Fast and Scalable Hierarchical Linear Solvers

Léopold Cambier

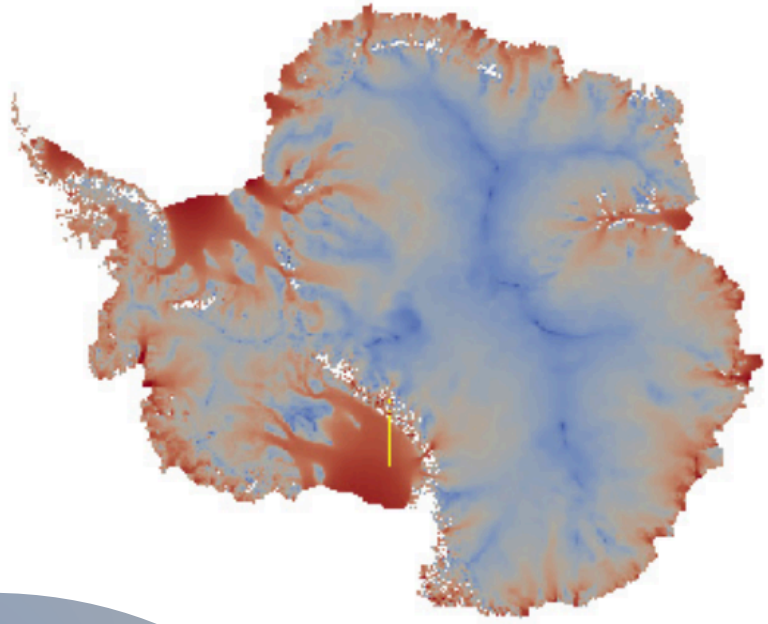
Advisor: Eric Darve

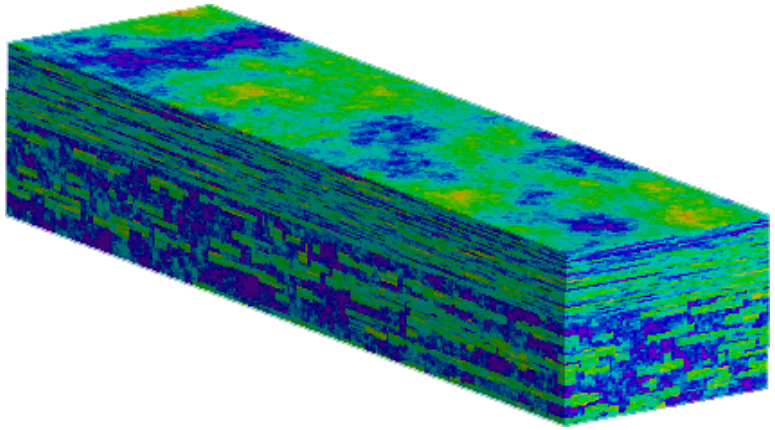
PhD Defense, November 9, 2020

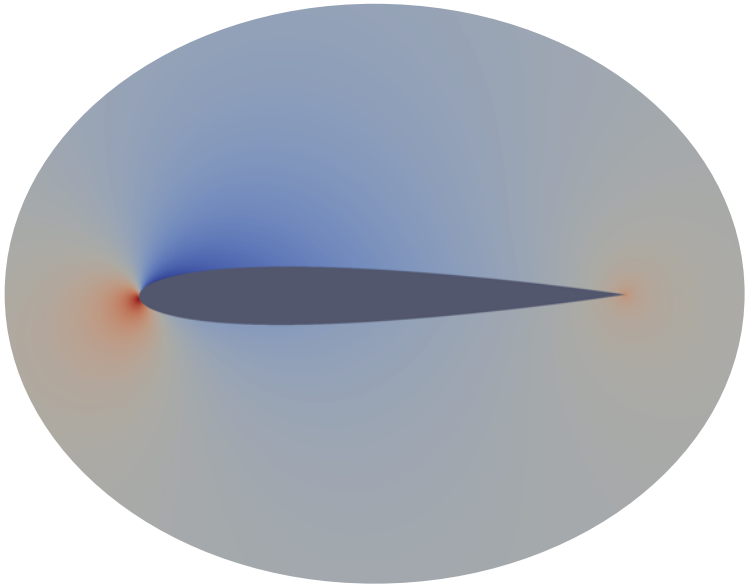
Committee: Juan J. Alonso, Erik G. Boman, Lexing Ying, Michael Saunders (Chair)

Scientific computing

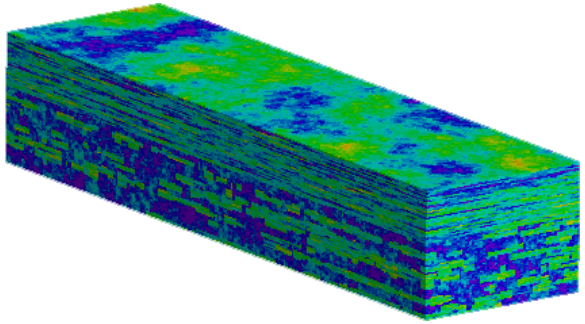
Natural phenomenon
⇒ Modeling
⇒ Discretization
⇒ Computer simulation







