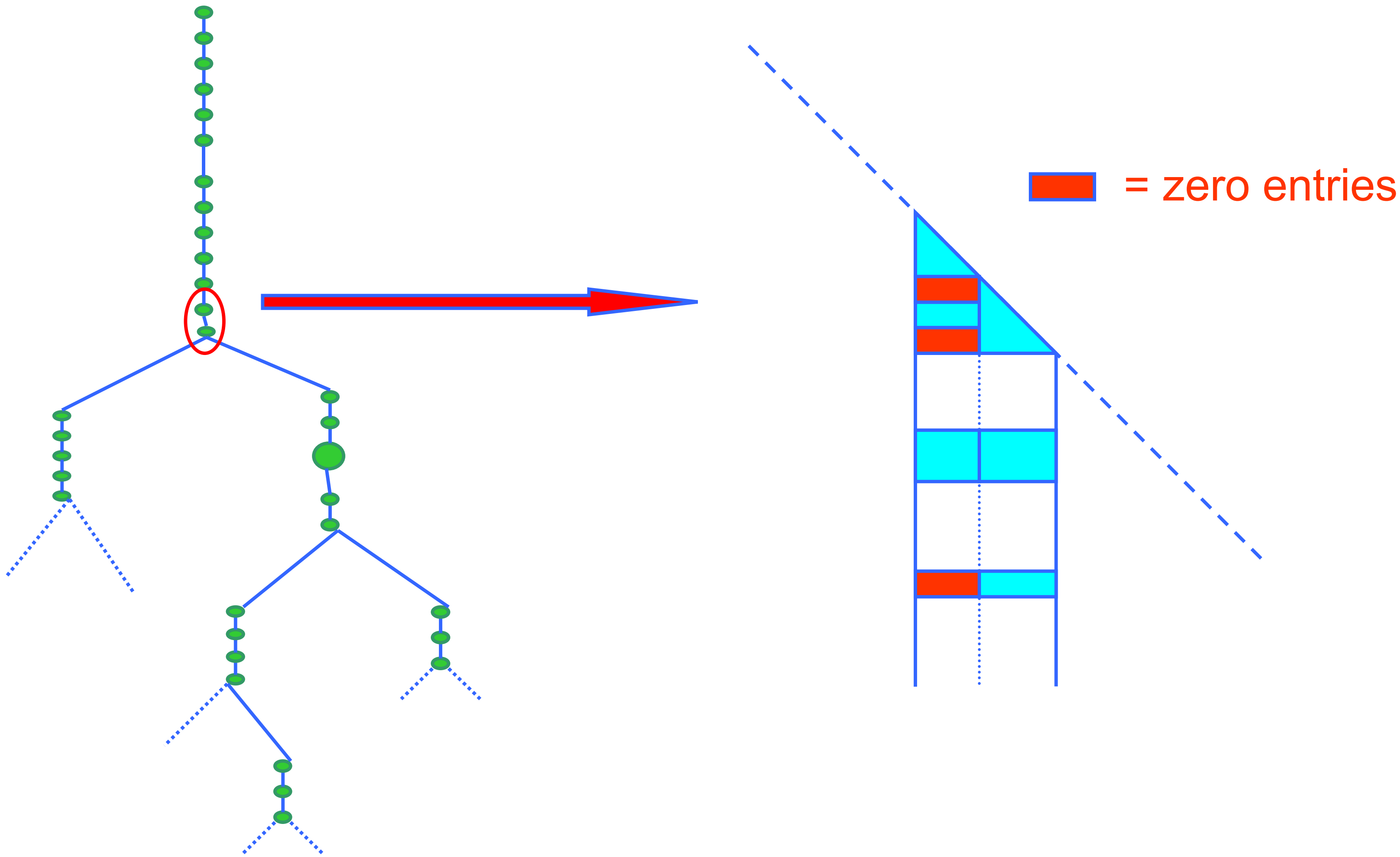


Need to update the « merge » cost of the father

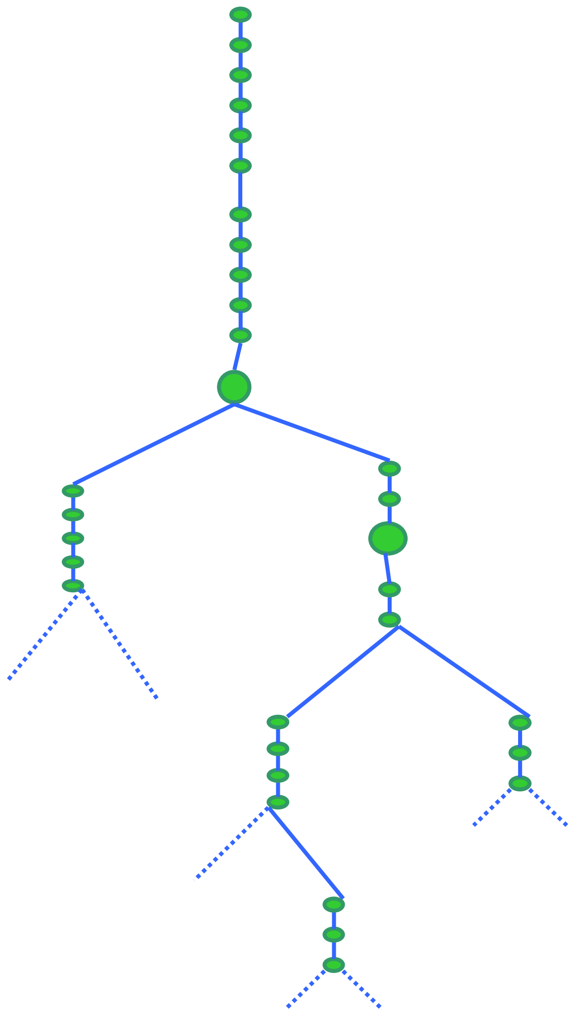
Finding an approximated Supernodes Partition (amalgamation algorithm)

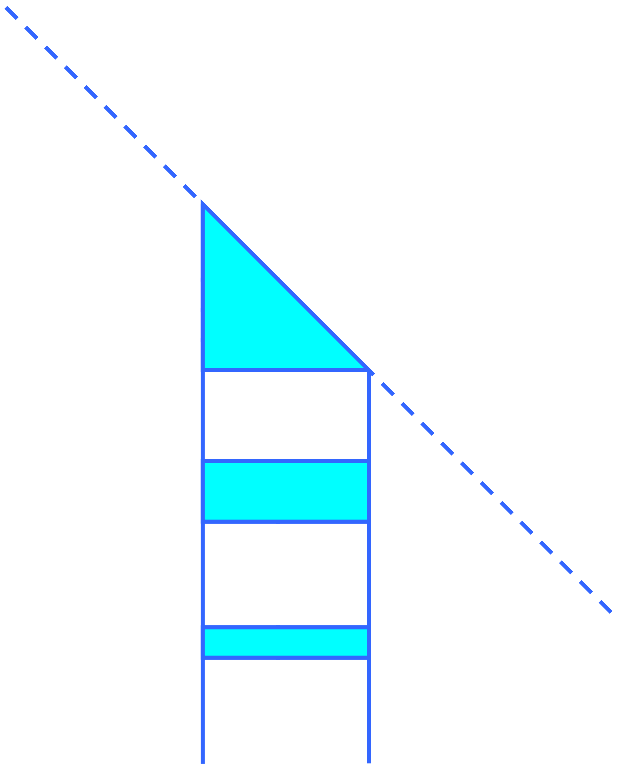


Finding an approximated Supernodes Partition (amalgamation algorithm)

