Improvement of existing solvers for the simulation of MHD instabilities

Numerical Flow Models for Controlled Fusion
Porquerolles, April 2007

P. Hénon, P. Ramet
LaBRI, UMR CNRS 5800, Université Bordeaux I
Projet ScAlApplix, INRIA Futurs

O. Czarny, G. Huysmans
EURATOM/CEA, Controlled Fusion Rech. Dept., Cadarache

Context

- ANR CIS 2006 Adaptive MHD Simulation of Tokamak ELMs for ITER (ASTER)
- The ELM (Edge-Localized-Mode) is MHD instability which localised at the boundary of the plasma
- The energy losses induced by the ELMs within several hundred microseconds are a real concern for ITER

- The non-linear MHD simulation code JOREK is under development at the CEA to study the evolution of the ELM instability

- To simulate the complete cycle of the ELM instability, a large range of time scales need to be resolved to study:
  - The evolution of the equilibrium pressure gradient (~seconds)
  - ELM instability (~hundred microseconds)

➢ a fully implicit time evolution scheme is used in the JOREK code

- This leads to a large sparse matrix system to be solved at every time step.
- This scheme is possible due to the recent advances made in the parallelized direct solution of general sparse matrices (MUMPS, PaStiX, SuperLU, ...)

Context

- To reduce the large memory requirements of the sparse matrix solve, the PaStiX library is being extended to include an iterative solver which uses an incomplete factorization of the original matrix as a preconditioner [P. Hénon, P. Ramet, and J. Roman. On finding approximate supernodes for an efficient ILU(k) factorization, submitted to Parallel Computing]

- The resulting solver is a hybrid method which can greatly reduce the memory requirements compared to the direct solver
- But, iterative solvers are difficult to apply to the system of equations that result from the MHD simulation due to the extremely large condition number of the matrices

- The workpackage~1 of ASTER project has to define the scope where our hybrid solver can solve the MHD systems from JOREK in the relevant regime of very high magnetic and fluid Reynolds numbers

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…)

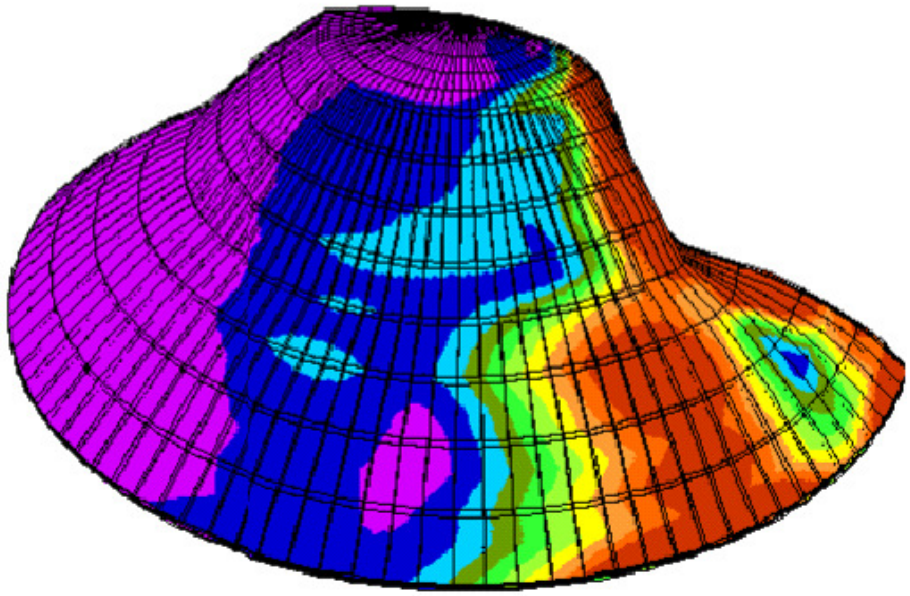
High memory consumption : very large 3D problems are out of their league (100 millions unknowns)

Great precision ~ 10^-18

We want a trade-off !

Numerical experiments (TERA1)

- Successful approach for a large collection of industrial test cases (PARASOL, Boeing Harwell, CEA) on IBM SP3
- TERA1 supercomputer of CEA Ile-de-France (ES45 SMP 4 procs)
- COUPOLE40000 :
  26.5 10^6 of unknowns
  1.5 10^{10} NNZL and 10.8Tflops
    - 356 procs: 34s
    - 512 procs: 27s
    - 768 procs: 20s



- (>500Gflop/s about 35% peak perf.)

Numerical experiments (TERA10)

- Successful approach on 3D mesh problem with about 30 millions of unkowns on TERA10 supercomputer
- But memory is the bottleneck !!!

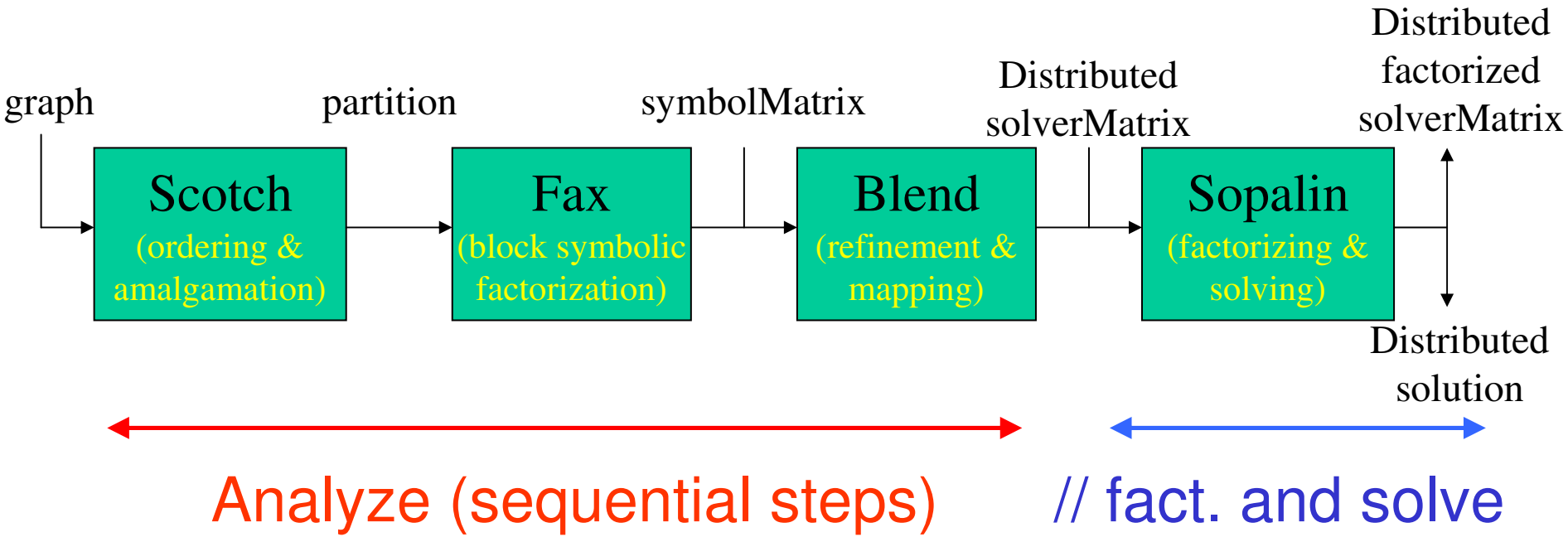
Motivation of this work

- 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

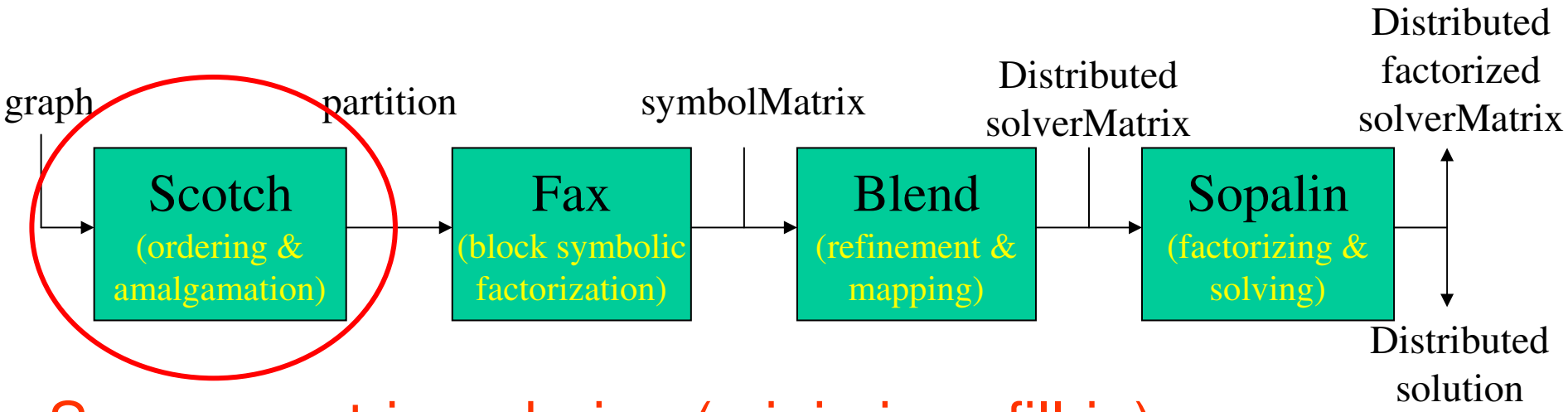
Outlines

- 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)



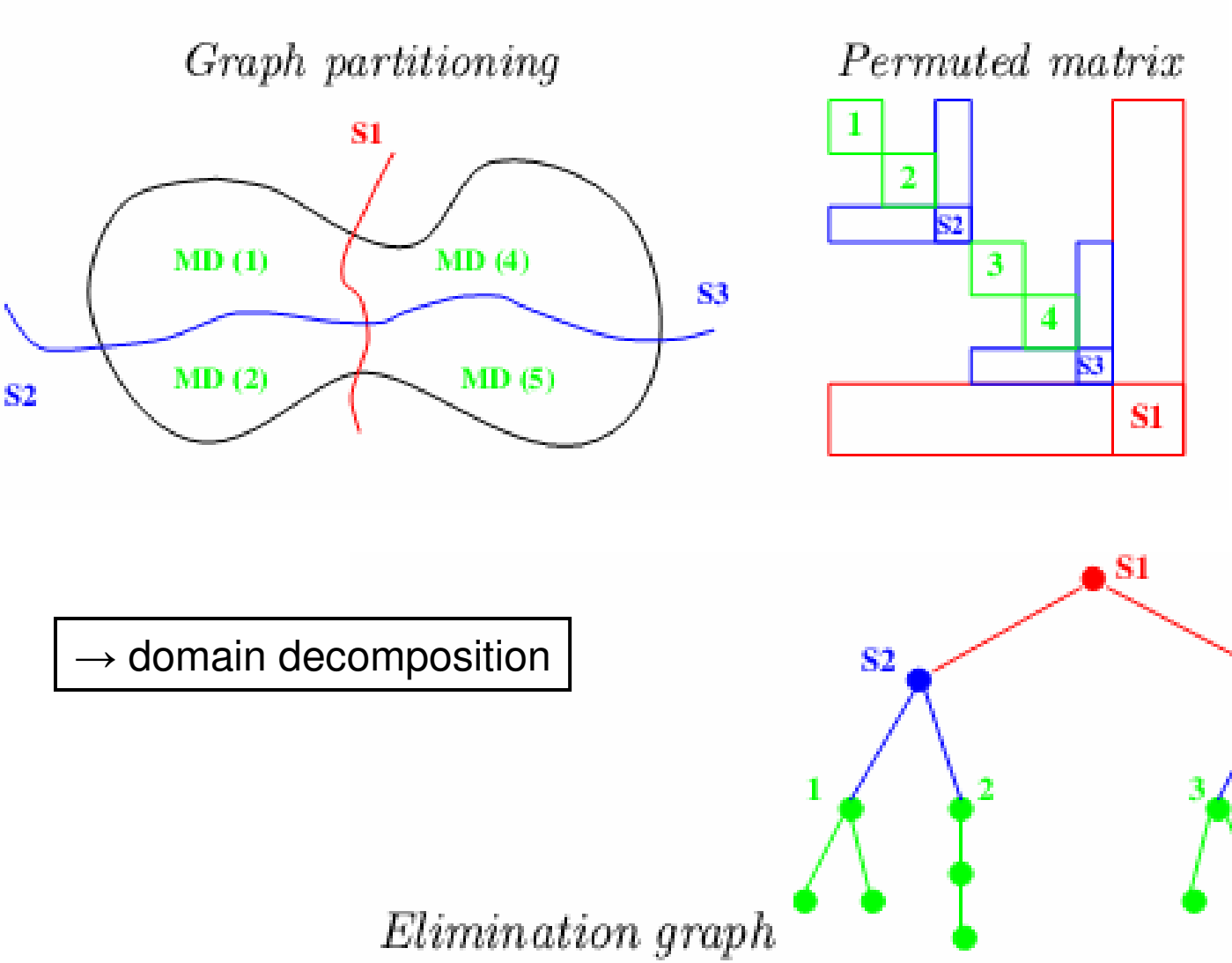
Direct solver chain (in PaStiX)