Scientific computing needs linear solvers



Linear PDE

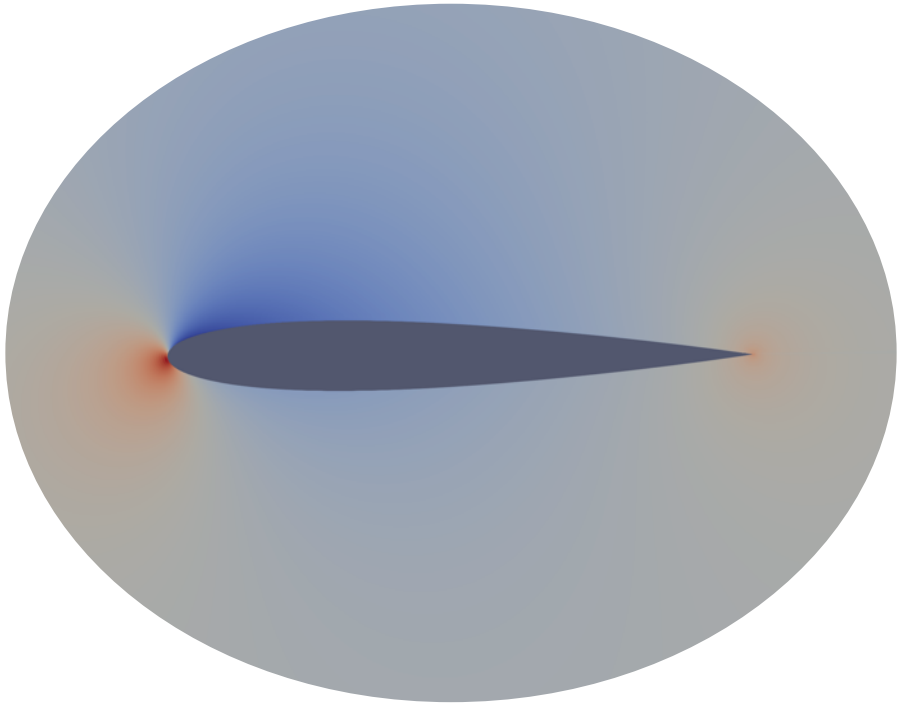
$\nabla \cdot \big(a(x)\nabla u(x)\big) = f(x)$

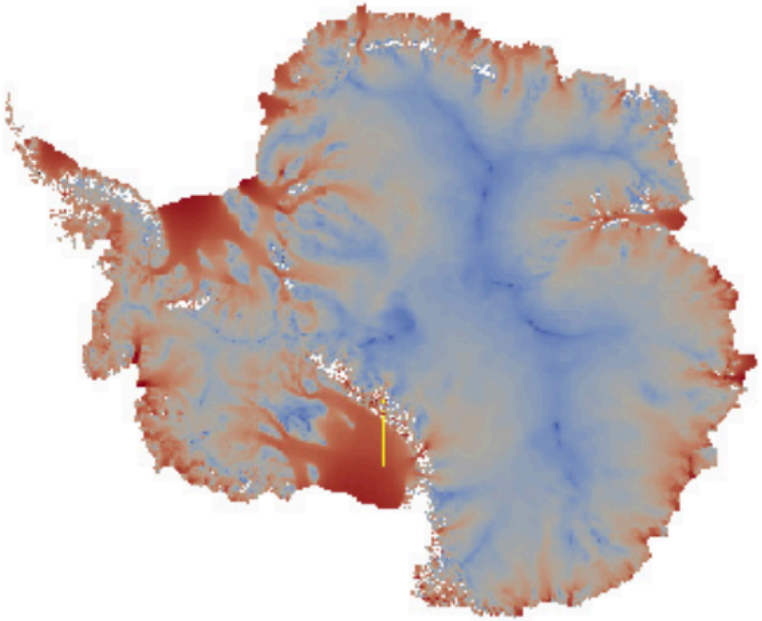
- $Ax = b$
- A sparse, SPD

For Newton steps...

- $Ax = b$
- A sparse, unsymmetric

PDE $F(x) = 0$





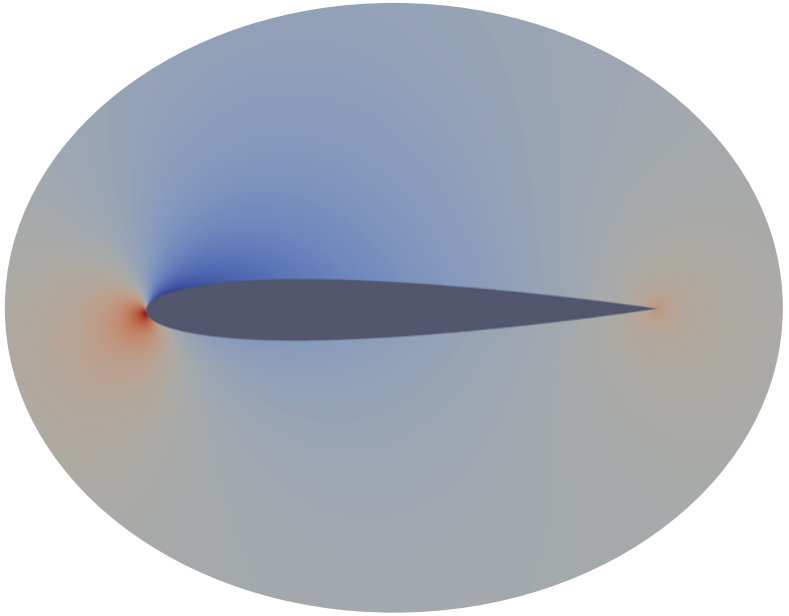
PDE $F(x) = 0$

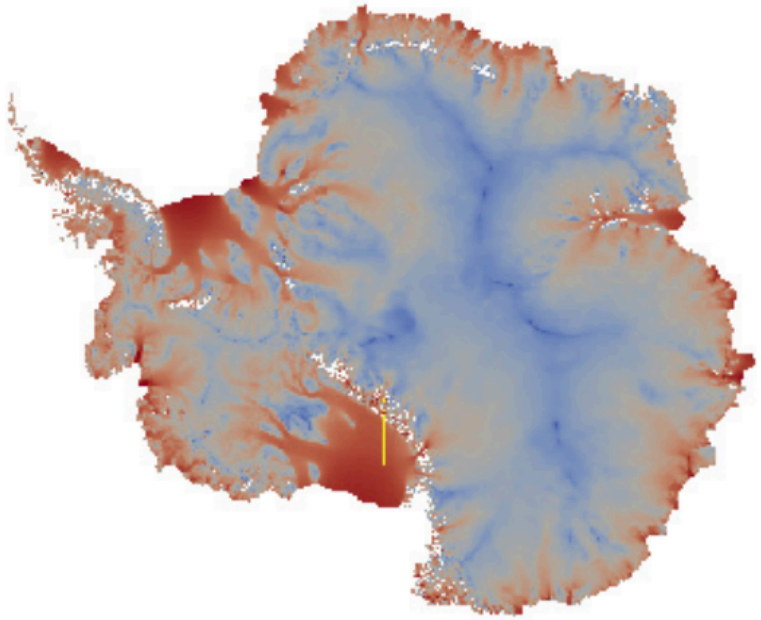
For Newton steps...