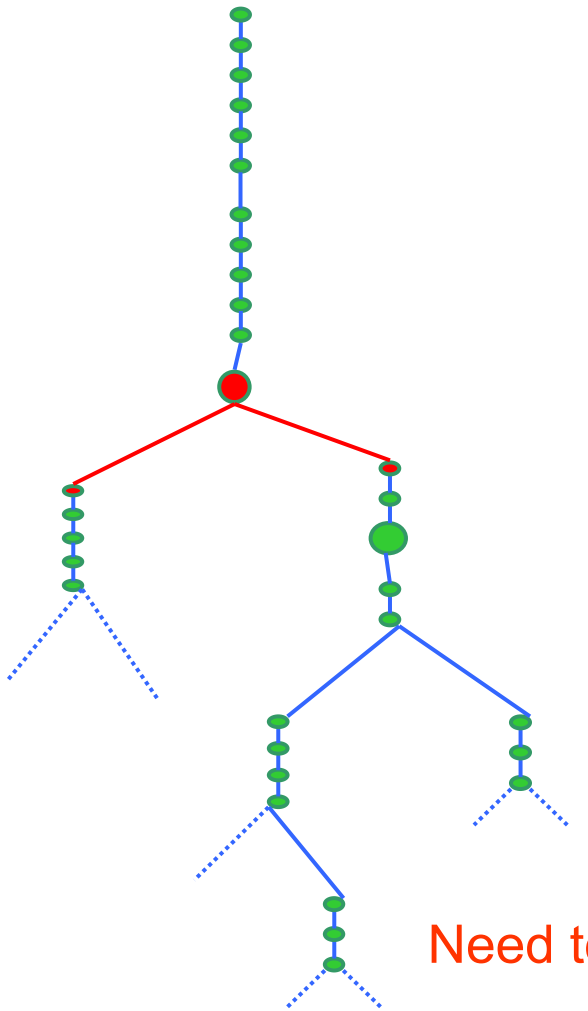


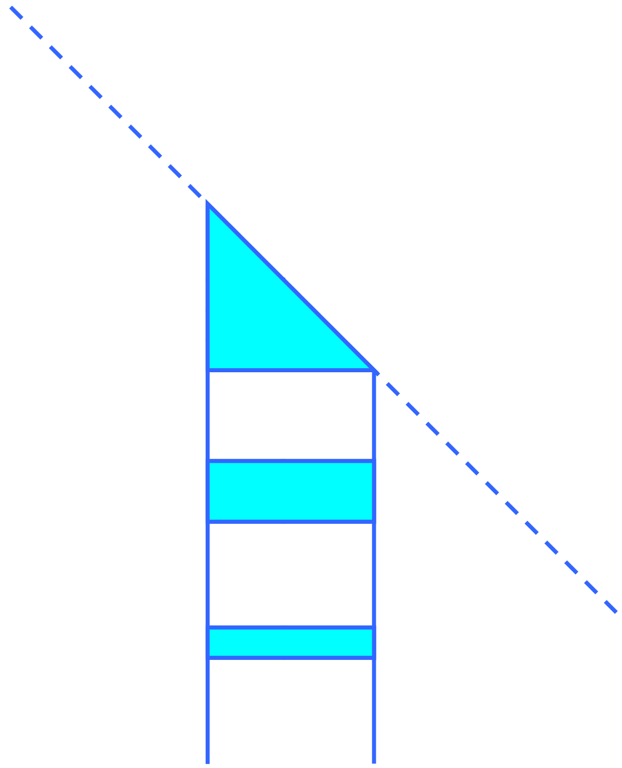
Finding an approximated Supernodes Partition (amalgamation algorithm)





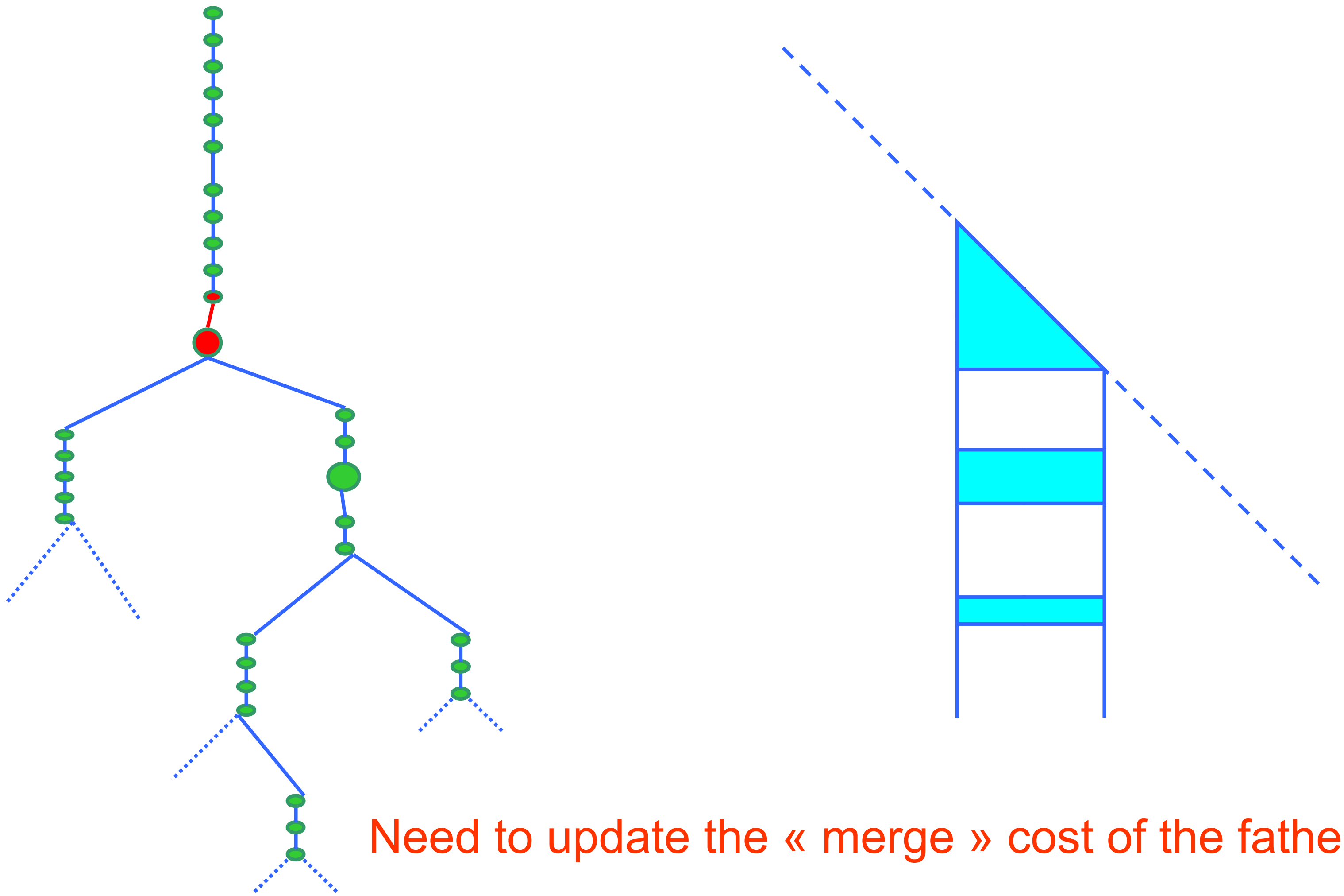
Finding an approximated Supernodes Partition (amalgamation algorithm)