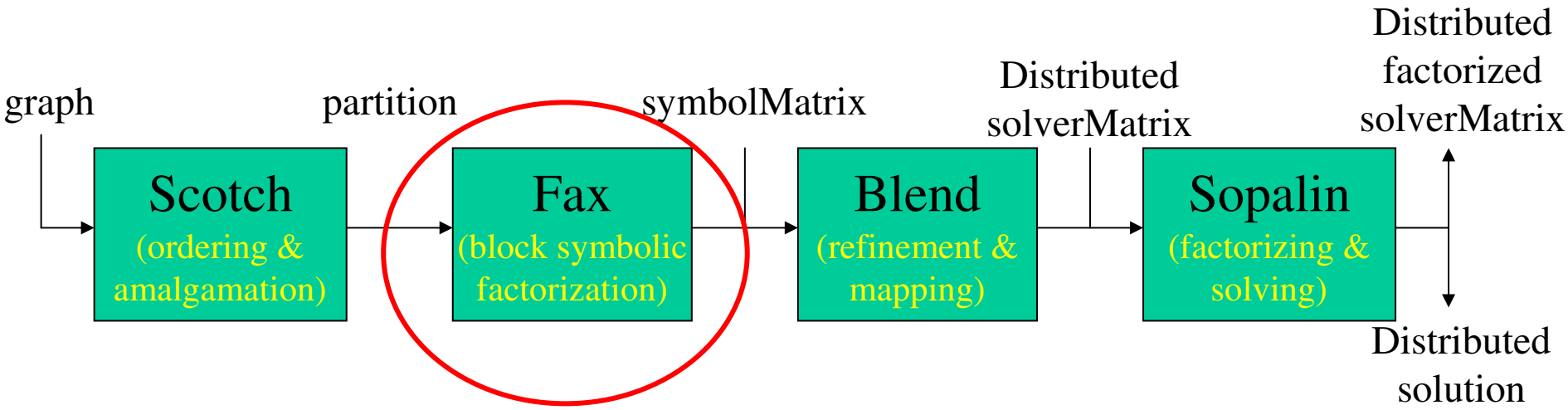
Sparse matrix ordering (minimizes fill-in)

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

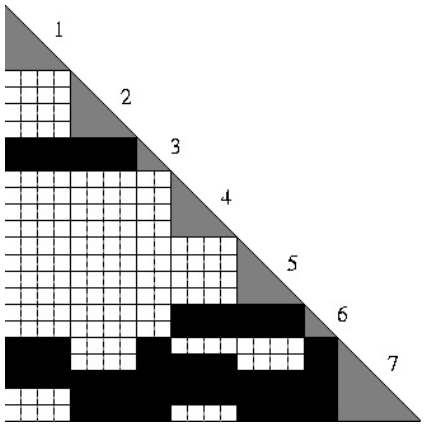
Partition and Block Elimination Tree



Direct solver chain (in PaStiX)



The symbolic block factorization



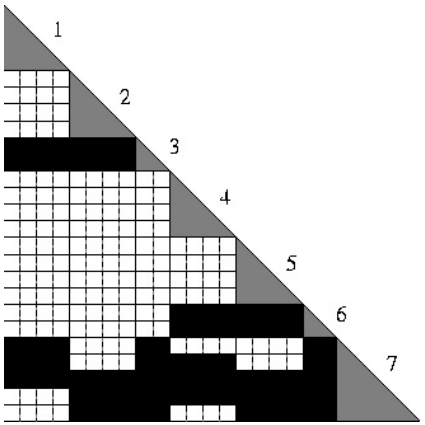
- ***Q(G,P)→Q(G,P)*=Q(G*,P) => linear in number of blocks!***
- Dense block structures ⇒ only a extra few pointers to store the matrix

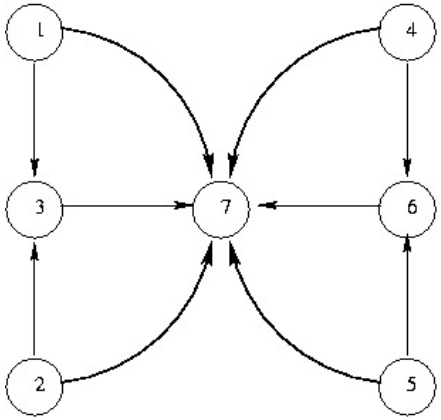
Matrix partitioning and mapping

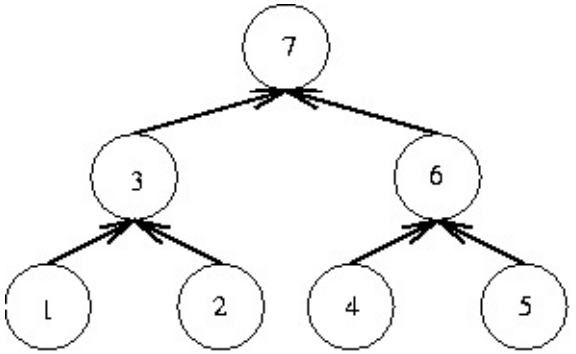
⇒ Manage parallelism induced by sparsity (block elimination tree).

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

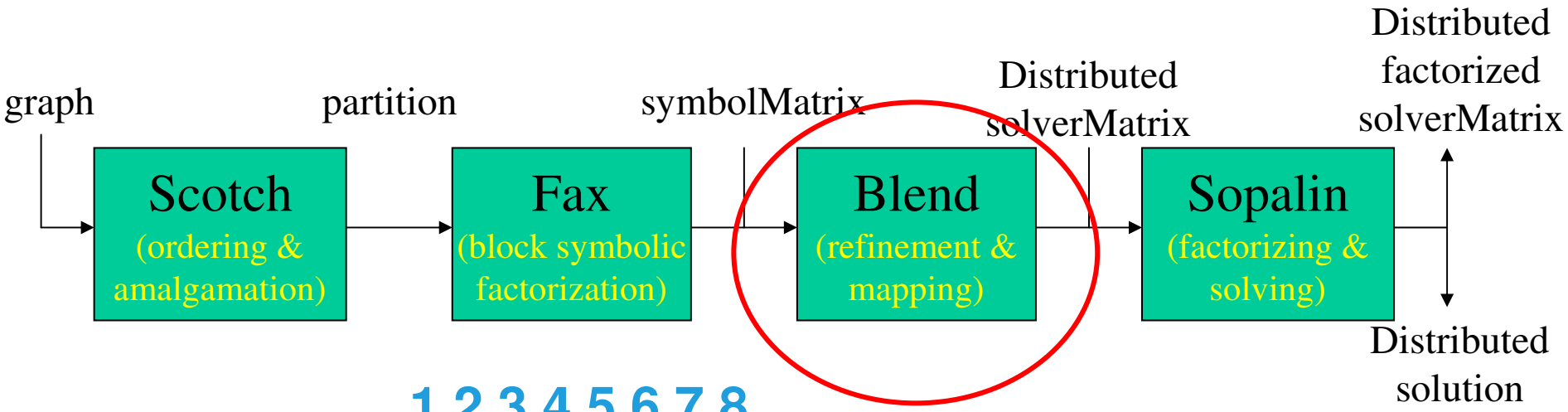
⇒ Use optimal block size for pipelined BLAS3 operations.

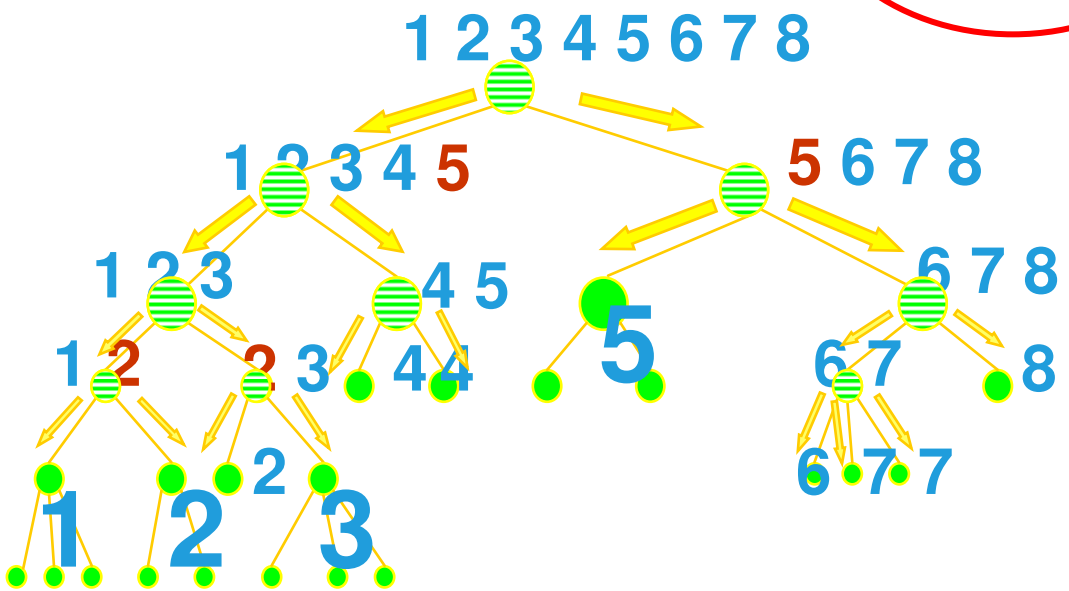




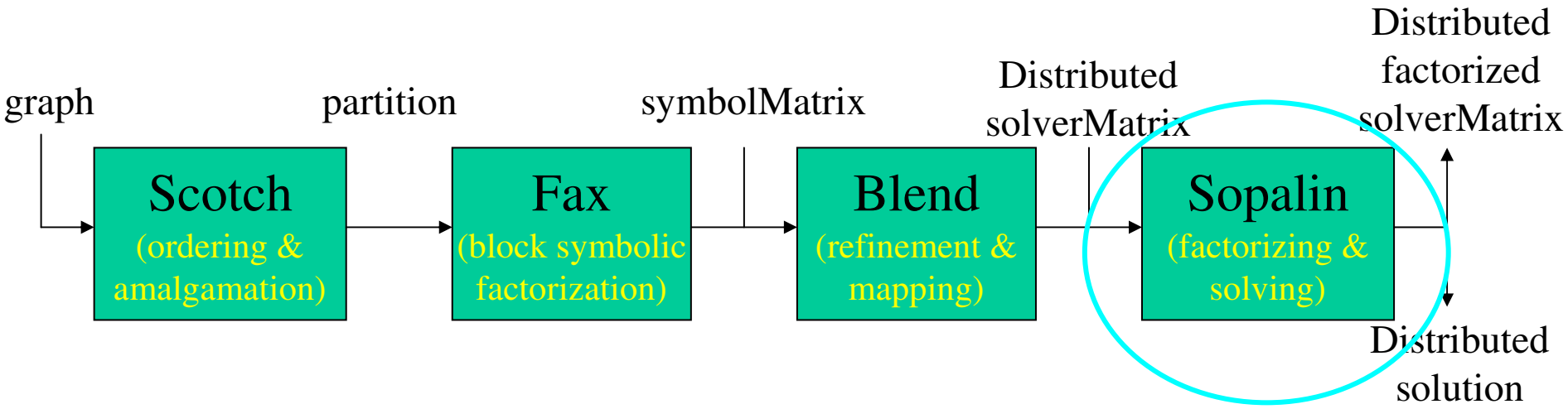


Direct solver chain (in PaStiX)