- $Ax = b$
- A sparse, SPD

Scientific computing needs parallel computing

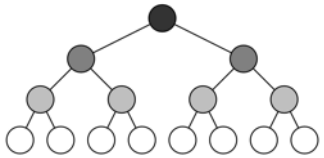
Even fast algorithms cannot solve problems with 100M+ unknowns...





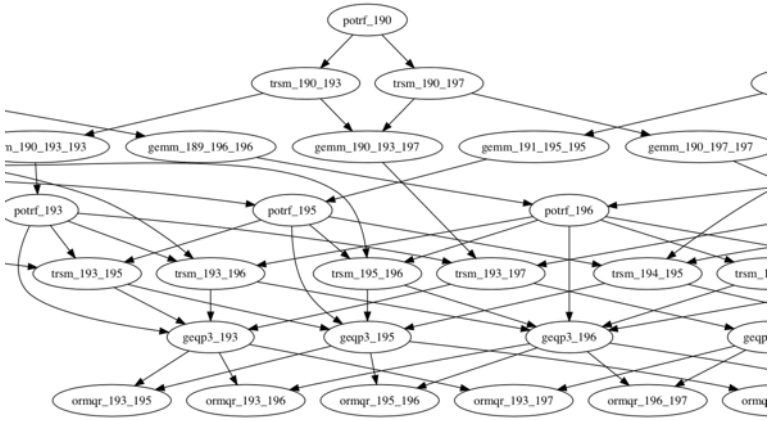
Three-parts talk

Sparsified Nested Dissection (spaND)

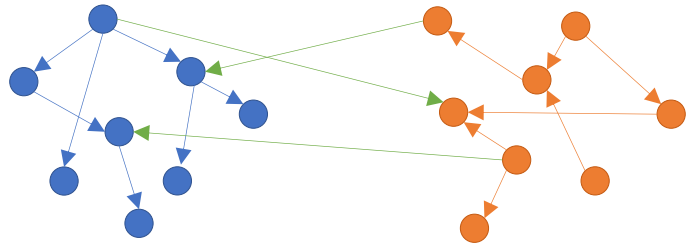


$Ax = b$

Parallel spaND with TaskTorrent



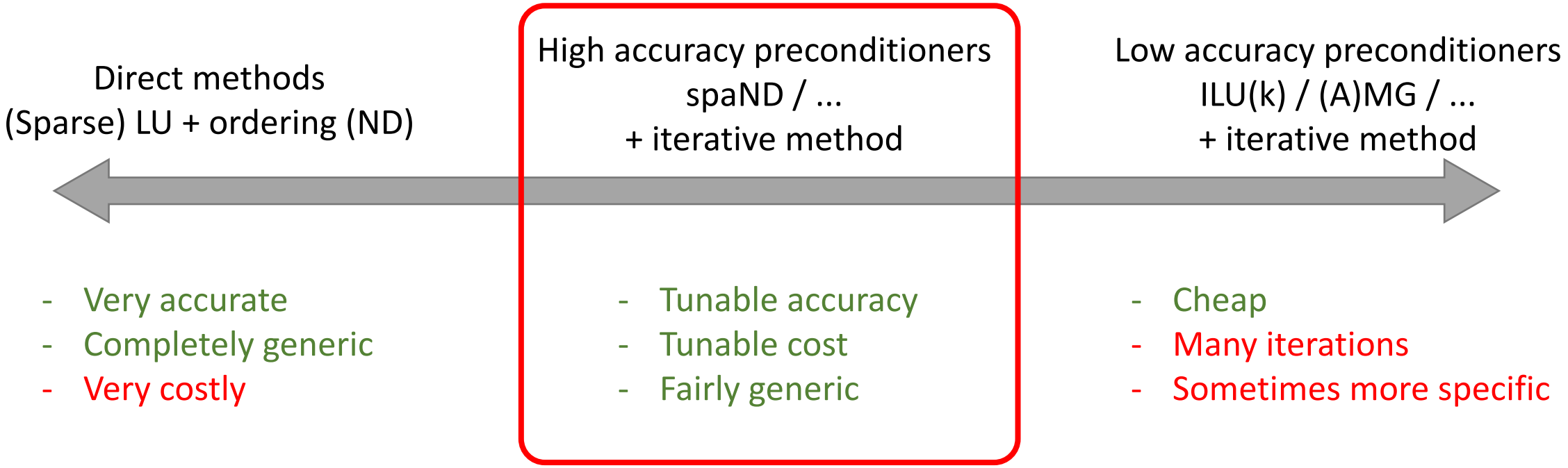
Task-based runtimes TaskTorrent



Sparsified Nested Dissection

Cambier, L., Chen, C., Boman, E. G., Rajamanickam, S., Tuminaro, R. S., & Darve, E. (2020). An algebraic sparsified nested dissection algorithm using low-rank approximations. *SIAM Journal on Matrix Analysis and Applications*, *41*(2), 715-746.

Why do we want hierarchical solvers?