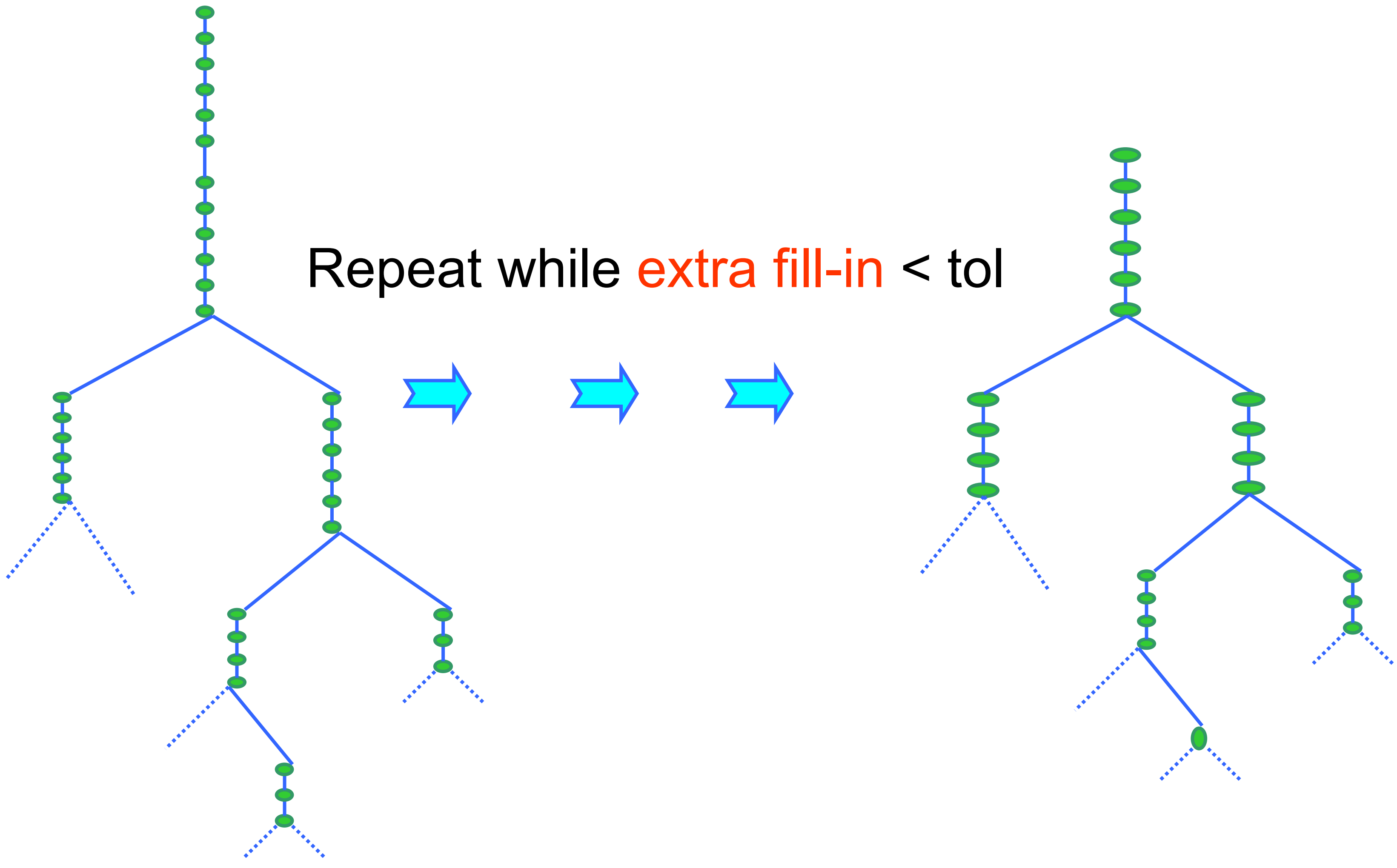
Need to update the « merge » cost of the sons

Finding an approximated Supernodes Partition (amalgamation algorithm)



Need to update the « merge » cost of the father

Finding an approximated Supernodes Partition (amalgamation algorithm)



Cost of the algorithm

- The approximate supernode merging algorithm is really cheap compare to the other steps

- At each step: recompute fill-add for modified (son-father) couples and maintain the heap sort.

- Complexity bound by $O(D.N_0 + N_0.Log(N_0))$
  N_0 : number of exact supernodes in ILU factors
  D : maximum number of extradiagonal blocks in a block-column

Numerical experiments

- Results on IBM power5 + Switch “Federation”
- All computations were performed in double precision
- Iterative accelerator was GMRES (no restart)
- Stopping criterion for iterative accelerators was a relative residual norm (||b-A.x||/||b||) of 1e-7

Test cases:

- AUDIKW_1 : Symmetric matrix (Parasol collection)
  n = 943,695 nnz(A) = 39,297,771
  With direct solver : nnz(L) = 31 x nnz(A)
  total solution in 115s on 16 procs
  ➔ 3D

- SHIPSEC5 : Symmetric matrix (Parasol collection)
  n = 179,860 nnz(A) = 4,966,618
  With direct solver : nnz(L) = 11 x nnz(A)
  total solution in 7s on 16 procs
  ➔2D