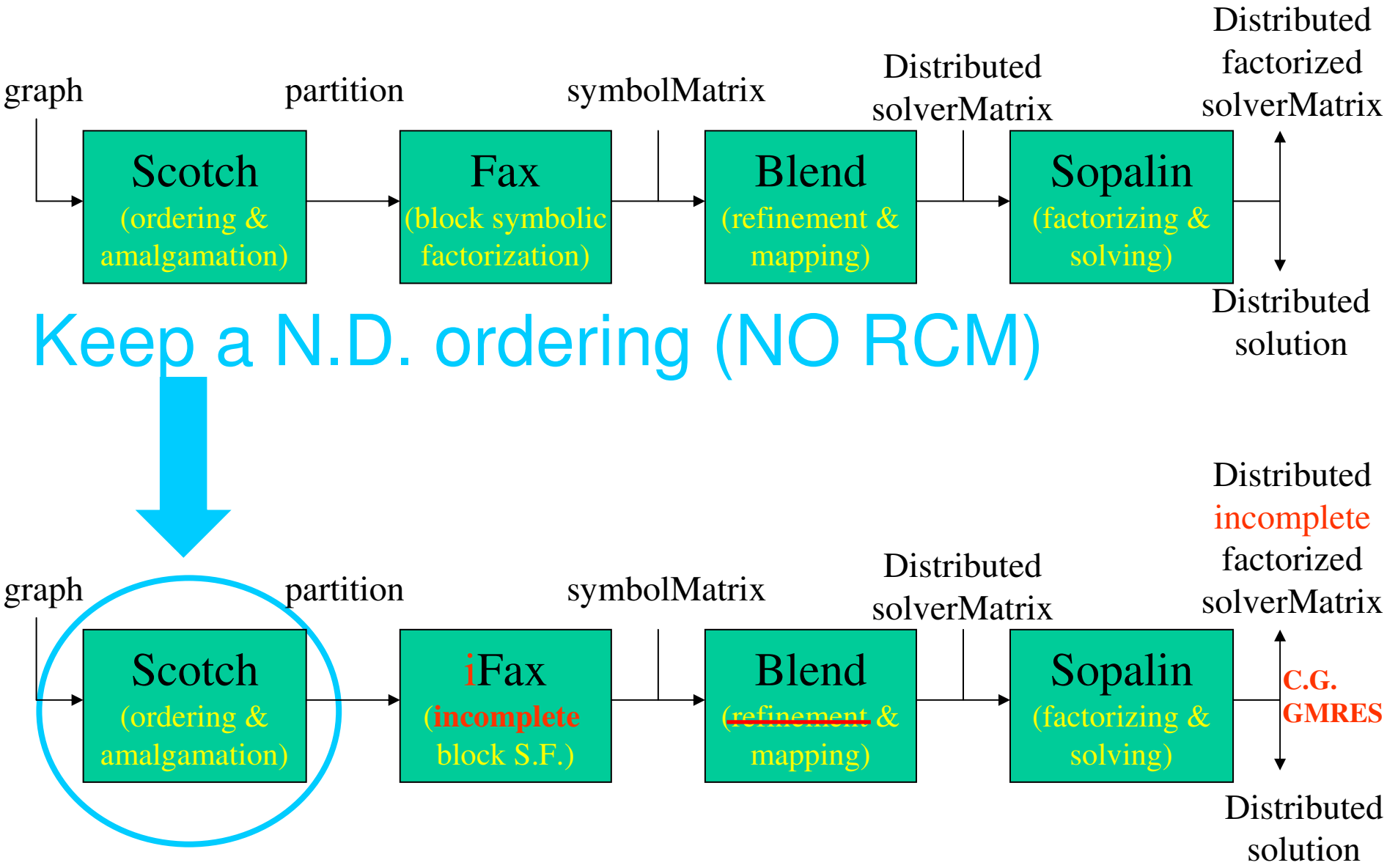


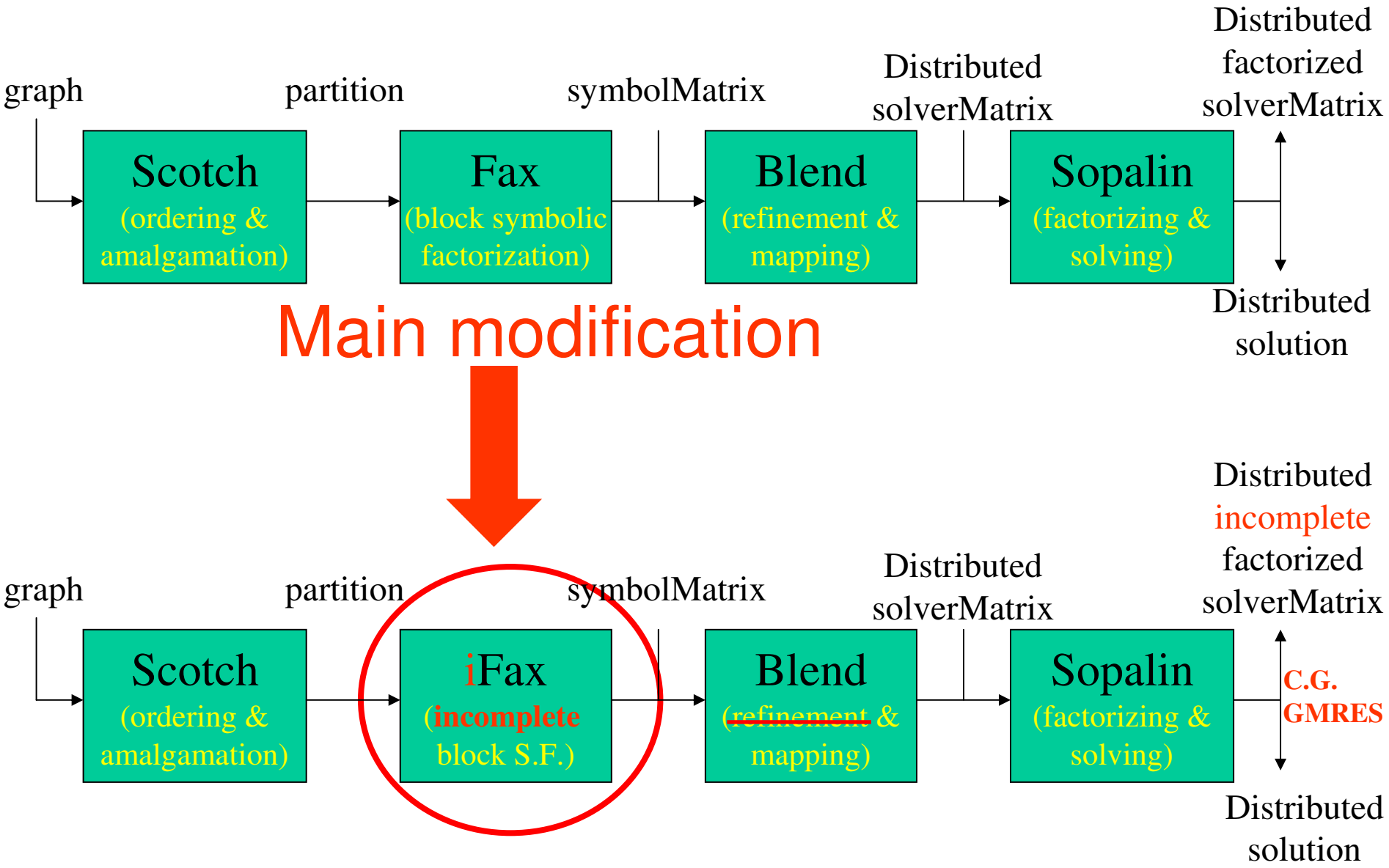


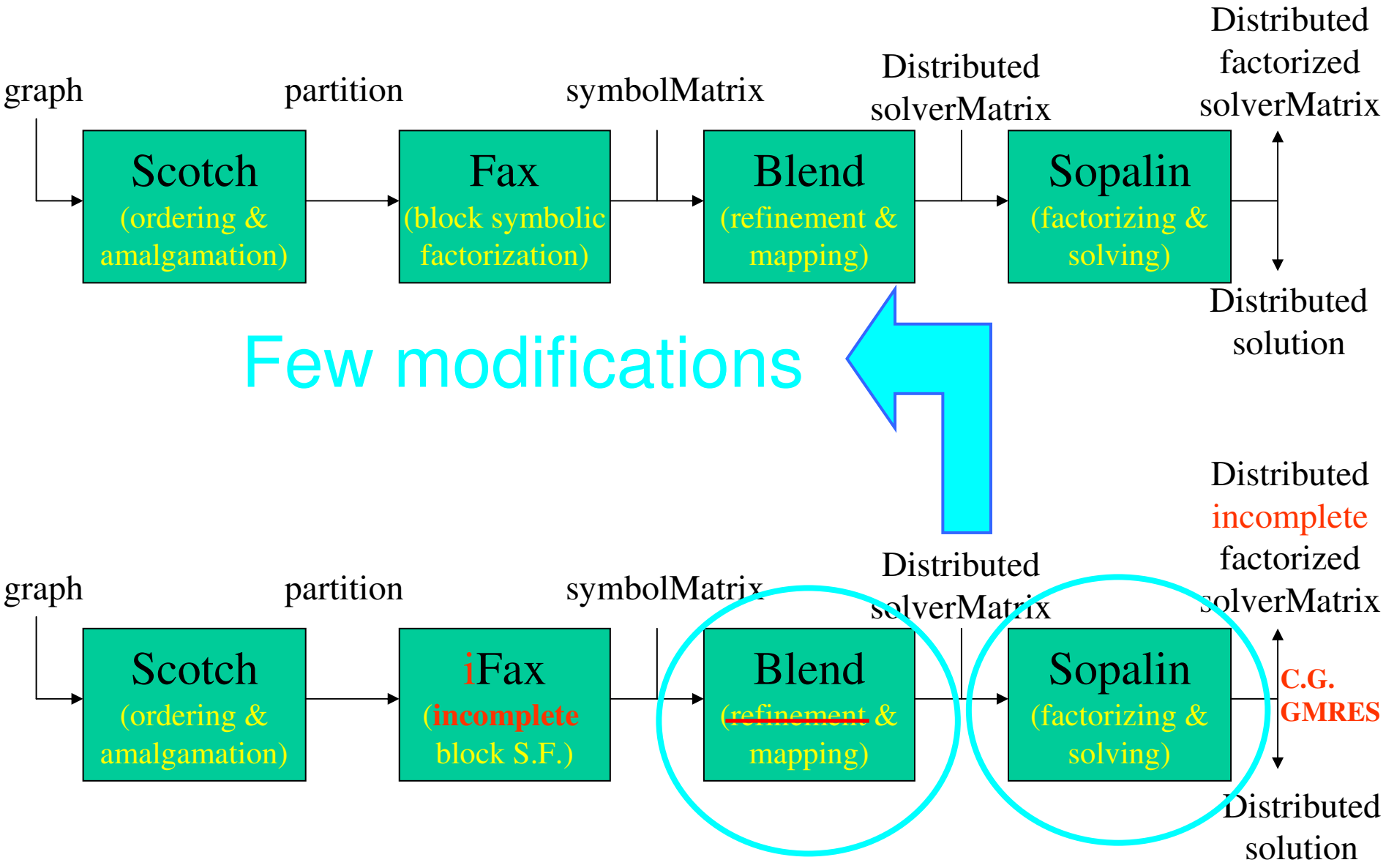
Direct solver chain (in PaStiX)



- 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

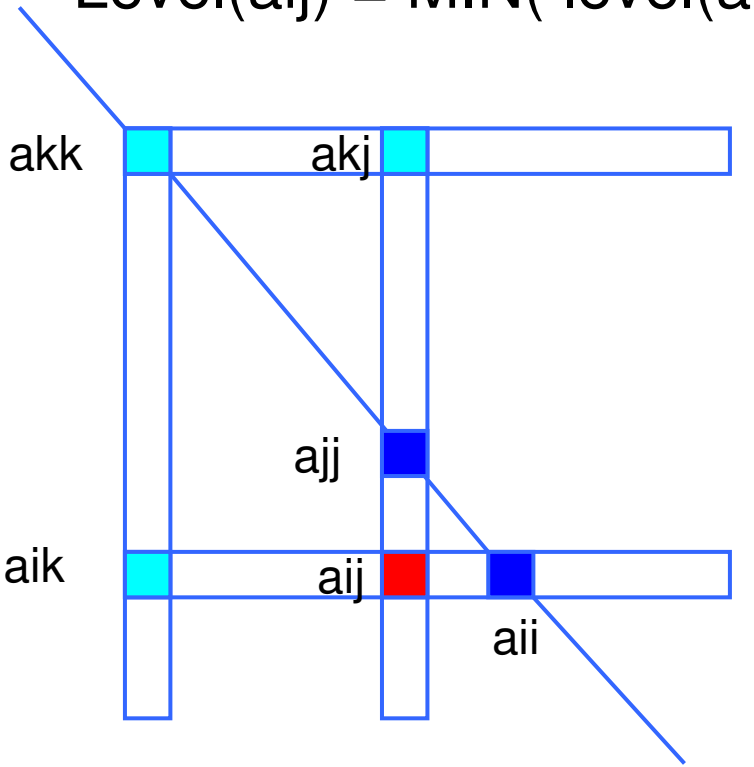
Outlines

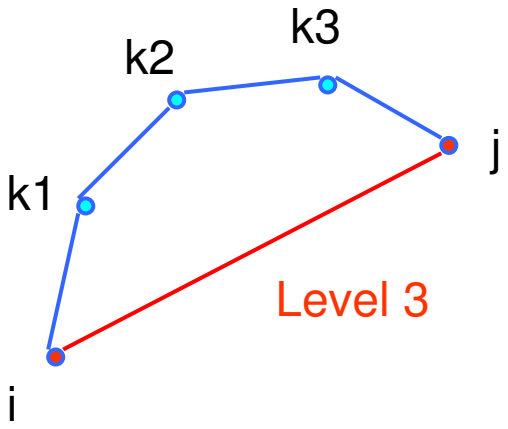
- Which modifications in the direct solver?
- ➢ The symbolic incomplete factorization
- An algorithm to get dense blocks in ILU
- Experiments
- Conclusion

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 k^{th} unknowns during the fact.
  then:

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





k1, k2, k3 < i and j

Level-of-fill metric

- Scalar formulation of the level-of-fill:
  - A fill-path, in the graph associated with the matrix A, is a path between two nodes i and j such that the nodes in this path have smaller number than i and j
  - There is a fill-in value (i,j), at the end of the Gauss elimination, iff there is a fill-path between i and j
  - At the end of the factorization, the level-of-fill value (i,j) is p iff there is a shortest fill-path of length p+1 between i and j (0 for terms in A)

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 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. :

  $$Q(G,P) \rightarrow Q(G,P)^k = Q(G^k,P) \quad P = \textit{partition of mesh nodes}$$

- 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 ?