$A = LU$

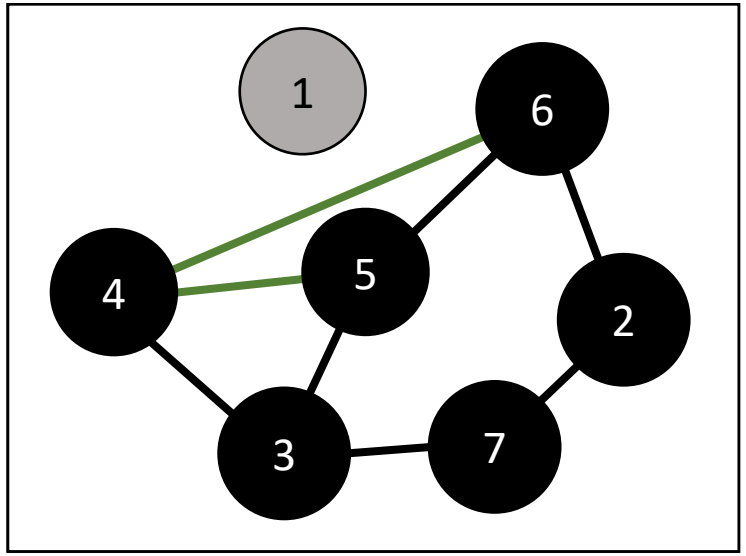
Very slow

$A \approx \prod F_i$

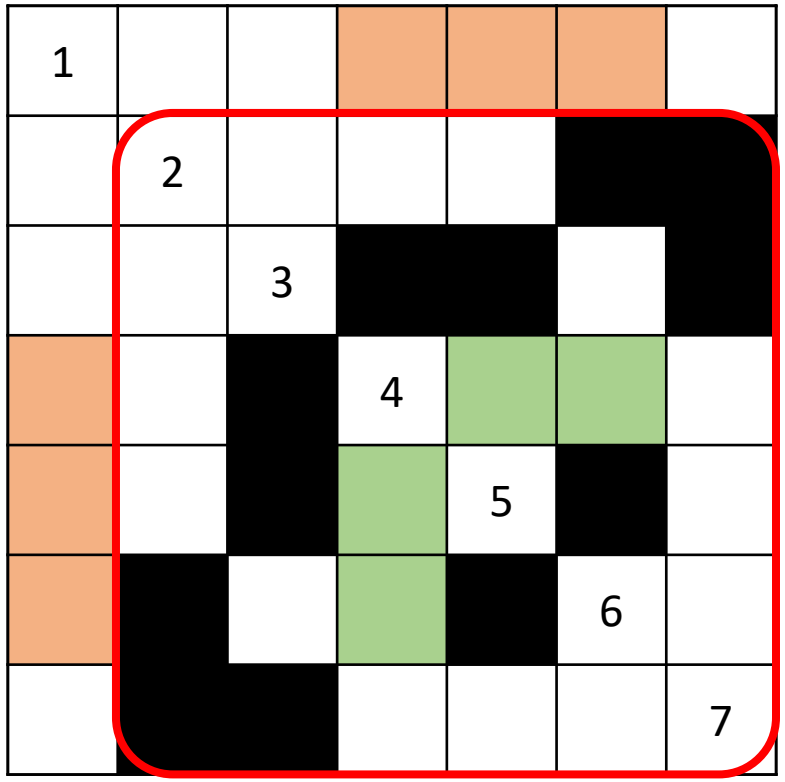
Much faster

Let’s start from direct methods

$$A = \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} I & \\ & A_{22} - A_{21}a_{11}^{-1}A_{12} \end{bmatrix} \begin{bmatrix} l_{11}^T & L_{21}^T \\ & I \end{bmatrix}$$



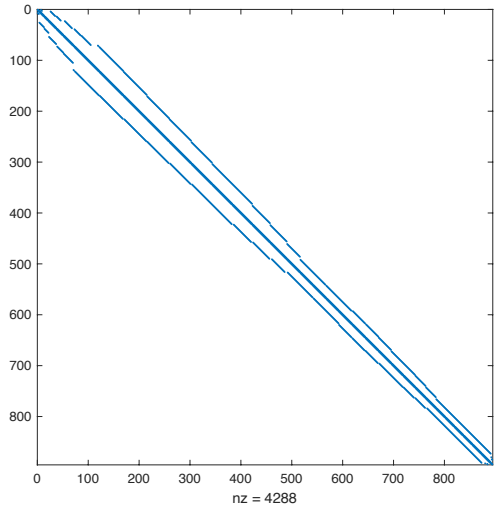
Matrix graph $G = (V, E)$
$(i, j) \in E \Leftrightarrow A_{ij} \neq 0$



A, symmetric, SPD

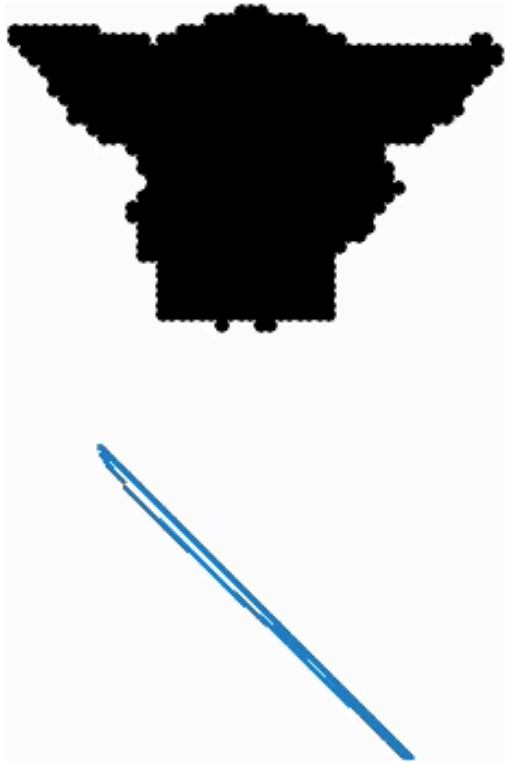
Not all orderings are equal

A (4k nnz)



graph of A

Natural ordering
$A = LL^T$

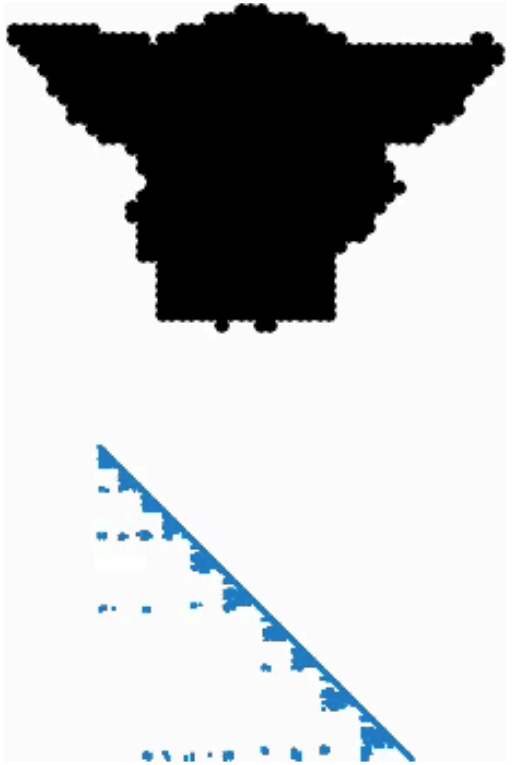


$A \mapsto L$ (26k nnz)