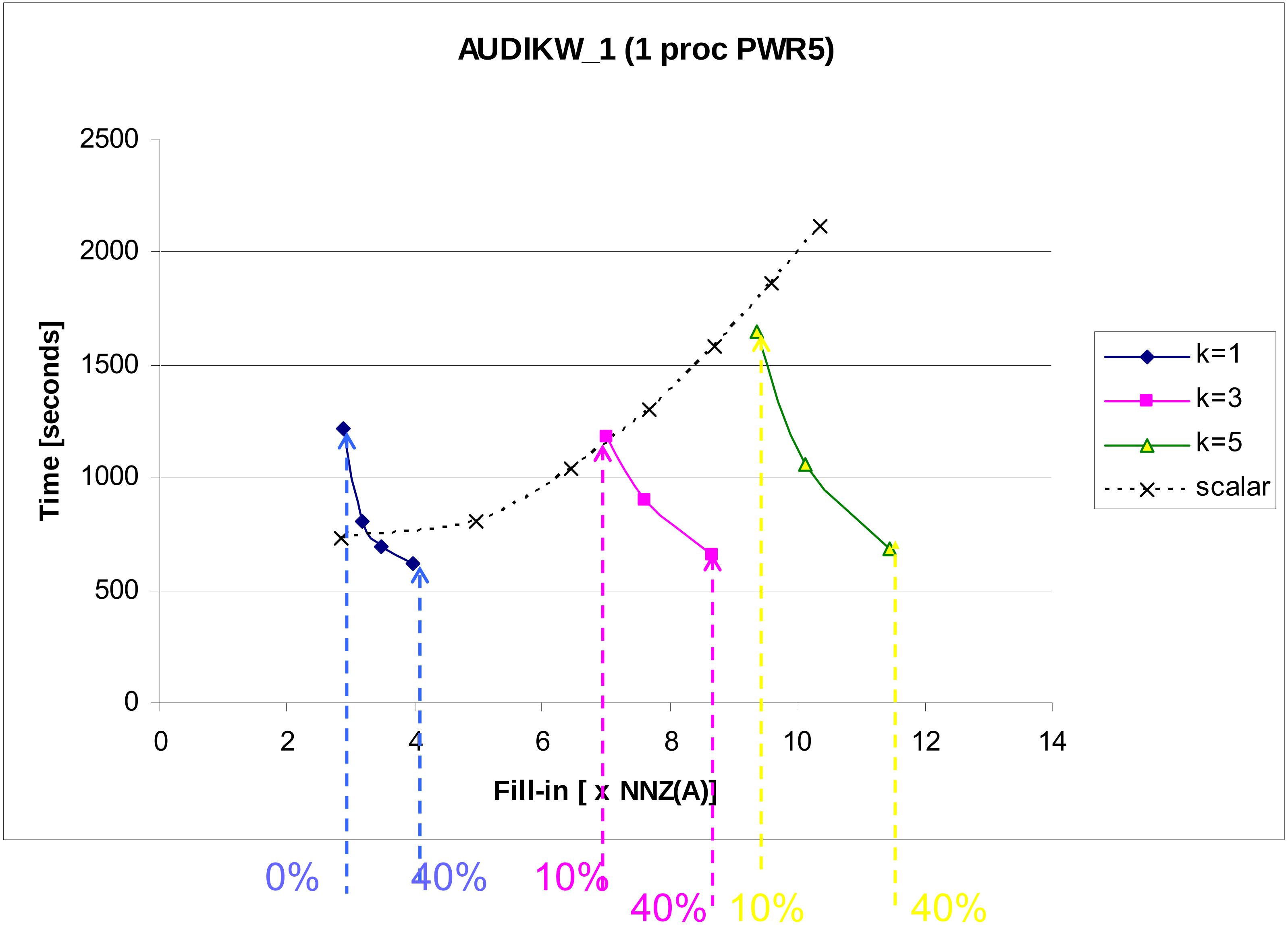
Effect of amalgamation ratio α

AUDIKW_1: n = 943,695 nnzA=39,297,771

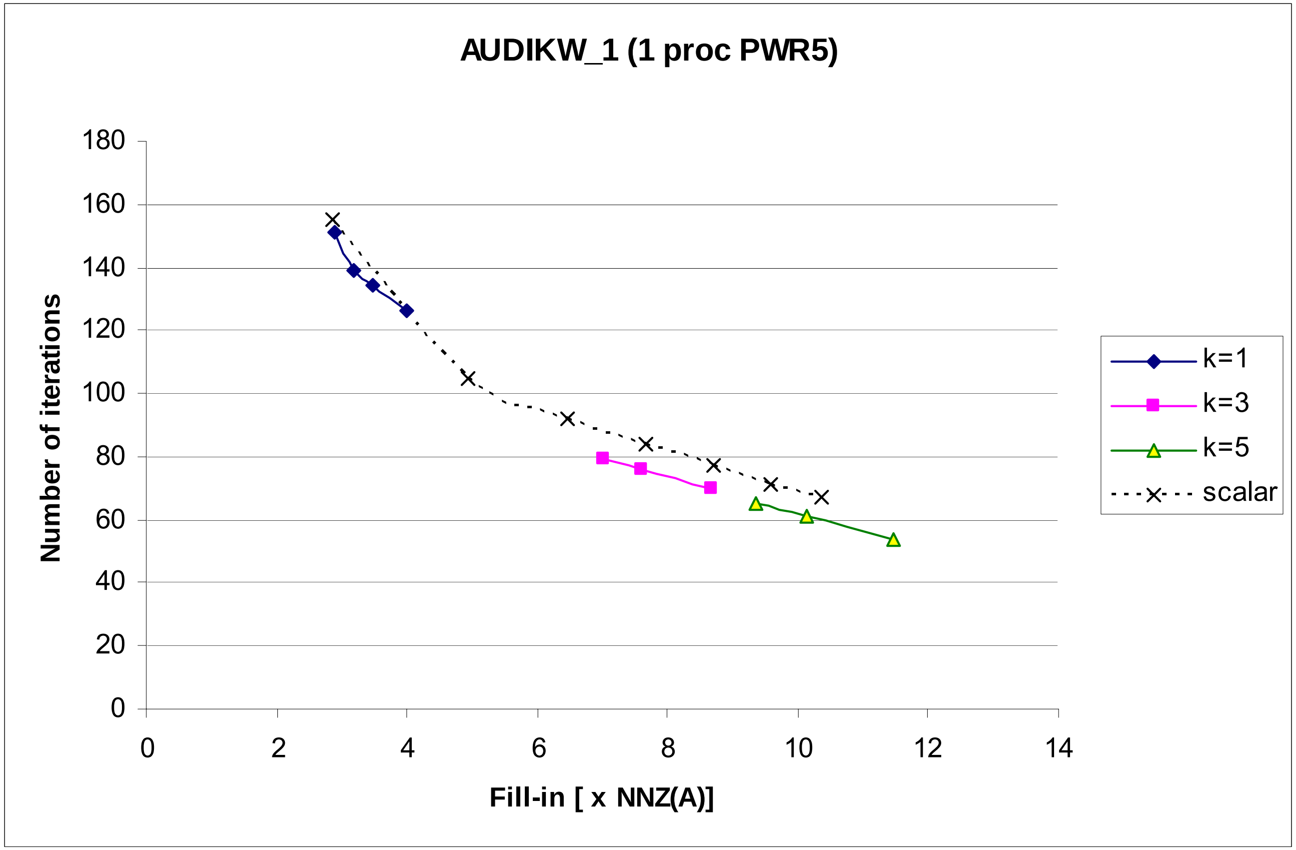
K	α	CBLK	BLOCKS	Amalg	Fact.	Tr. solve
1	0%	300,386	11,893,366	4.74	167.19	6.94
1	20 %	133,102	4,422,368	8.18	71.72	4.67
1	40 %	83,168	2,564,865	9.59	53.10	4.50
3	0%	292,096	27,099,992	8.63		
3	20 %	85,759	6,255,623	14.18	293.33	7.96
3	40 %	41,515	2,278,474	15.71	163.88	7.00
5	0%	275,012	35,399,482	11.04		
5	20 %	62,203	6,453,393	17.23	518.57	8.86
5	40 %	27,915	1,890,939	19.00	258.11	7.80

Sequential Time: total fact. + solve.