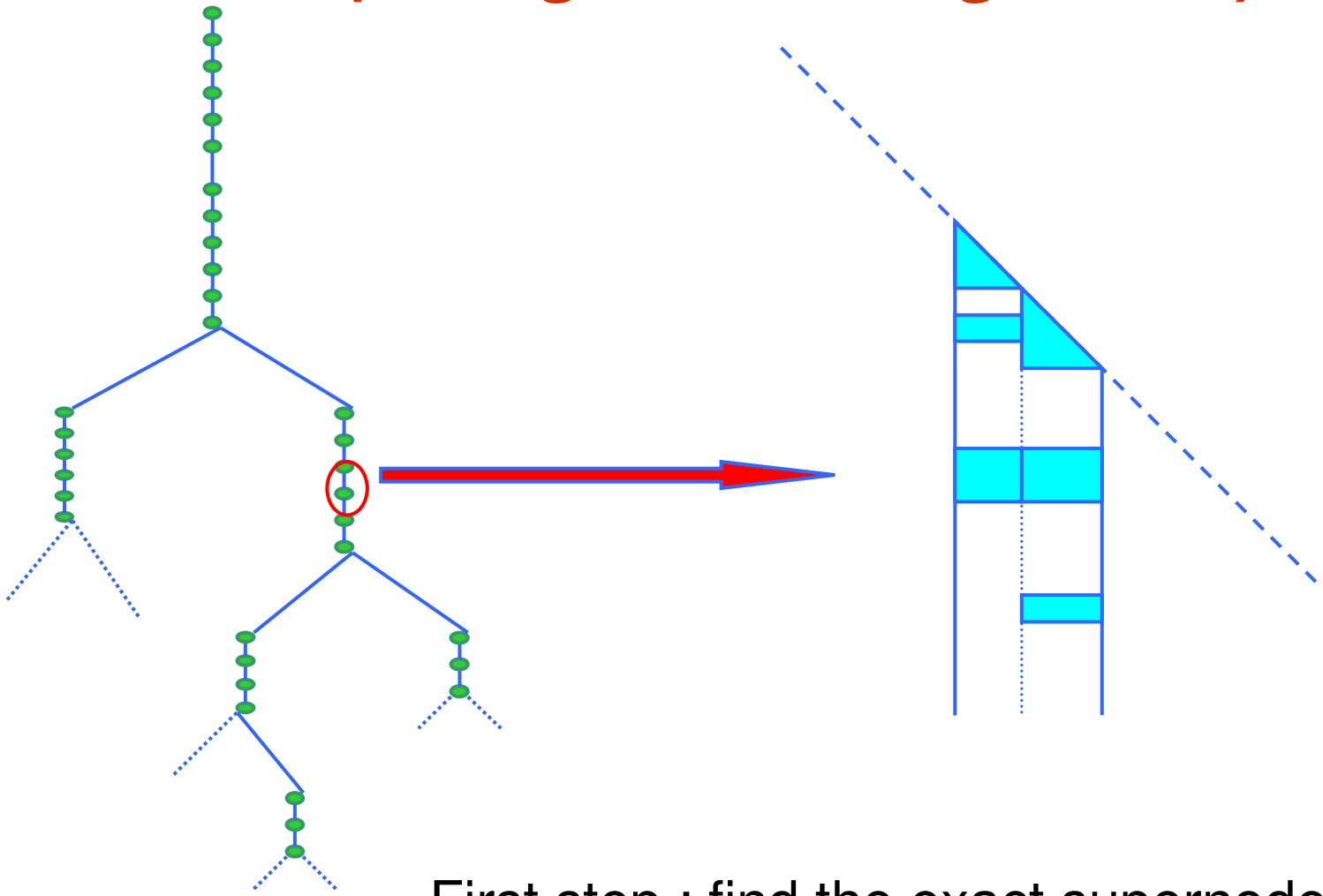
- We imposed some constraints:

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

➢ any permutation should not destroy the elimination tree structure

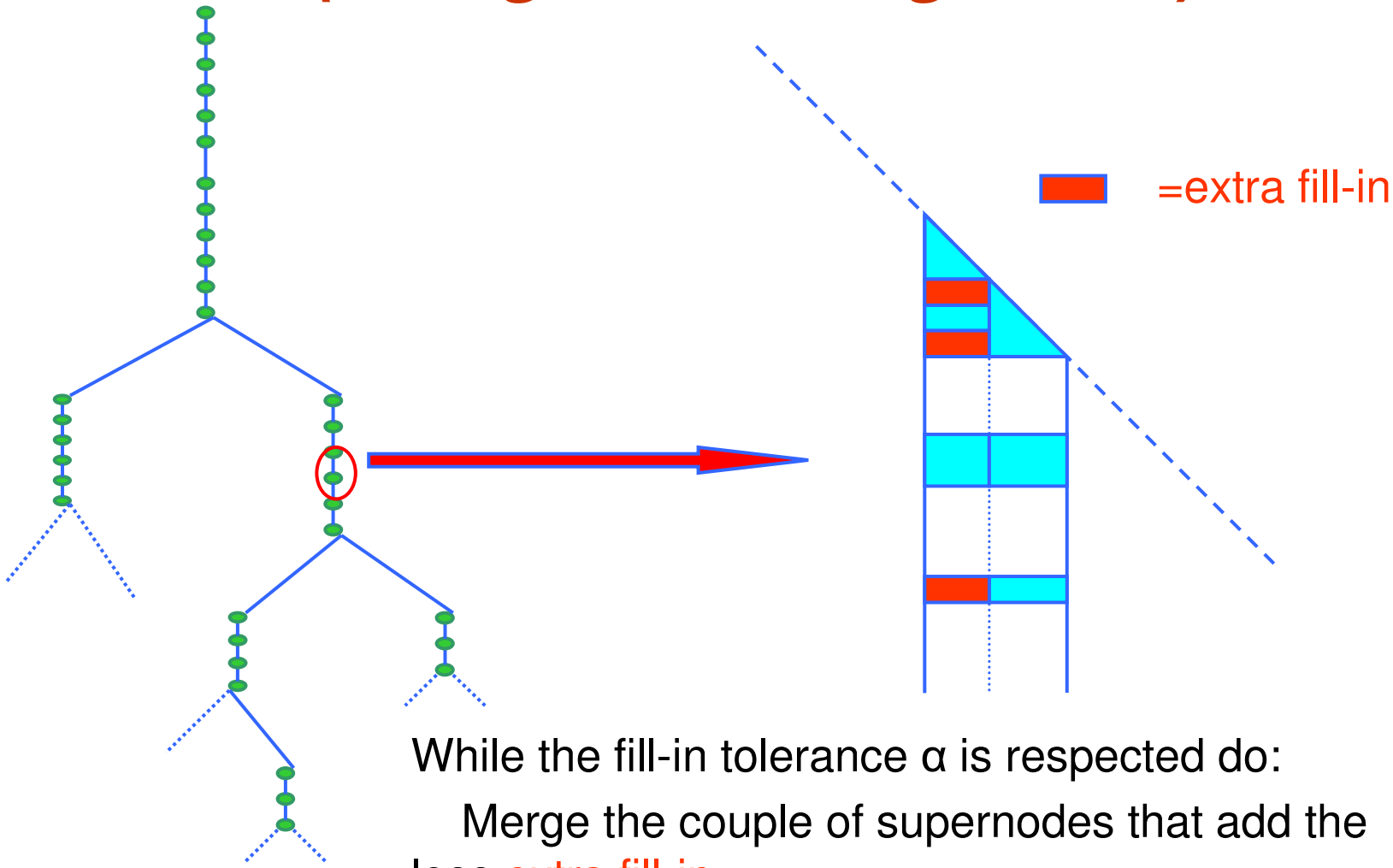
=> We impose the rule « merge only with your father… » for the supernode

Finding an approximated Supernodes Partition (amalgamation algorithm)



First step : find the exact supernode partition

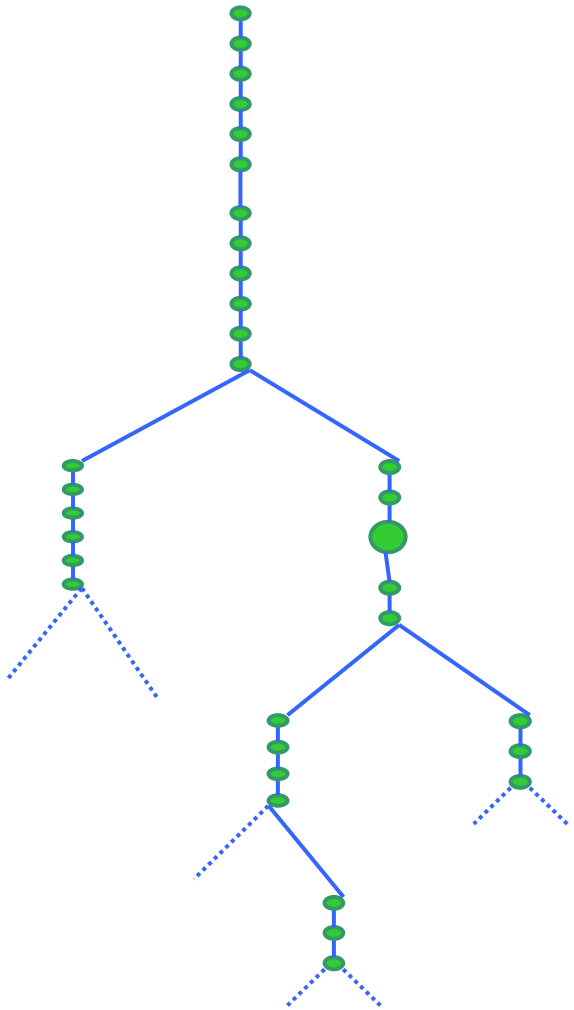
Finding an approximated Supernodes Partition (amalgamation algorithm)

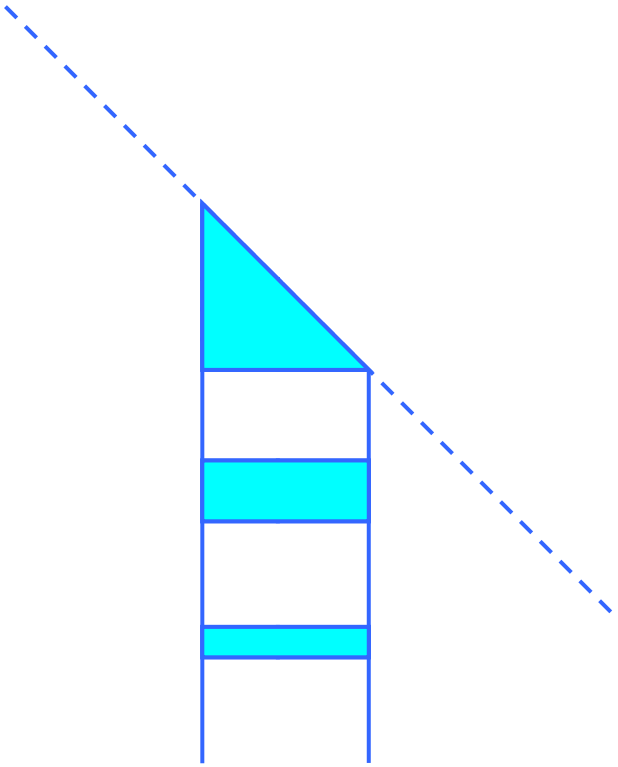


While the fill-in tolerance α is respected do:

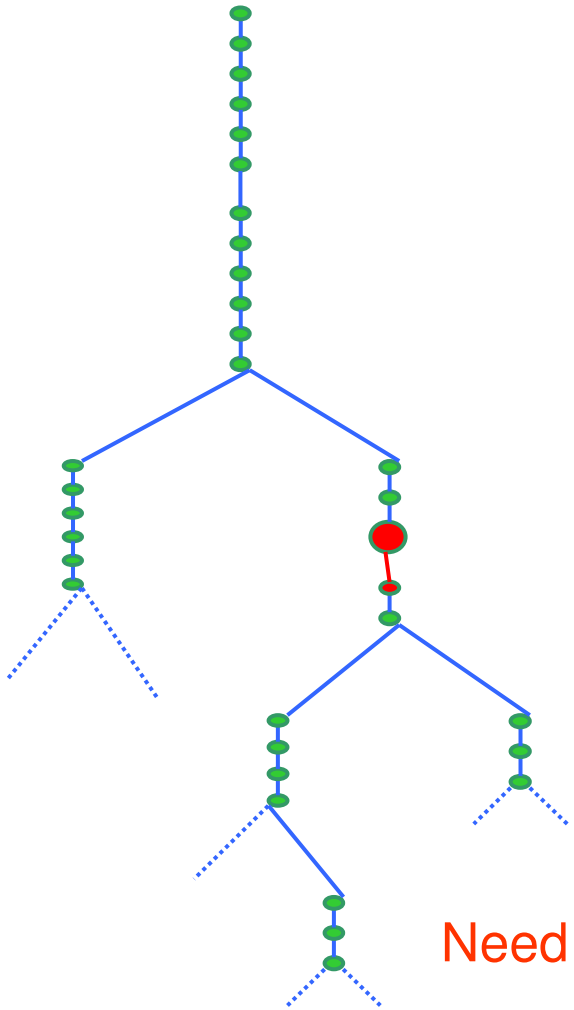
Merge the couple of supernodes that add the less extra fill-in

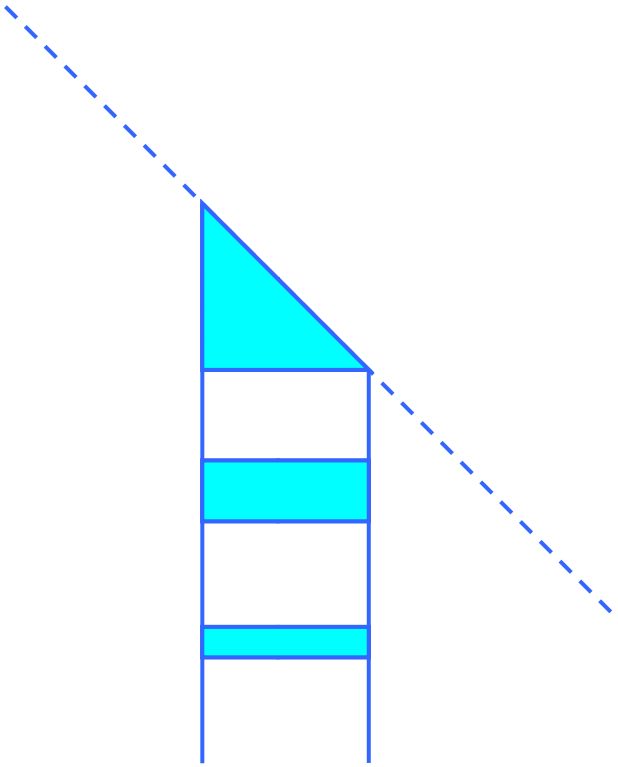
Finding an approximated Supernodes Partition (amalgamation algorithm)





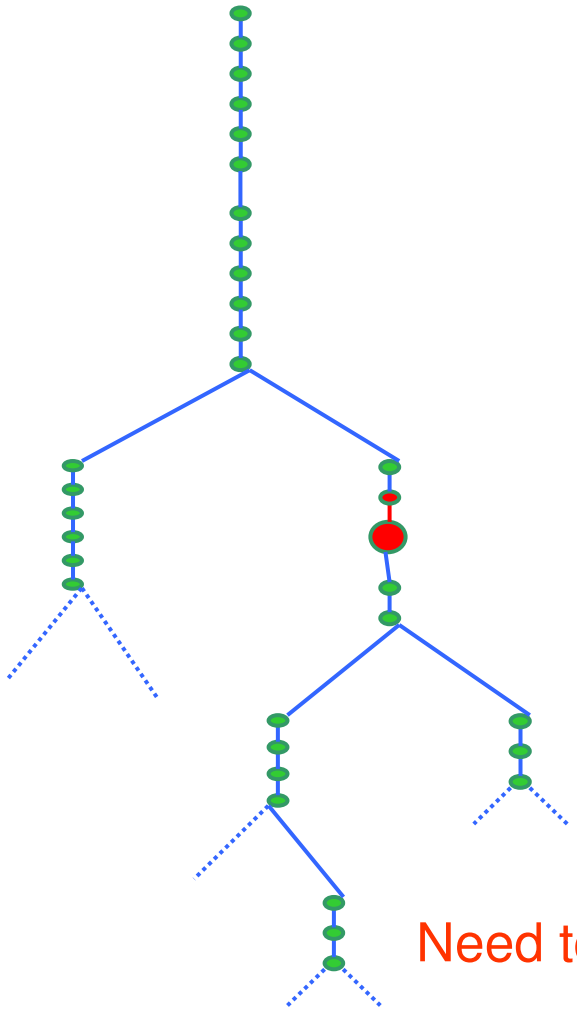
Finding an approximated Supernodes Partition (amalgamation algorithm)

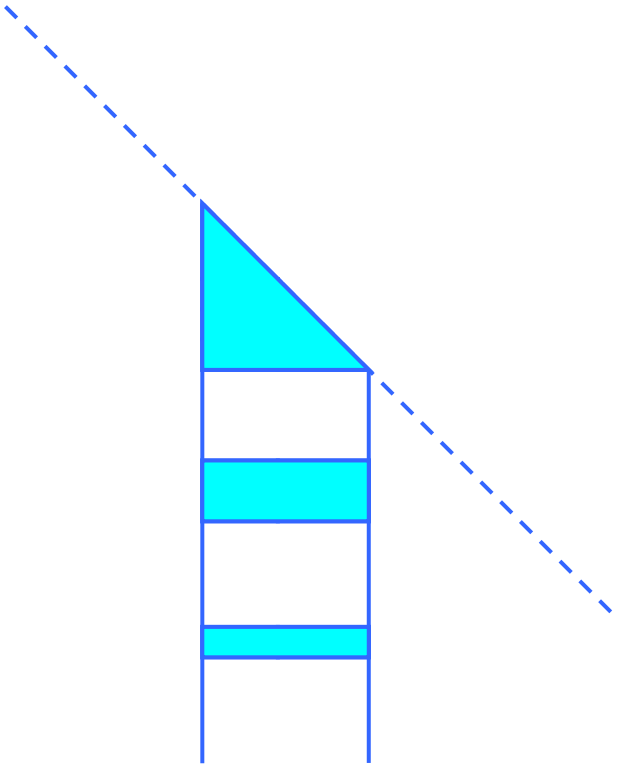




Need to update the « merge » cost of the son

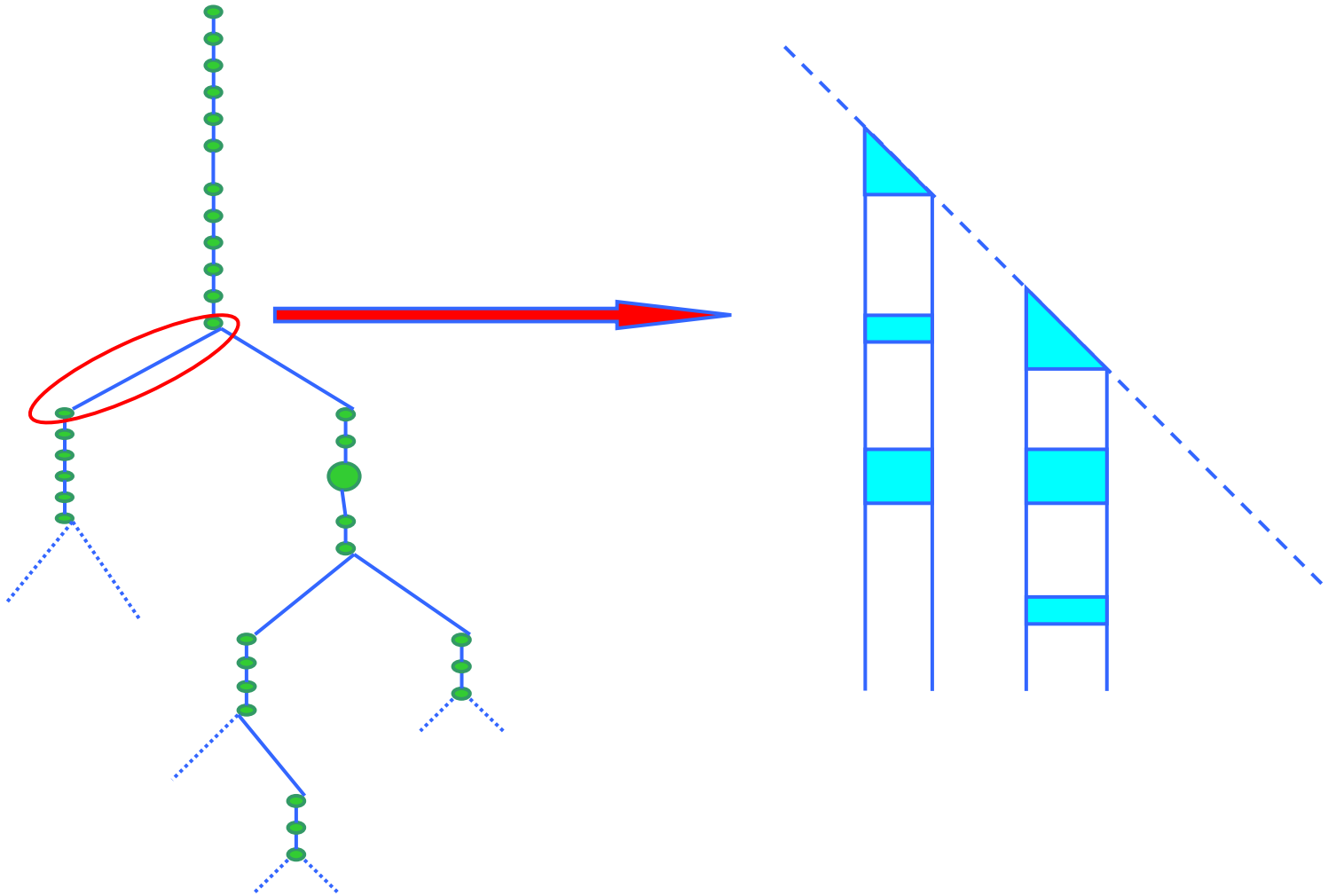
Finding an approximated Supernodes Partition (amalgamation algorithm)