AMD ordering
$P_1 A P_1^T = L_1 L_1^T$

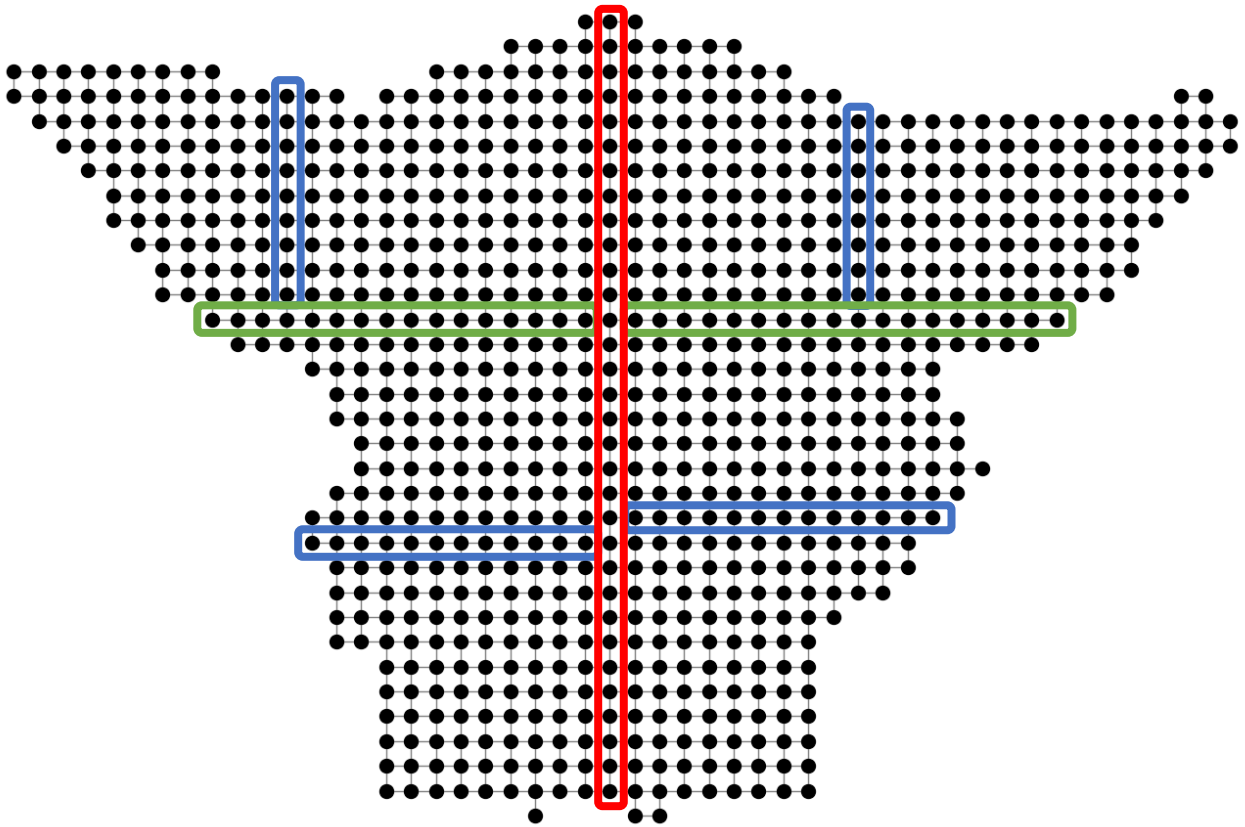
$P_1 A P_1^T \mapsto L_1$ (8k nnz)