Res. precision= 1e-7



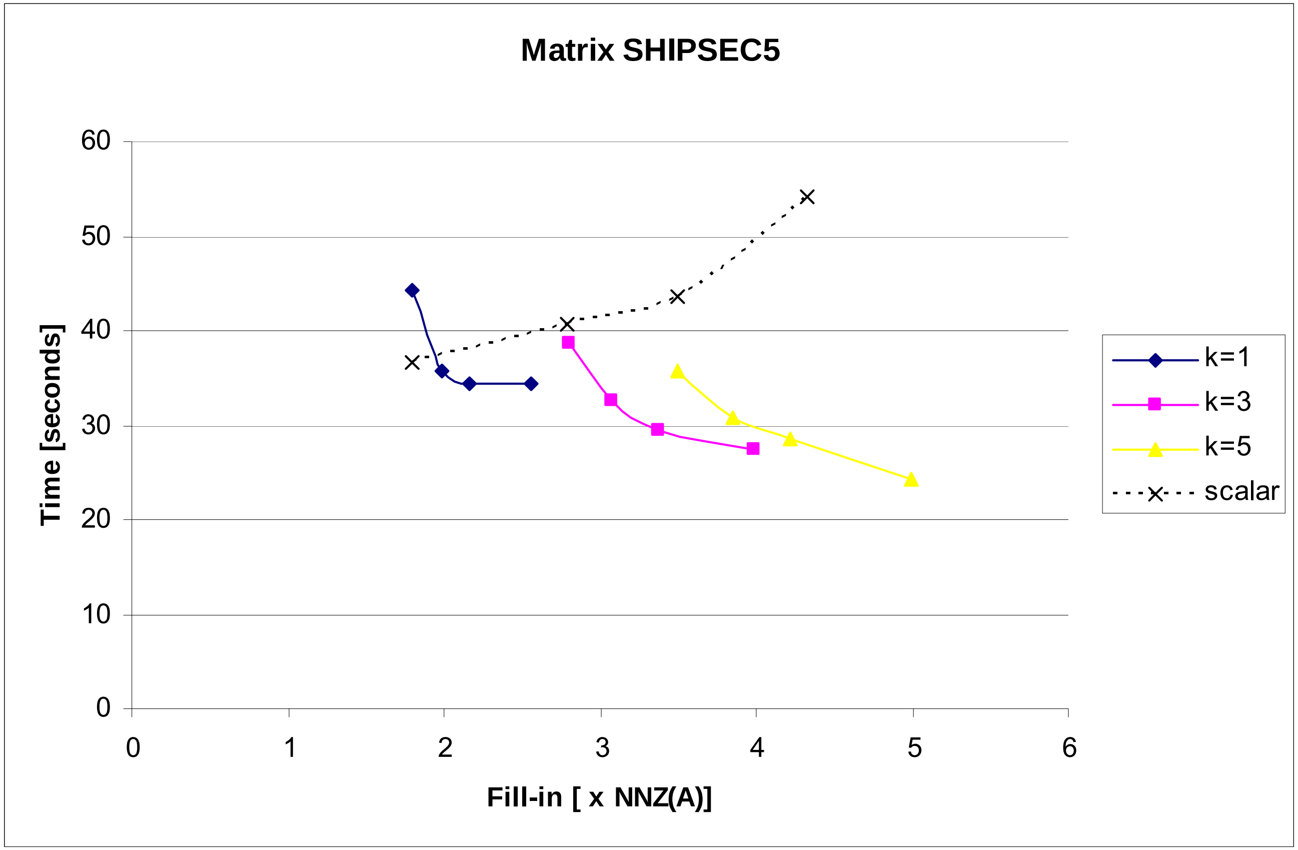
Number of Iterations



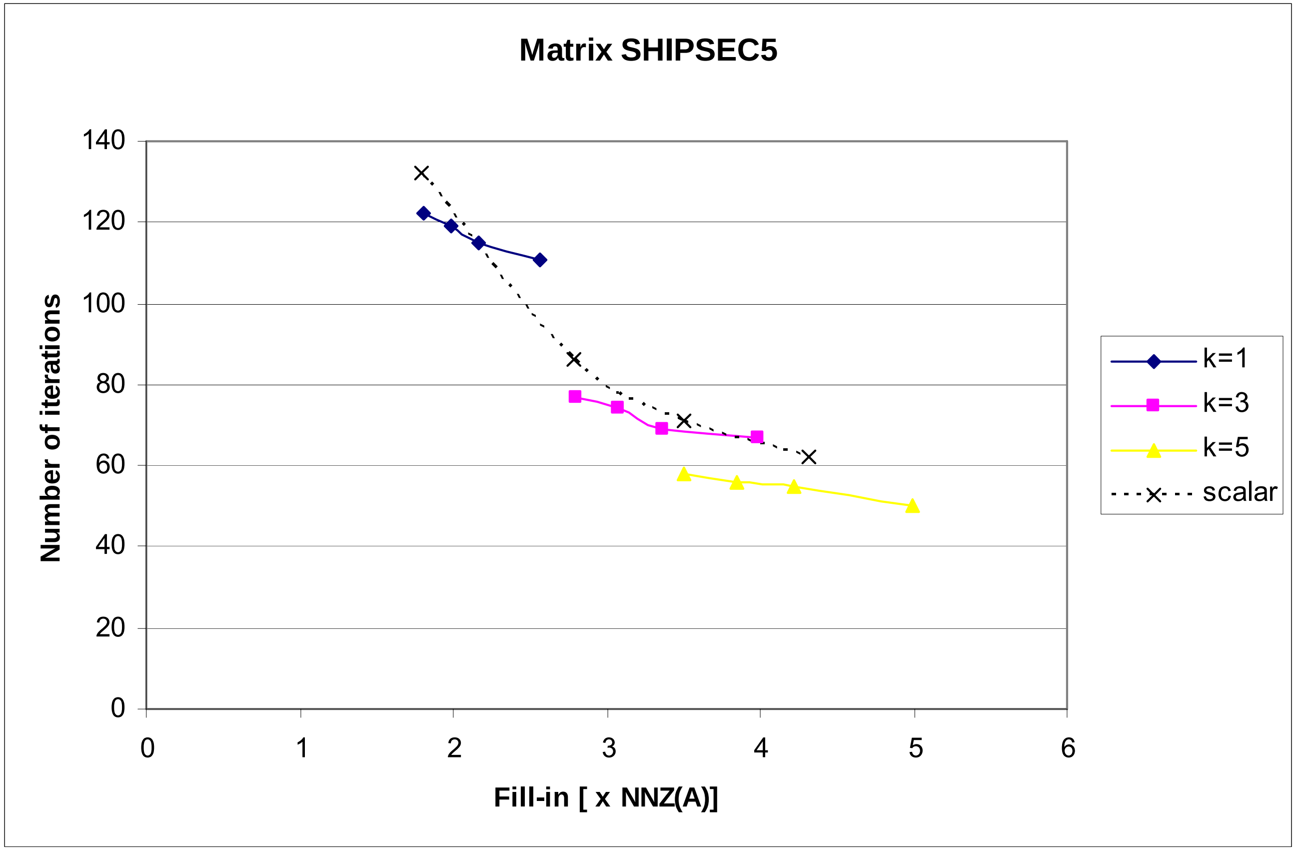
Parallel Time: AUDIKW_1

		1 processor			16 processors		
K	α	Fact	TR solv	Total	Fact	TR Solv	Total
1	20 %	74.5	4.59	690.1	21.4	0.51	91.5
1	40 %	56.4	4.44	620.3	12.7	0.42	67.0
3	20 %	331.1	7.97	936.8	39.2	0.91	108.7
3	40 %	194.6	7.57	732.0	18.6	0.66	65.7
5	20 %	518.5	8.86	1058.9	52.3	1.16	123.1
5	40 %	258.1	7.80	679.3	21.2	0.78	63.3

Sequential Time: total fact. + solve.



Number of Iterations



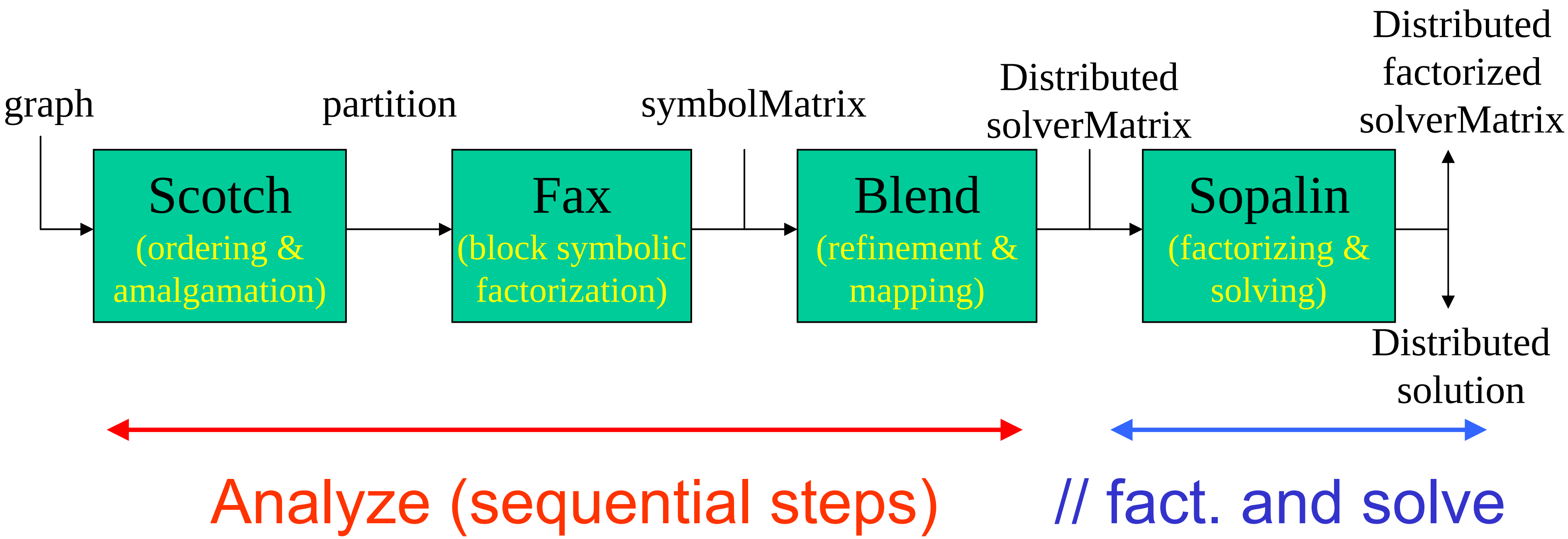
Conclusion

- ⇒This method provides an efficient parallel implementation of ILU(k) precon. (and does not depends on the numbers of proc.)
- ⇒The amalg. algorithm could be improved by relaxing the constraint of the « merge only with your father » but this requires further modifications in the solver chain.