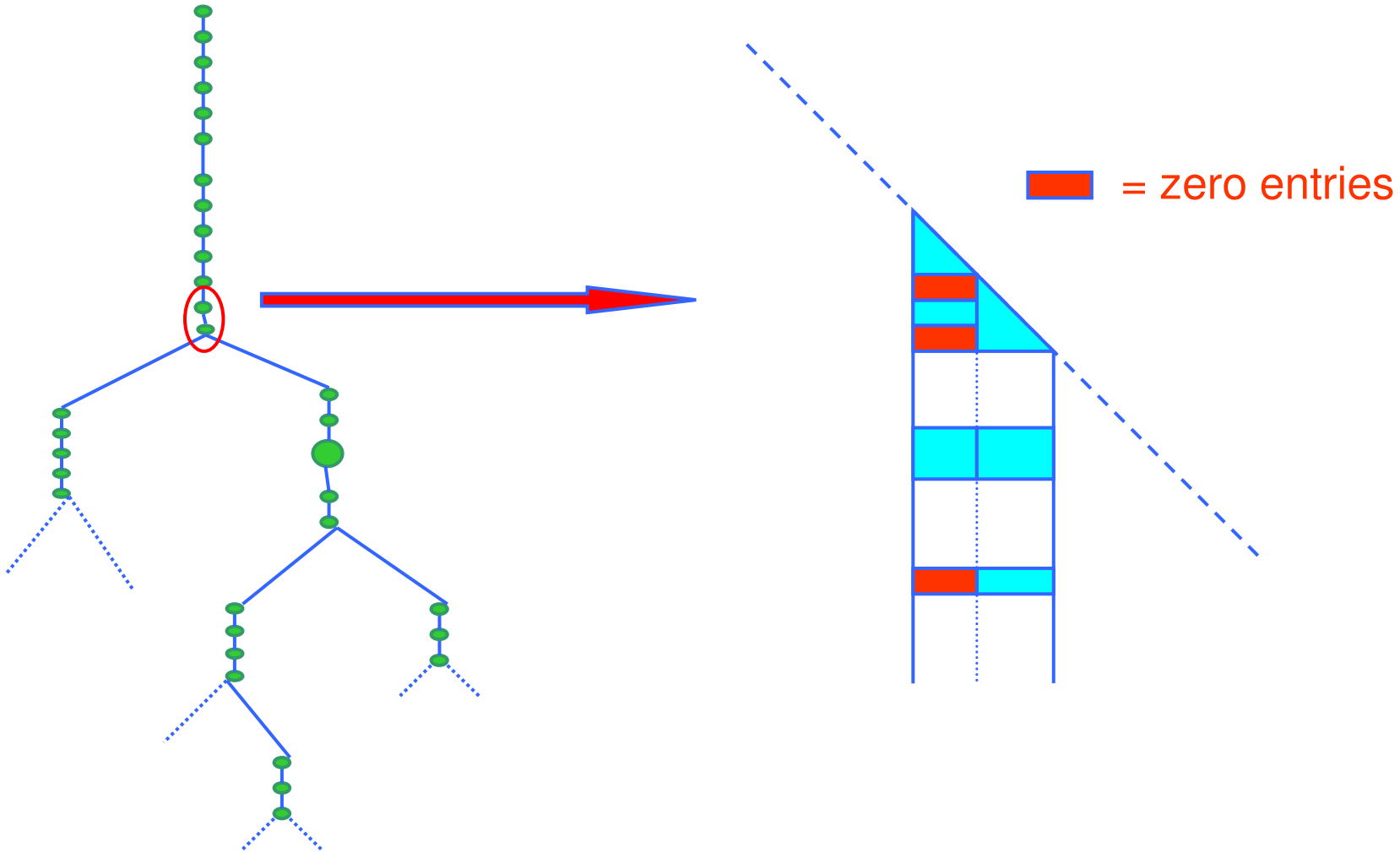


Need to update the « merge » cost of the father

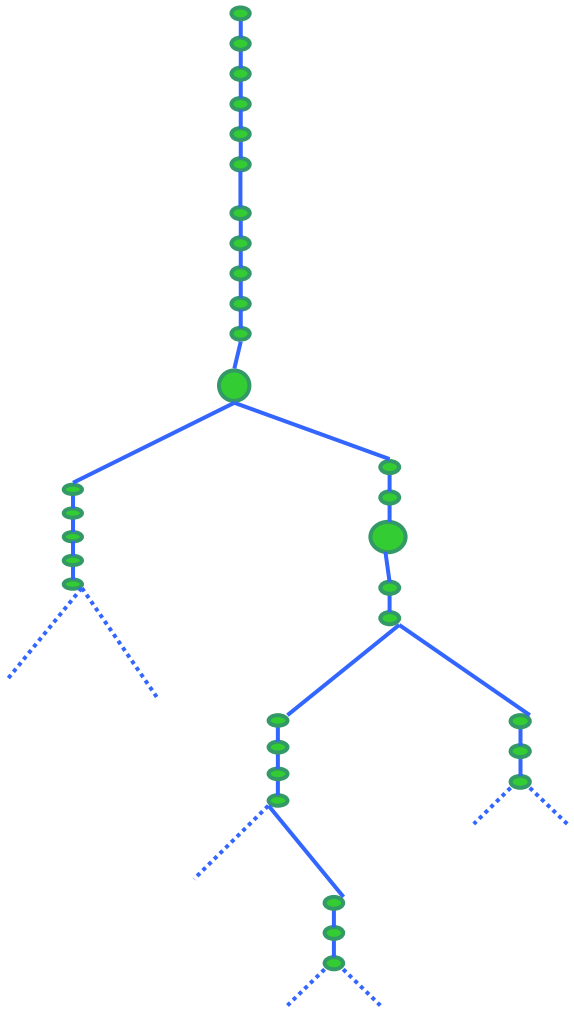
Finding an approximated Supernodes Partition (amalgamation algorithm)

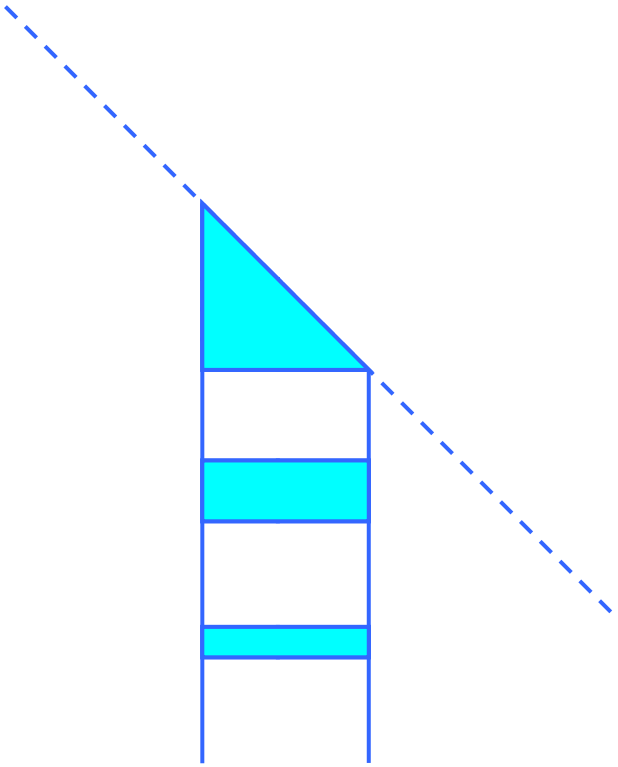


Finding an approximated Supernodes Partition (amalgamation algorithm)

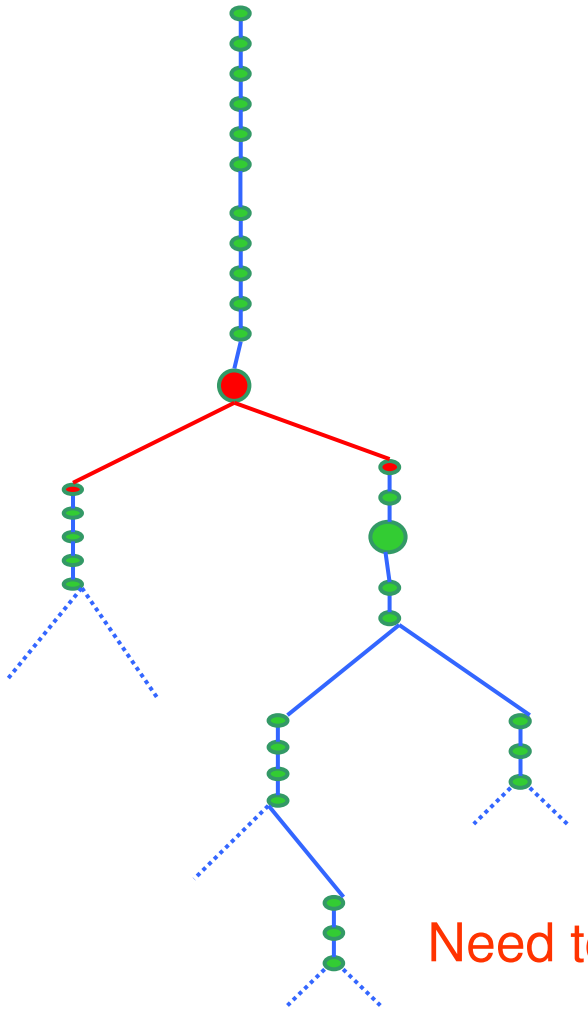


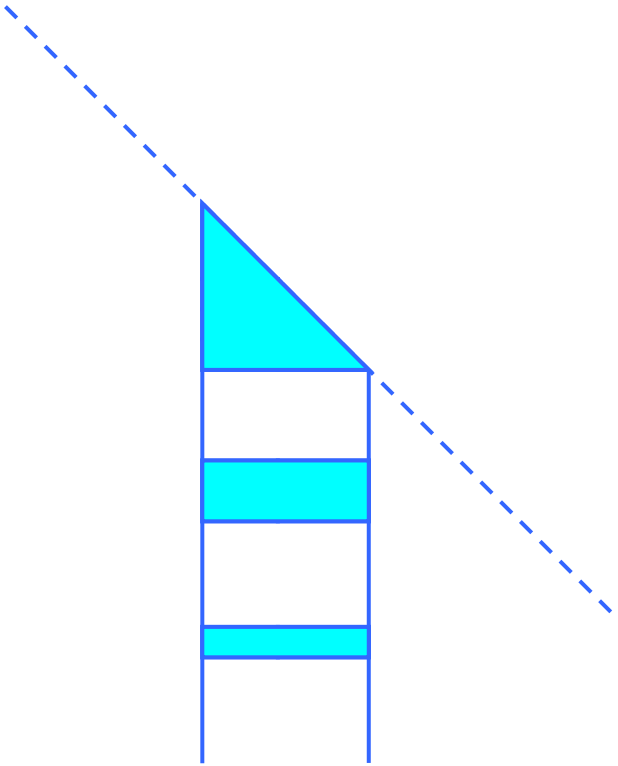
Finding an approximated Supernodes Partition (amalgamation algorithm)





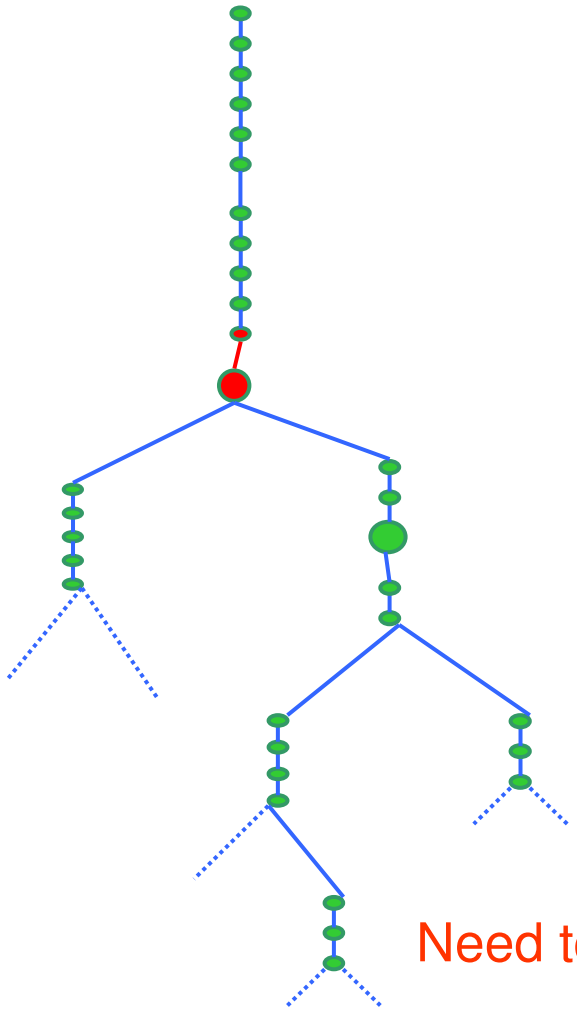
Finding an approximated Supernodes Partition (amalgamation algorithm)