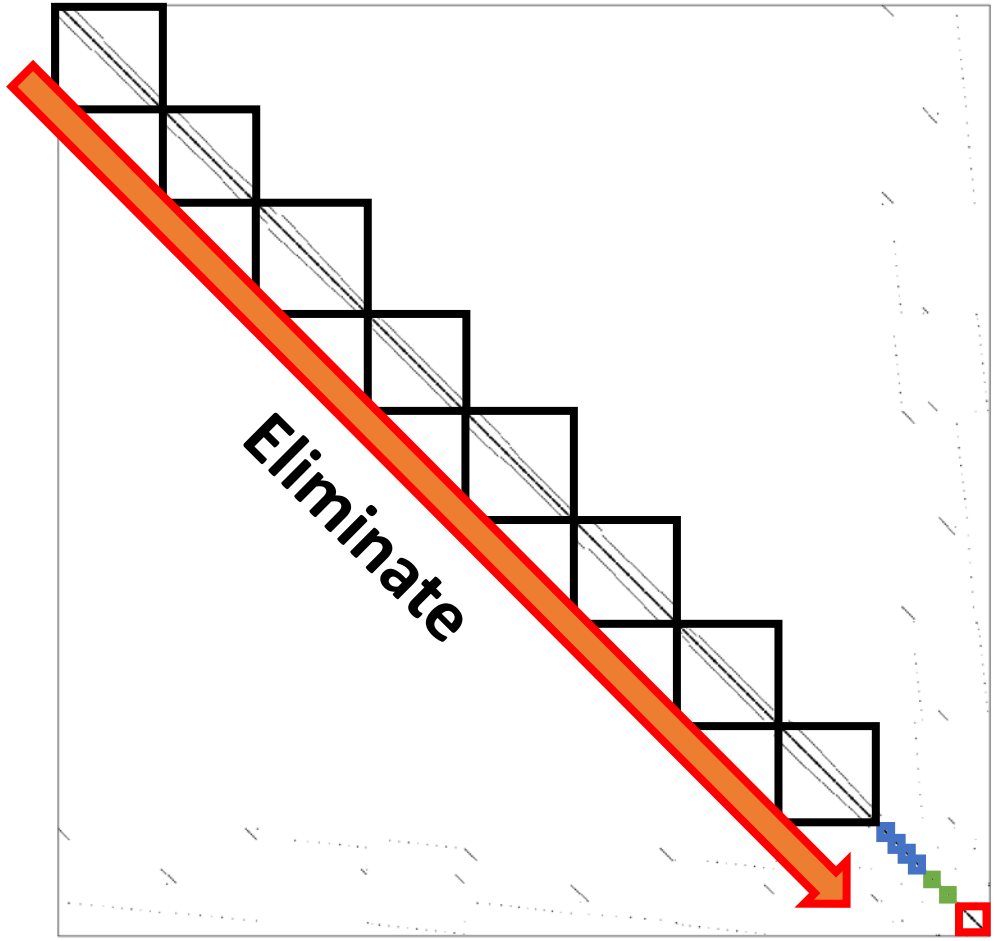
ND ordering
$P_2 A P_2^T = L_2 L_2^T$

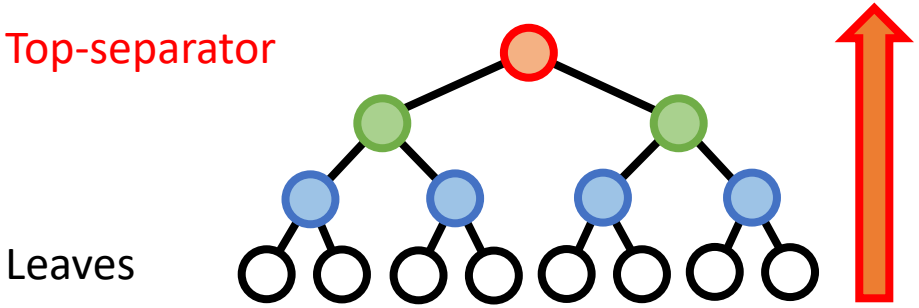


$P_2 A P_2^T \mapsto L_2$ (9k nnz)

How to minimize fill-in & cost? Nested Dissection!

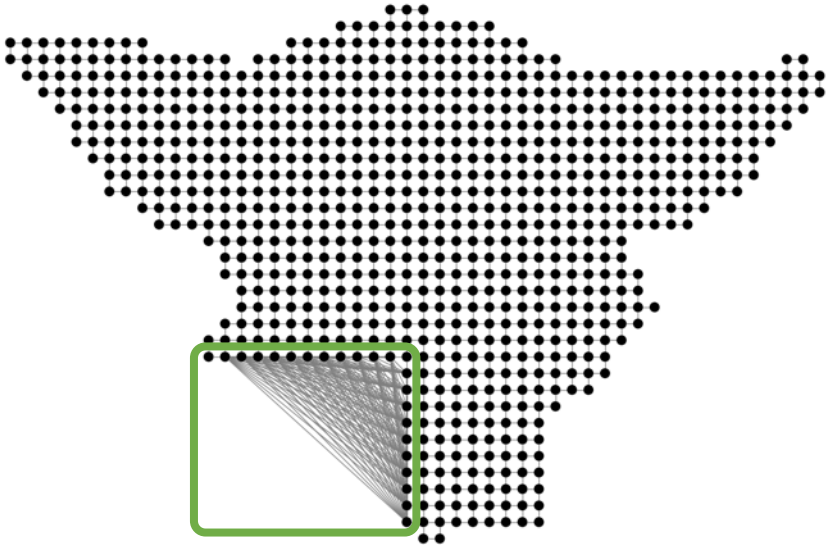






Block elimination

Graph of A



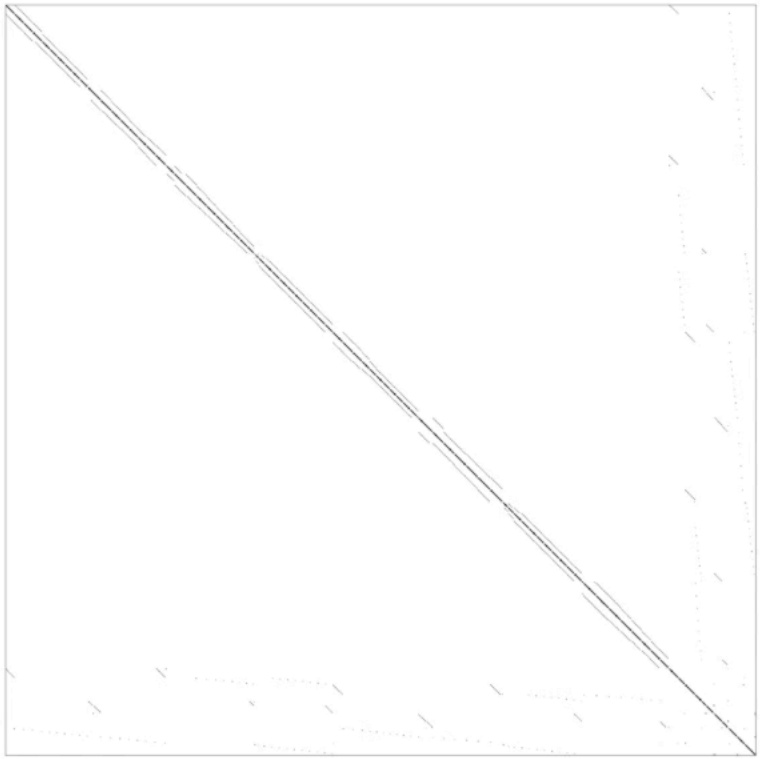
Fill-ins

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1} = \begin{bmatrix} I & & \\ & A_{nn} - A_{np}A_{pp}^{-1}A_{pn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