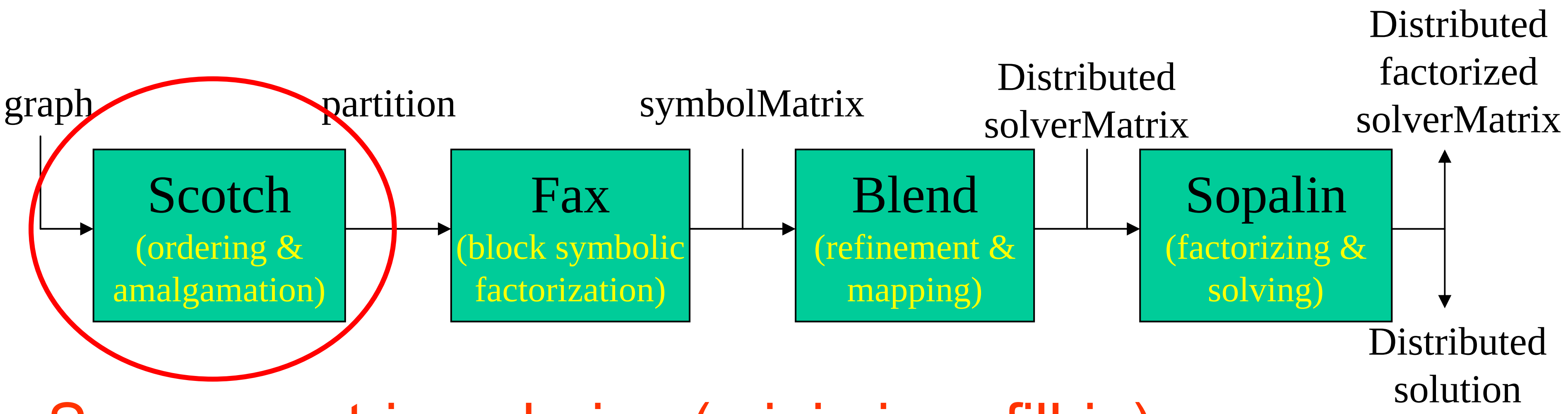
Symbolic ILU(k) Audikw_1

Level of fill	Symb. Facto.	Num. Fact.
K=1	16.8	74.97
K=3	73.8	466.94
K=5	131.11	1010.4

Direct solver chain (in PaStiX)



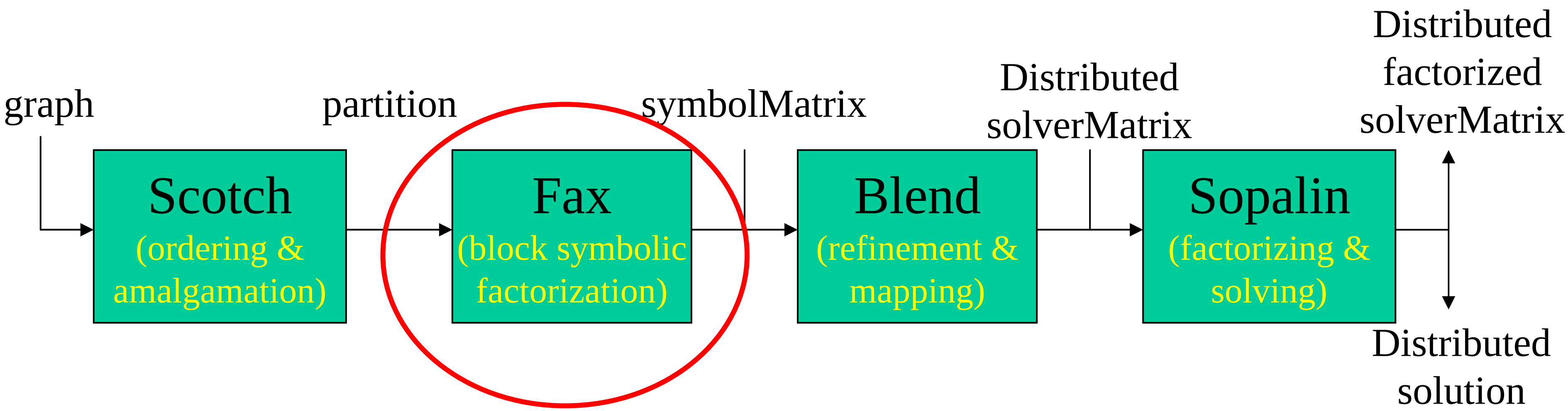
Direct solver chain (in PaStiX)



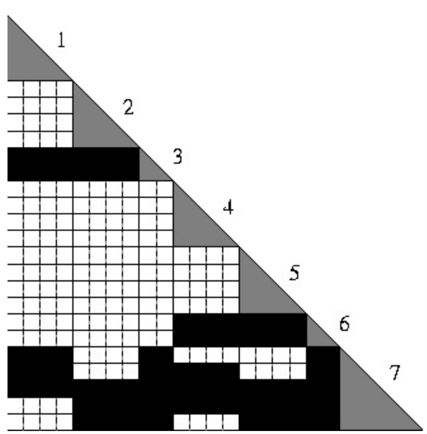
Sparse matrix ordering (minimizes fill-in)

- Scotch: an hybrid algorithm
  - incomplete Nested Dissection
  - the resulting subgraphs being ordered with an Approximate Minimum Degree method under constraints (HAMD)

Direct solver chain (in PaStiX)