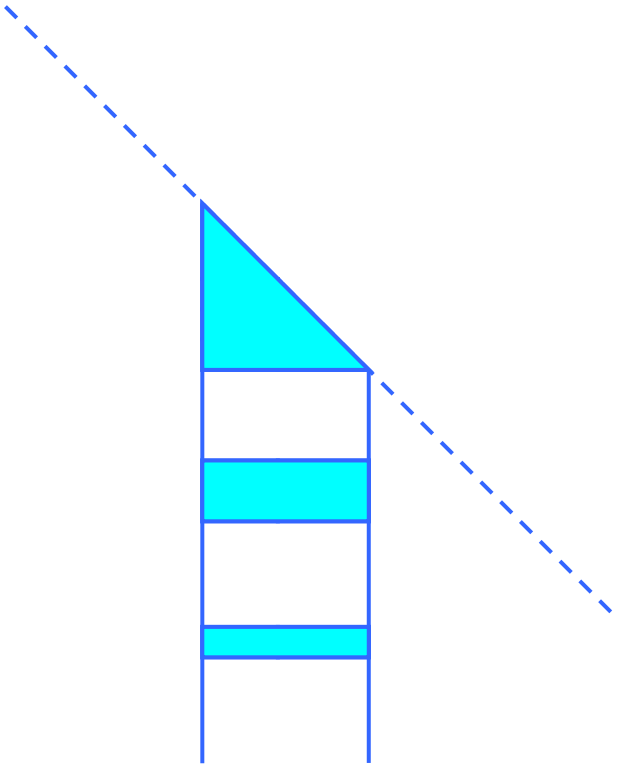




Need to update the « merge » cost of the sons

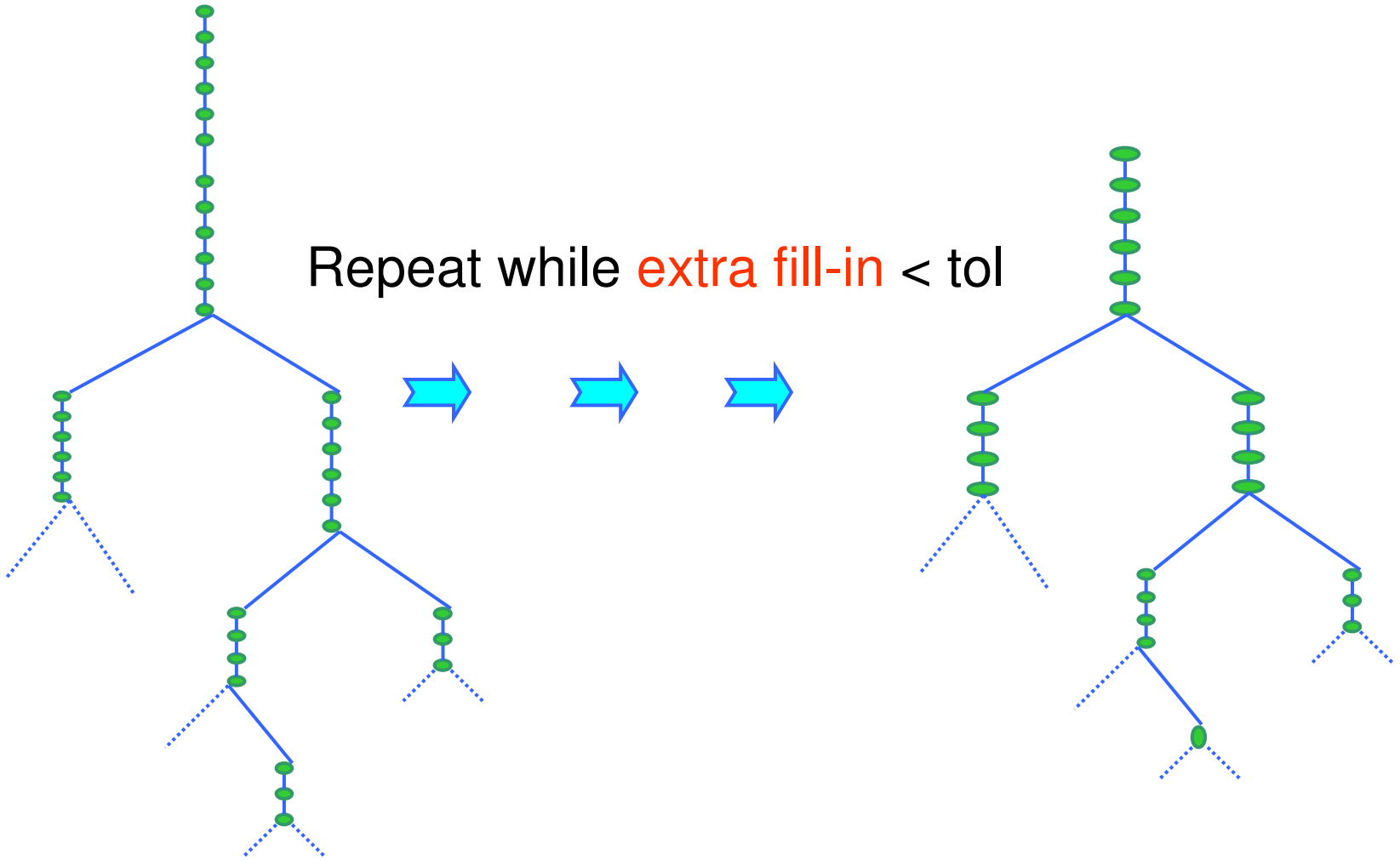
Finding an approximated Supernodes Partition (amalgamation algorithm)





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

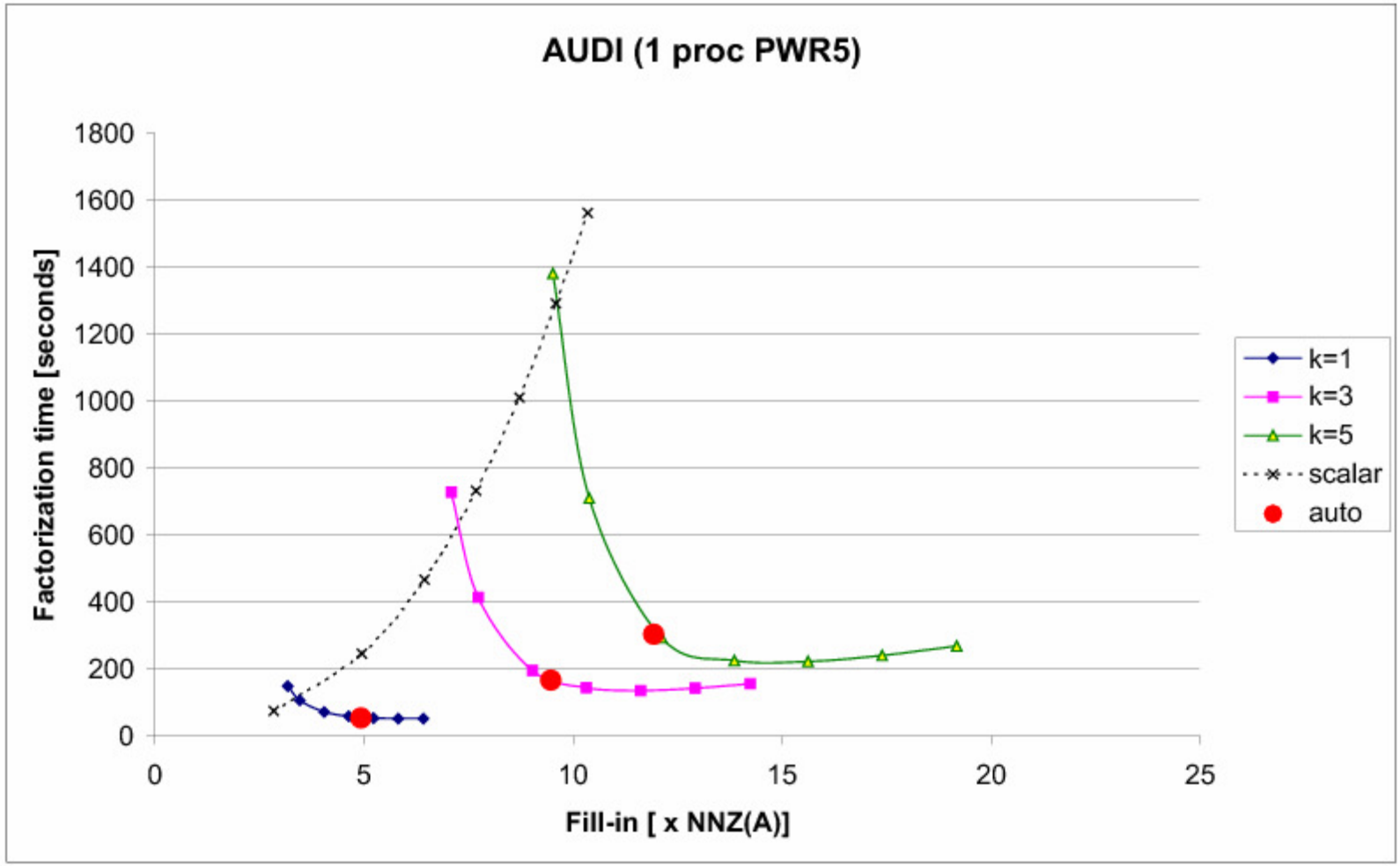
Numerical experiments

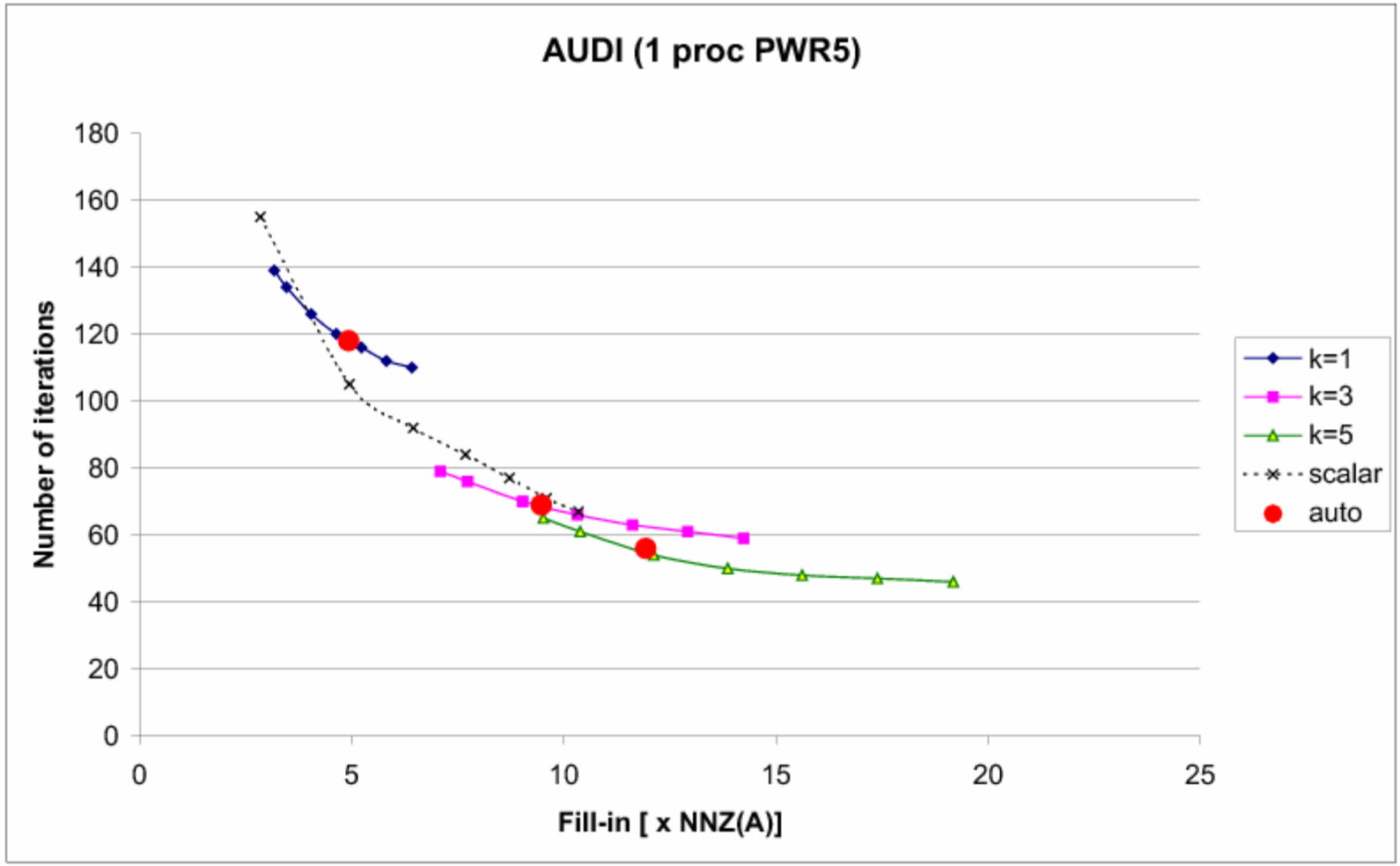
- Results on IBM power5 + Switch “Federation”
- All computations were performed in double precision
- Iterative accelerator was GMRES (no restart)
- Stopping criterion for iterative accelerators was a relative residual norm (||b-A.x||/||b||) of 1e-7