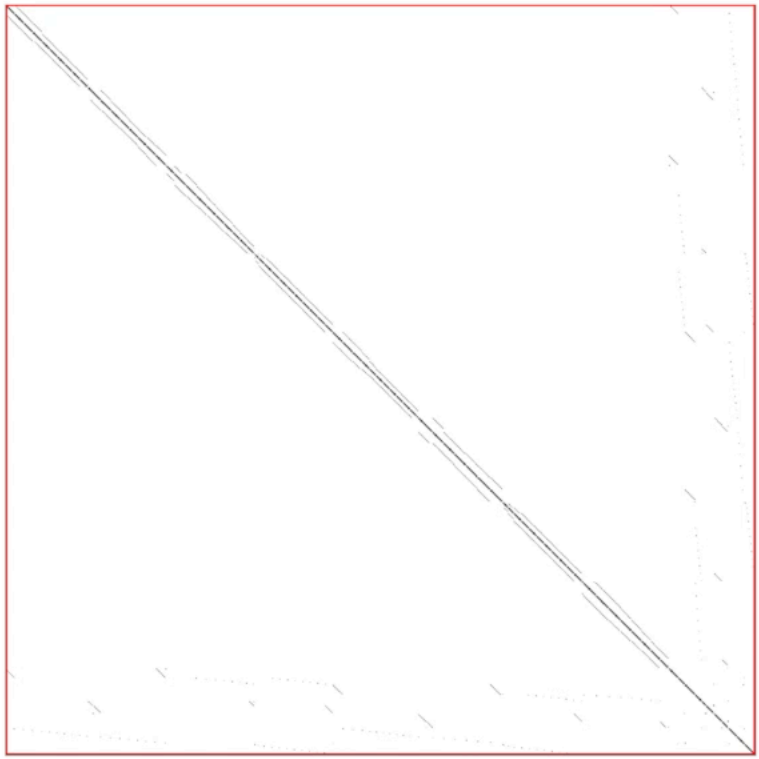
Fill-in in Nested Dissection

Graph of A

Zoom on trailing matrix



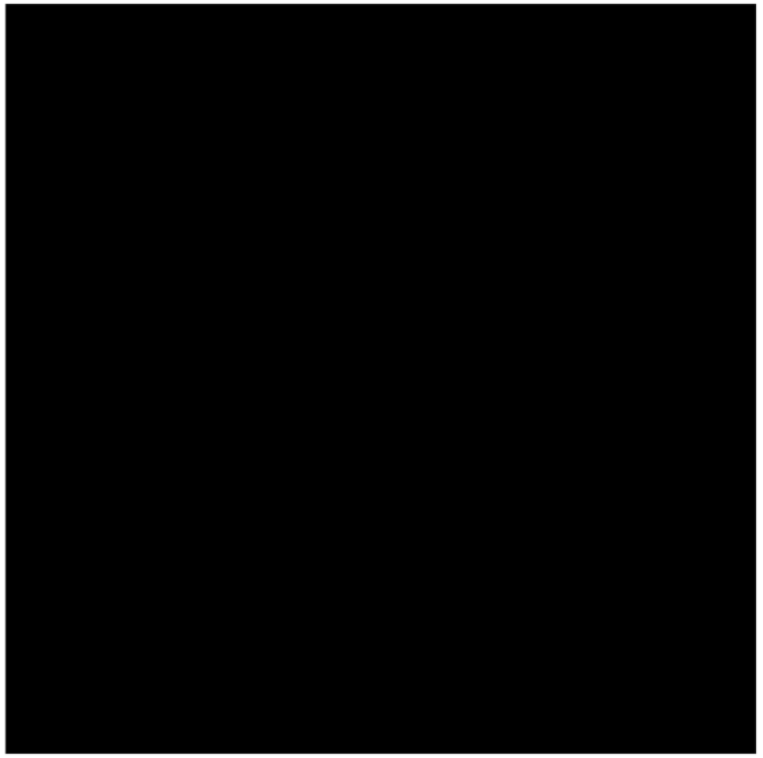
Trailing matrix

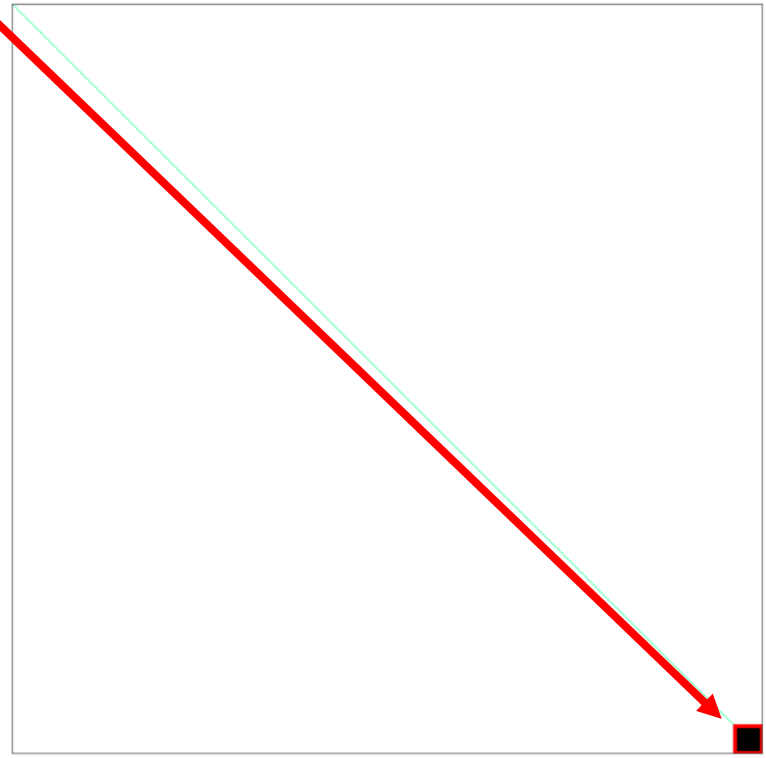


Direct methods have a lot of fill-ins

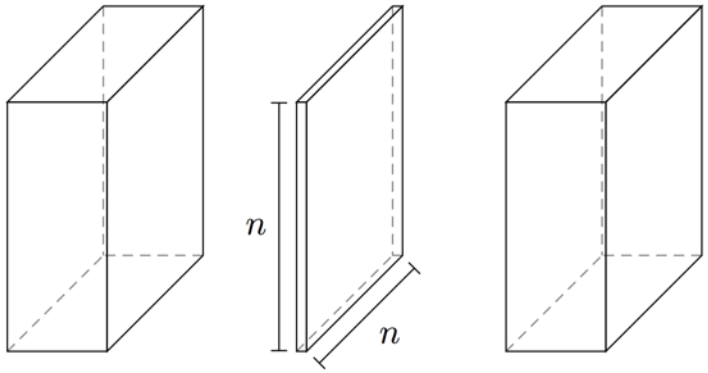