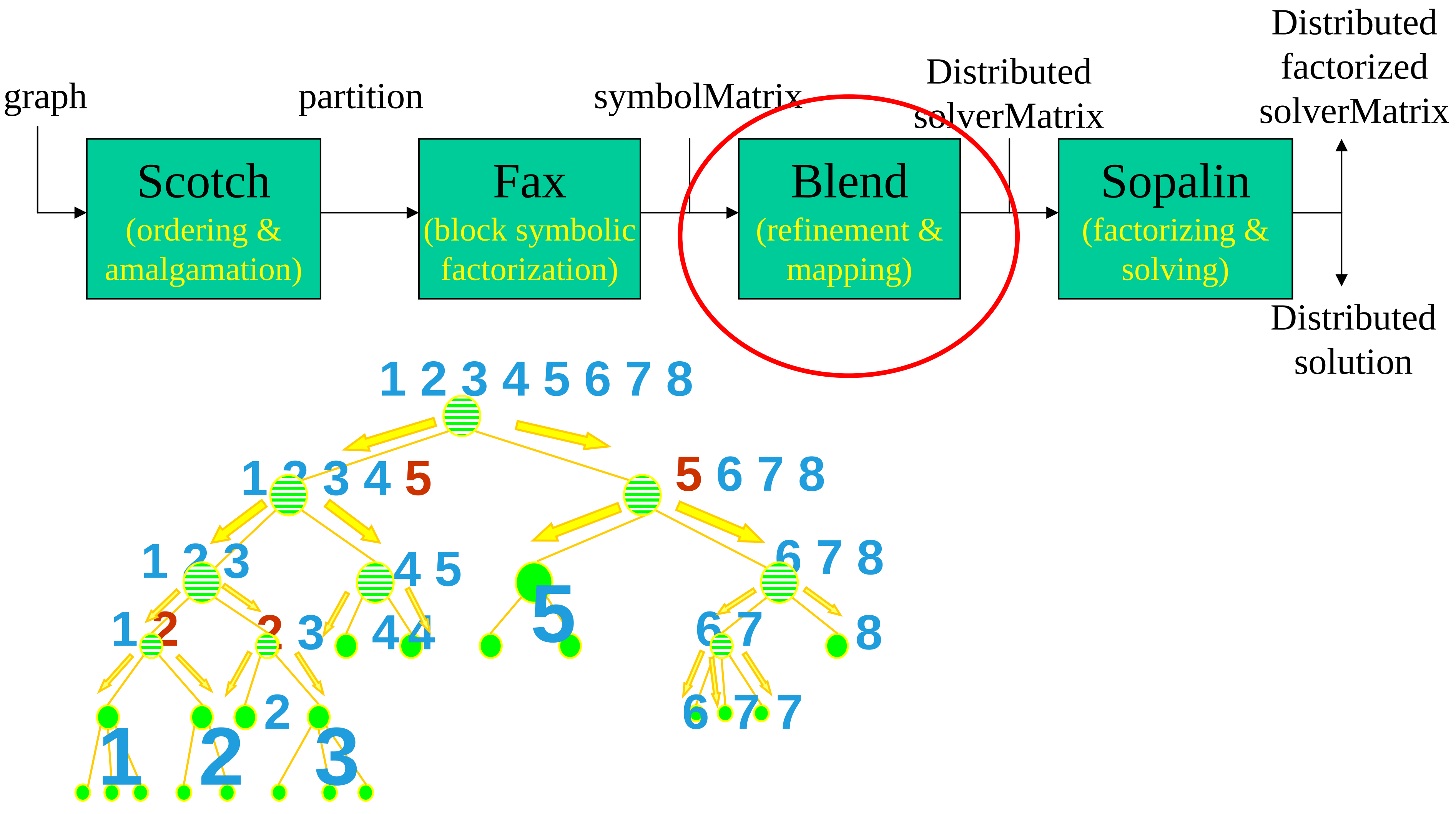


The symbolic block factorization

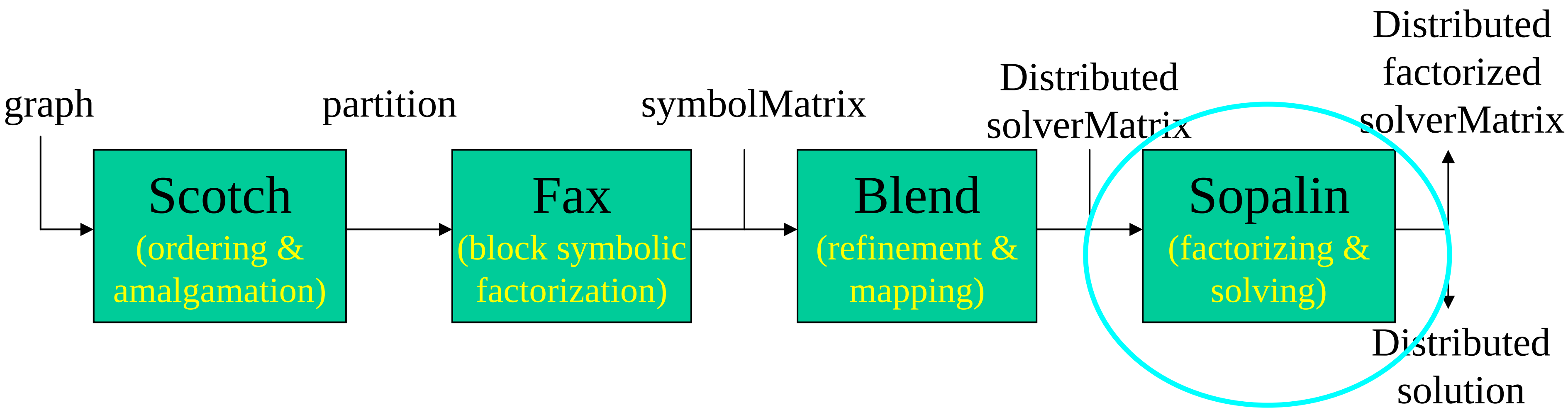


- ***Q(G,P)→Q(G,P)*=Q(G*,P) => linear in number of blocks!***
- Dense block structures
  ⇒ only a extra few pointers to store the matrix

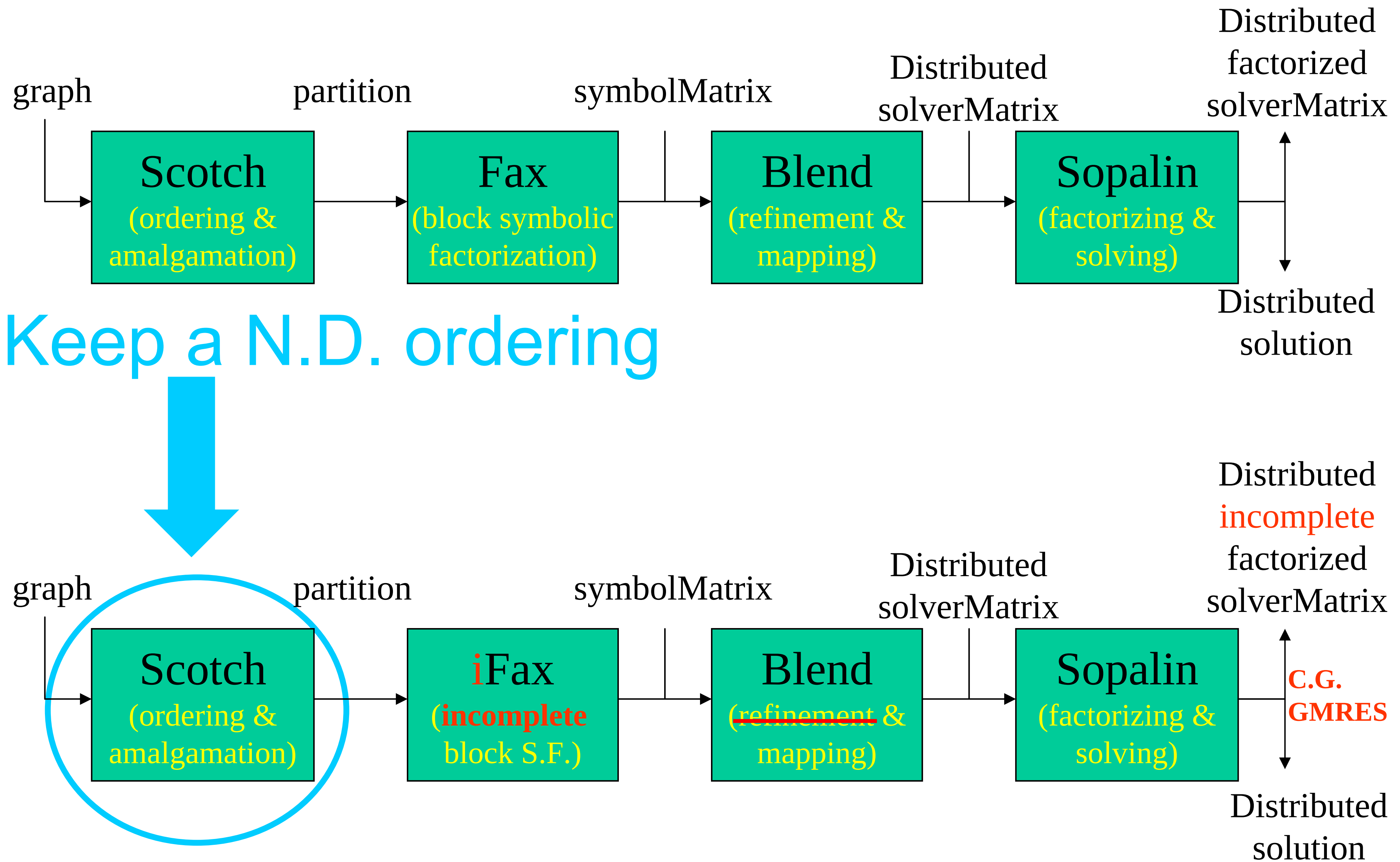
Direct solver chain (in PaStiX)