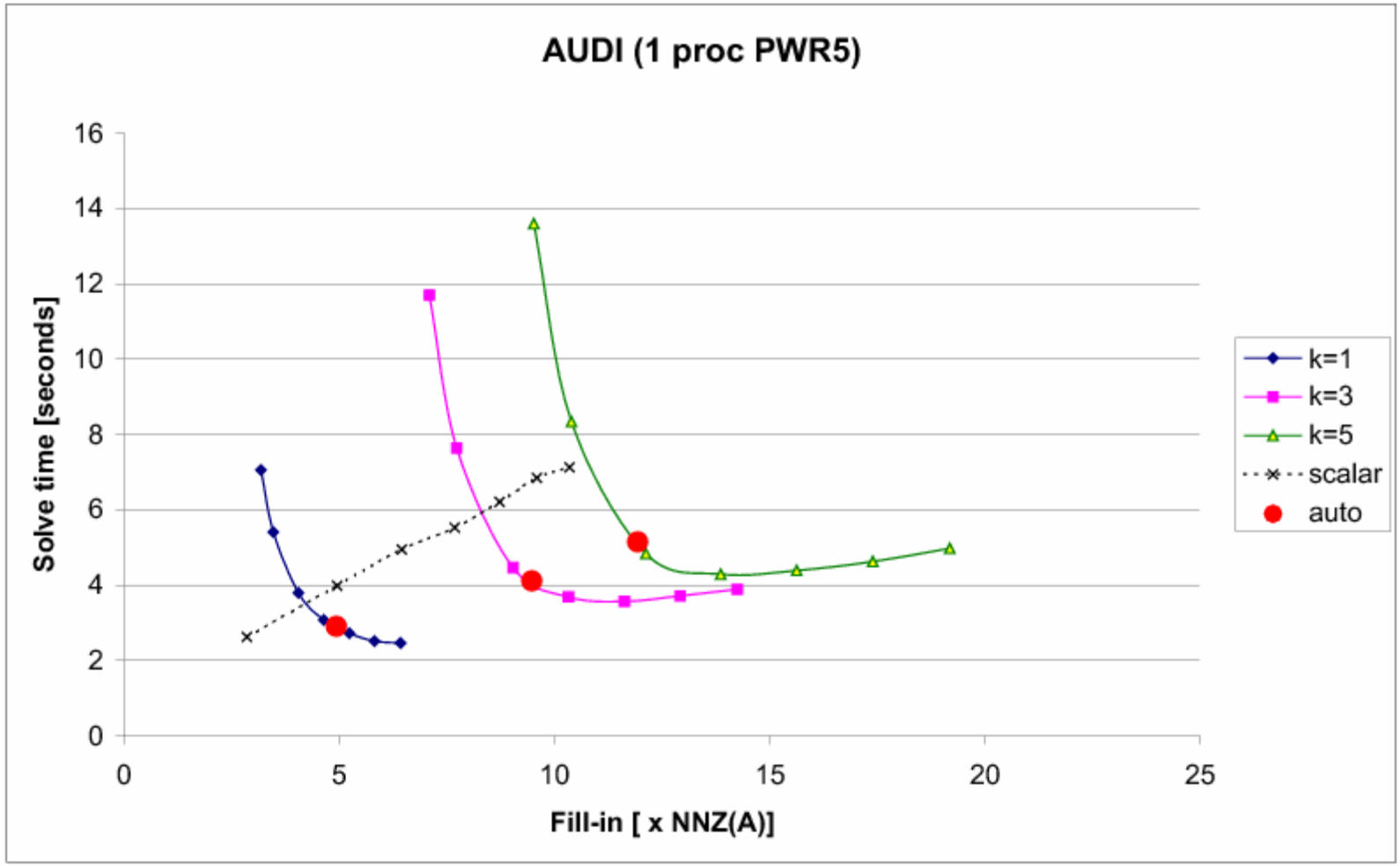
Test cases:

- AUDIKW_1 : Symmetric matrix (Parasol collection)
  n = 943,695 nnz(A) = 39,297,771
  With direct solver : nnz(L) = 30 x nnz(A)
  total solution in 91s on 16 procs
  ➔ 3D

- MHD : Unsymmetric matrix (Y. Saad collection)
  n = 485,597 nnz(A) = 24,233,141
  With direct solver : nnz(L) = 46 x nnz(A)
  total solution in 139s on 16 procs
  ➔ 3D


AUDI (1 proc PWR5)
Factorization time [seconds]
1800
1600
1400
1200
1000
800
600
400
200
0
0
5
10
15
20
25
Fill-in [x NNZ(A)]
k=1
k=3
k=5
scalar
auto


AUDI (1 proc PWR5)
Number of iterations
180
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
Fill-in [x NNZ(A)]
k=1
k=3
k=5
scalar
auto


AUDI (1 proc PWR5)
Solve time [seconds]
16
14
12
10
8
6
4
2
0
0
5
10
15
20
25
Fill-in [x NNZ(A)]
k=1
k=3
k=5
scalar
auto

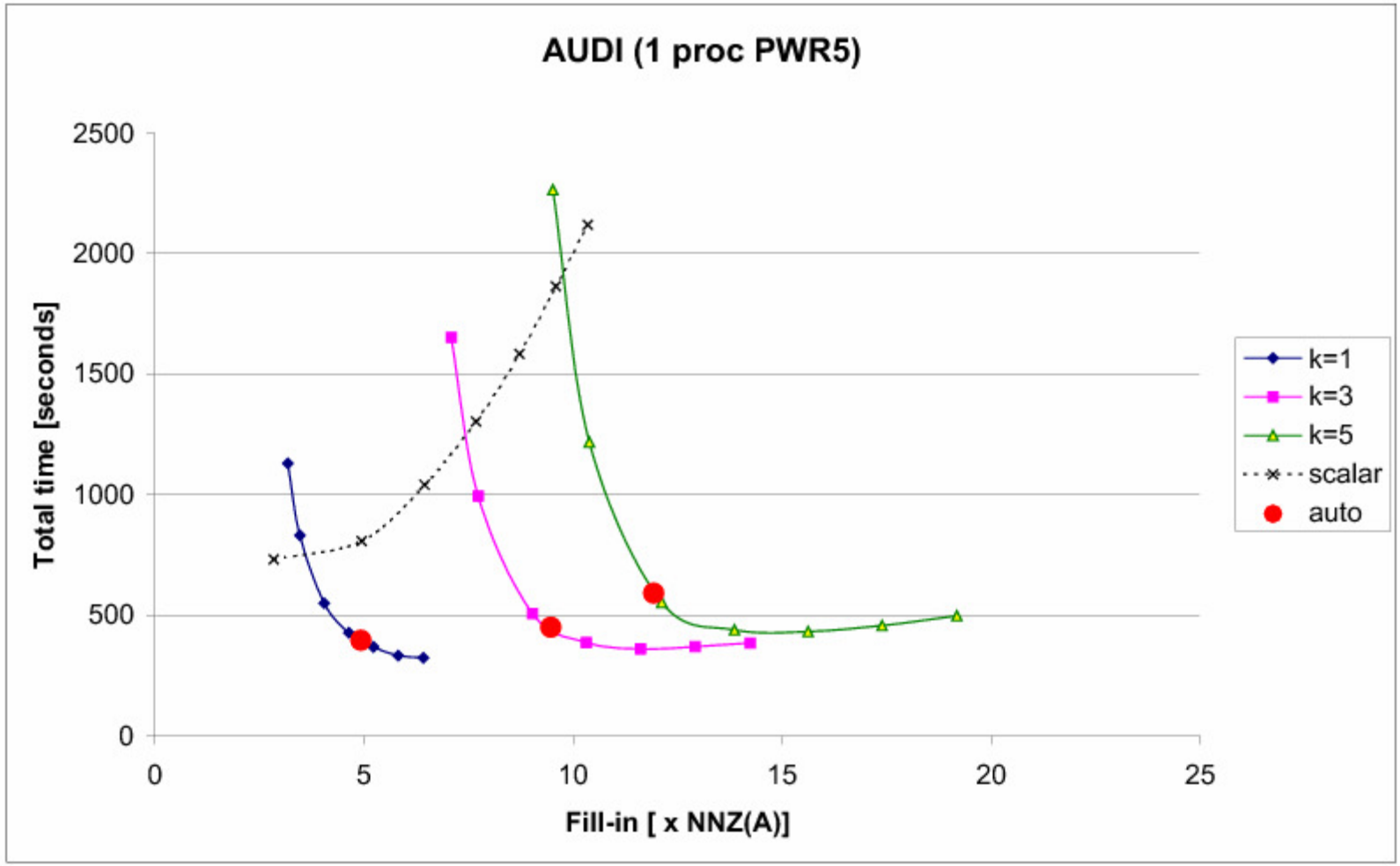

AUDI (1 proc PWR5)
Total time [seconds]
2500
2000
1500
1000
500
0
0
5
10
15
20
25
Fill-in [x NNZ(A)]
k=1
k=3
k=5
scalar
auto

Table 3. Effect of amalgamation ratio α for MHD problem

k	α	# Supernodes	# Blocks	Fill-in	Amalg.	Inc. Fact.	Triang. Solve	Iterations	Total
1	∞	40215	528640	4.04	4.36	12.3	1.05	153	172
1	0	132615	1585901	1.77	1.51	16.0	2.04	172	366
1	10	103119	1199872	1.96	1.83	14.3	1.67	164	288
1	20	91975	1086335	2.16	1.92	13.7	1.55	164	267
1	40	71767	903500	2.57	2.15	12.7	1.36	162	233
1	60	56402	753112	2.98	2.33	12.1	1.22	158	204
1	80	43912	651356	3.41	2.48	12.2	1.12	156	186
1	100	33590	561447	3.81	2.62	12.4	1.05	153	173
1	120	26375	479080	4.19	2.76	12.8	0.97	149	**157**
3	∞	50361	803279	6.34	6.32	37.4	1.44	88	164
3	0	132485	3202800	3.69	2.20	85.6	3.52	100	437
3	10	93524	2296760	4.10	2.67	66.7	2.66	98	327
3	20	76199	1862555	4.51	2.93	57.8	2.30	97	280
3	40	53717	1390499	5.34	3.26	50.1	1.91	94	229
3	60	38617	1091834	6.15	3.53	46.5	1.67	92	200
3	80	27862	853927	6.96	3.75	44.6	1.49	89	177
3	100	20806	658369	7.74	3.92	43.8	1.35	86	159
3	120	16239	521220	8.57	4.05	44.2	1.27	83	**149**
5	∞	47646	932096	8.66	7.84	73.6	1.71	67	188
5	0	131806	4633544	5.42	2.76	217	4.81	79	596
5	10	83467	3215398	6.03	3.42	164	3.56	78	441
5	20	64718	2692706	6.65	3.69	145	3.12	77	385
5	40	41257	1811205	7.82	4.13	115	2.38	73	288
5	60	27553	1181086	8.97	4.45	94.3	1.91	69	226
5	80	19373	833068	10.15	4.68	85.5	1.65	66	194
5	100	14174	608861	11.35	4.84	80.4	1.53	64	178
5	120	10875	455535	12.51	4.96	79.6	1.48	61	**169**

Table 4. Performances on 1, 4, 8 and 16 processors PWR5 for 3 test ca

AUDI						
	1 processor			4 processors		
k	Inc. Fact.	Triang. Solve	Total	Inc. Fact.	Triang. Solve	Total
1	53.7	2.91	397	14.2	0.84	113
3	167	4.12	451	43.1	1.21	126
5	304	5.15	592	78.2	1.55	165
	8 processor			16 processors		
k	Inc. Fact.	Triang. Solve	Total	Inc. Fact.	Triang. Solve	Total
1	7.56	0.51	68.4	6.34	0.39	52.2
3	22.4	0.74	73.8	12.3	0.52	48.4
5	40.8	0.91	91.7	22.1	0.76	64.9

MHD						
	1 processor			4 processors		
k	Inc. Fact.	Triang. Solve	Total	Inc. Fact.	Triang. Solve	Total
1	12.3	1.05	172	3.25	0.29	48.2
3	37.4	1.44	164	9.83	0.41	46.5
5	73.6	1.71	188	19.6	0.50	53.3
	8 processor			16 processors		
k	Inc. Fact.	Triang. Solve	Total	Inc. Fact.	Triang. Solve	Total
1	1.94	0.18	29.6	2.08	0.17	27.9
3	5.25	0.27	28.9	4.17	0.25	26.1
5	10.2	0.32	31.8	6.56	0.29	26.2

Table 5. Direct factorization on 16 processors

Name	Columns	NNZ_A	Fill-in	Num. Fact.	Triang. Solve
AUDI	943695	39297771	30.1	91.4	1.21
MHD	485597	24233141	45.7	139	0.56

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.

Prospects:

- Parallelization of the ordering (ParMetis, PT-Scotch) and of the Inc. Symbolic Factorization
- Perform more experiments to explore different classes of problems with symmetric and unsymmetric version
- Plug this solver in real simulations (CEA, ITER)

- Matrices from G. Huysmans problems seems to be good candidates for hybrid solvers (large enough to reach memory limits on parallel clusters, k~3 for medium size systems)
- Manage mesh refinement techniques into our solvers (ASTER project)

Links

- Scotch : http://gforge.inria.fr/projects/scotch
- PaStiX : http://gforge.inria.fr/projects/pastix
- MUMPS : http://mumps.enseeiht.fr/ http://graal.ens-lyon.fr/MUMPS
- ScAlApplix : http://www.labri.fr/project/scalapplix

- ANR CIGC Numasis
- ANR CIS Solstice & Aster