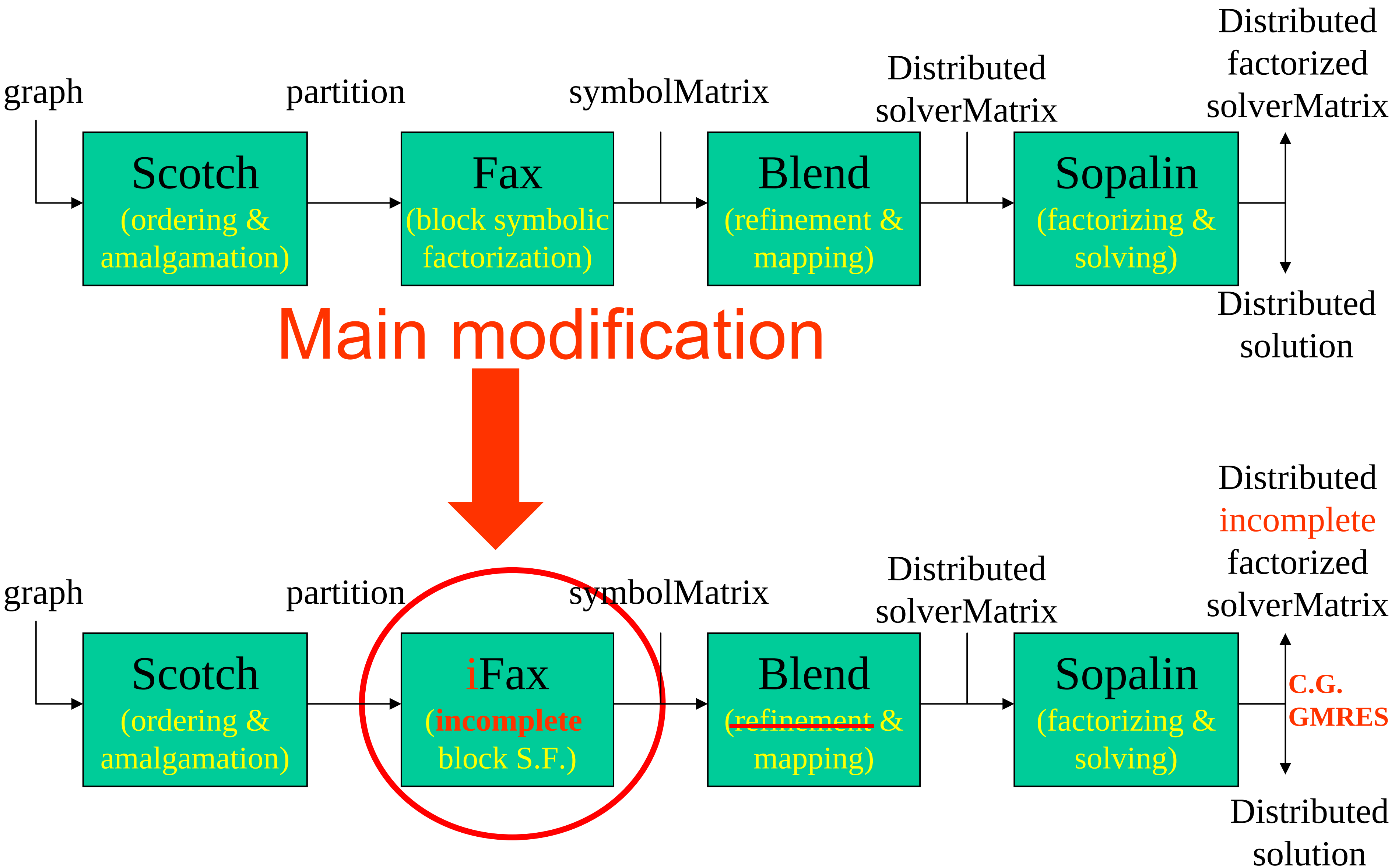


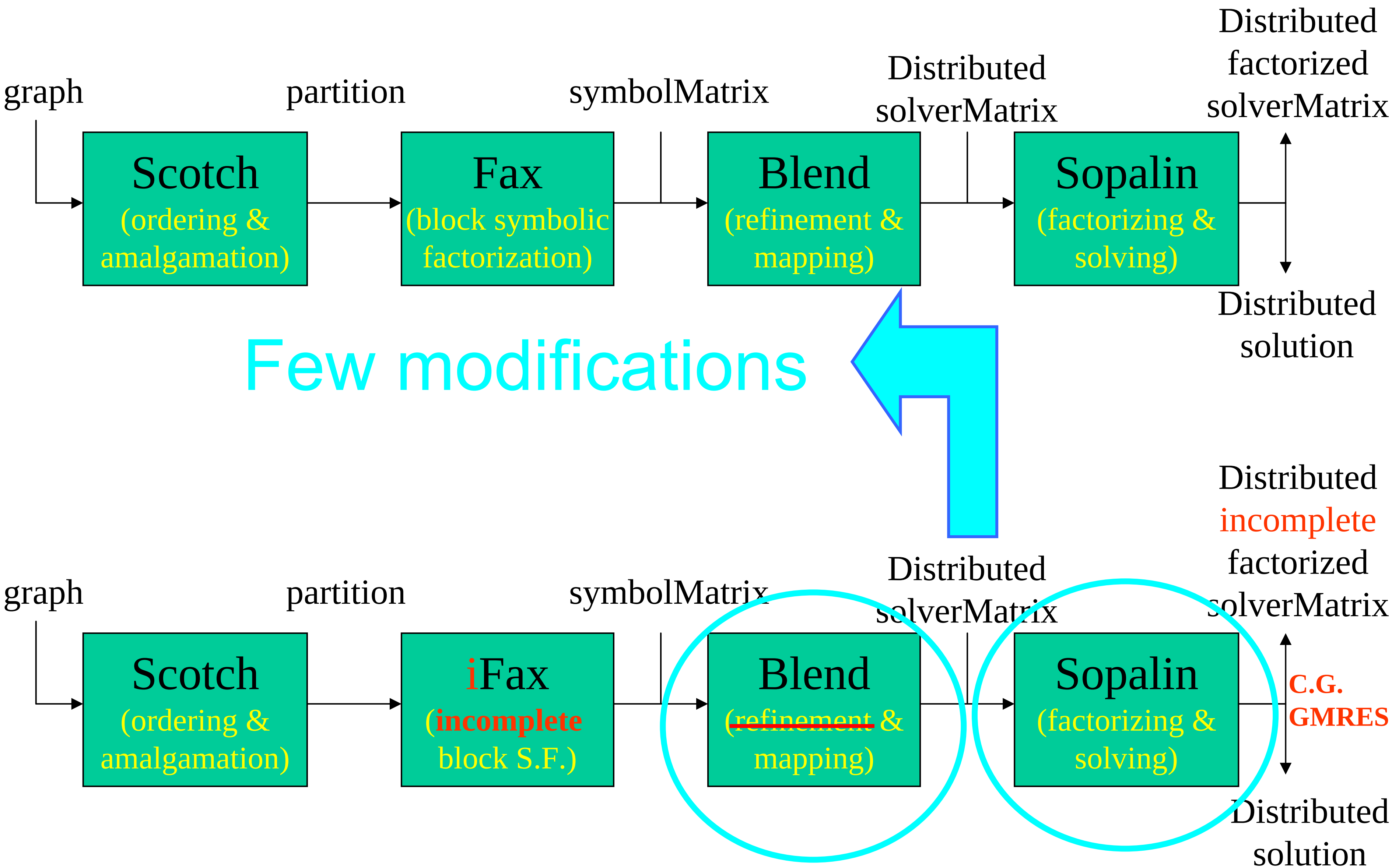
Direct solver chain (in PaStiX)



- Modern architecture management (SMP nodes) : hybrid Threads/MPI implementation (all processors in the same SMP node work directly in share memory
  - Less MPI communication and lower the parallel memory overcost

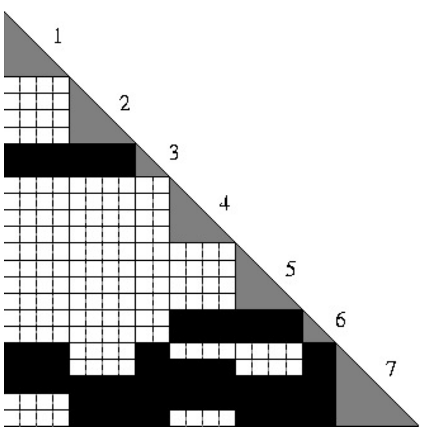

graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch
(ordering & amalgamation)
Fax
(block symbolic factorization)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
Distributed solution
Keep a N.D. ordering
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch
(ordering & amalgamation)
iFax
(incomplete block S.F.)
Blend
(~~refinement~~ & mapping)
Sopalin
(factorizing & solving)
C.G.
GMRES
Distributed solution


graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch
(ordering & amalgamation)
Fax
(block symbolic factorization)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
Distributed solution
Main modification
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch
(ordering & amalgamation)
iFax
(incomplete block S.F.)
Blend
(~~refinement~~ & mapping)
Sopalin
(factorizing & solving)
C.G.
GMRES
Distributed solution


graph
partition
symbolMatrix
Distributed solverMatrix
Distributed factorized solverMatrix
Scotch
(ordering & amalgamation)
Fax
(block symbolic factorization)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
Distributed solution
Few modifications
graph
partition
symbolMatrix
Distributed solverMatrix
Distributed incomplete factorized solverMatrix
Scotch
(ordering & amalgamation)
iFax
(incomplete block S.F.)
Blend
(refinement & mapping)
Sopalin
(factorizing & solving)
C.G.
GMRES
Distributed solution

Direct Factorization techniques

- Ordering to minimize fill-in and allow // is based upon ND
- Partition of supernodes P is found in 0(nnzA).
- ***Q(G,P)→Q(G,P)*=Q(G*,P)***
  => linear in number of blocks!
- Dense block structures
  ⇒ only a extra few pointers
  to store the matrix



Direct Factorization techniques

⇒ Manage parallelism induced by sparsity (block elimination tree).

⇒ Split and distribute the dense blocks in order to take into account the potential parallelism induced by dense computations .