Dense block! Cholesky costs $O(m^3)$

The largest separator, size m

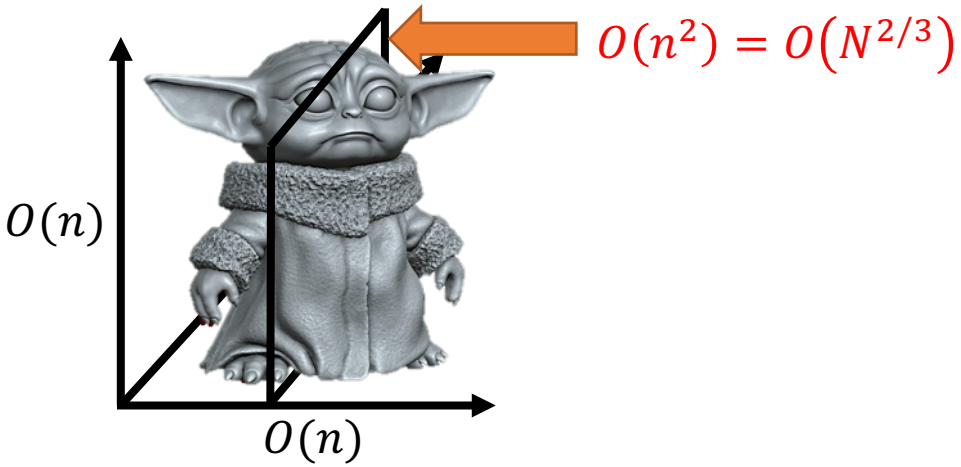


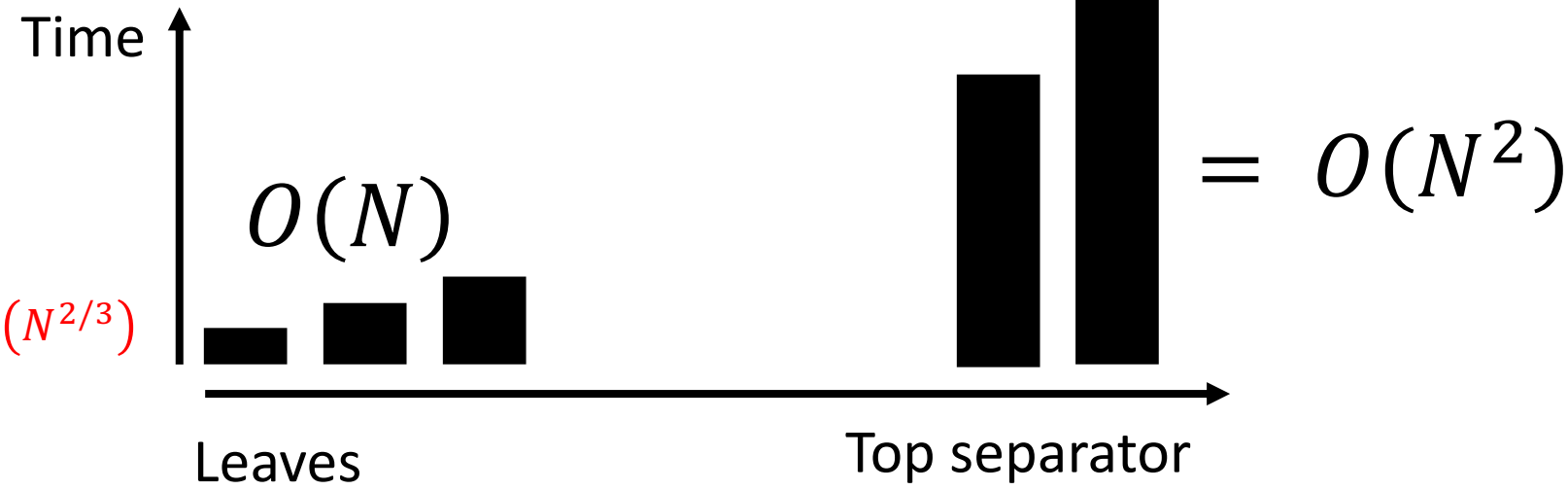


Dense blocks take too long to factor



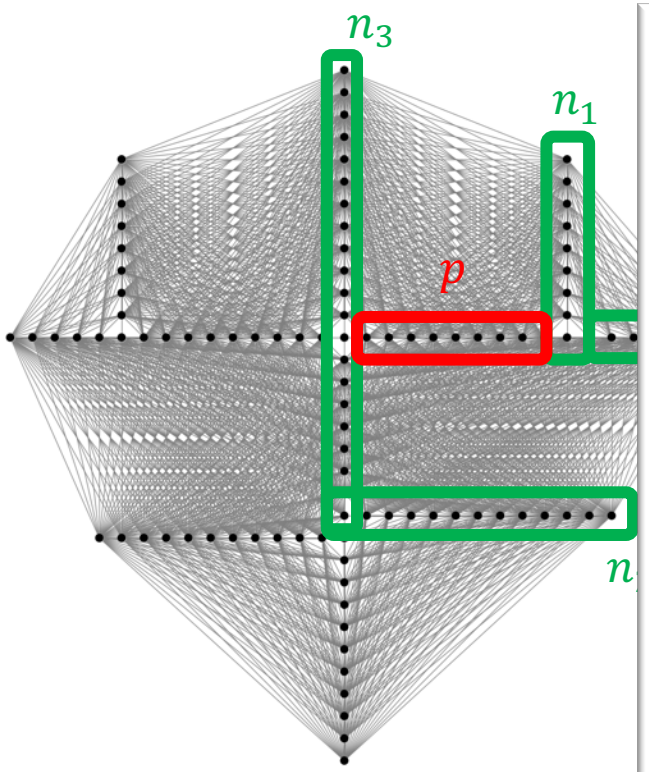
Factoring $m = n^2 = N^{\frac{2}{3}}$
takes $O(m^3) = O(N^2)$



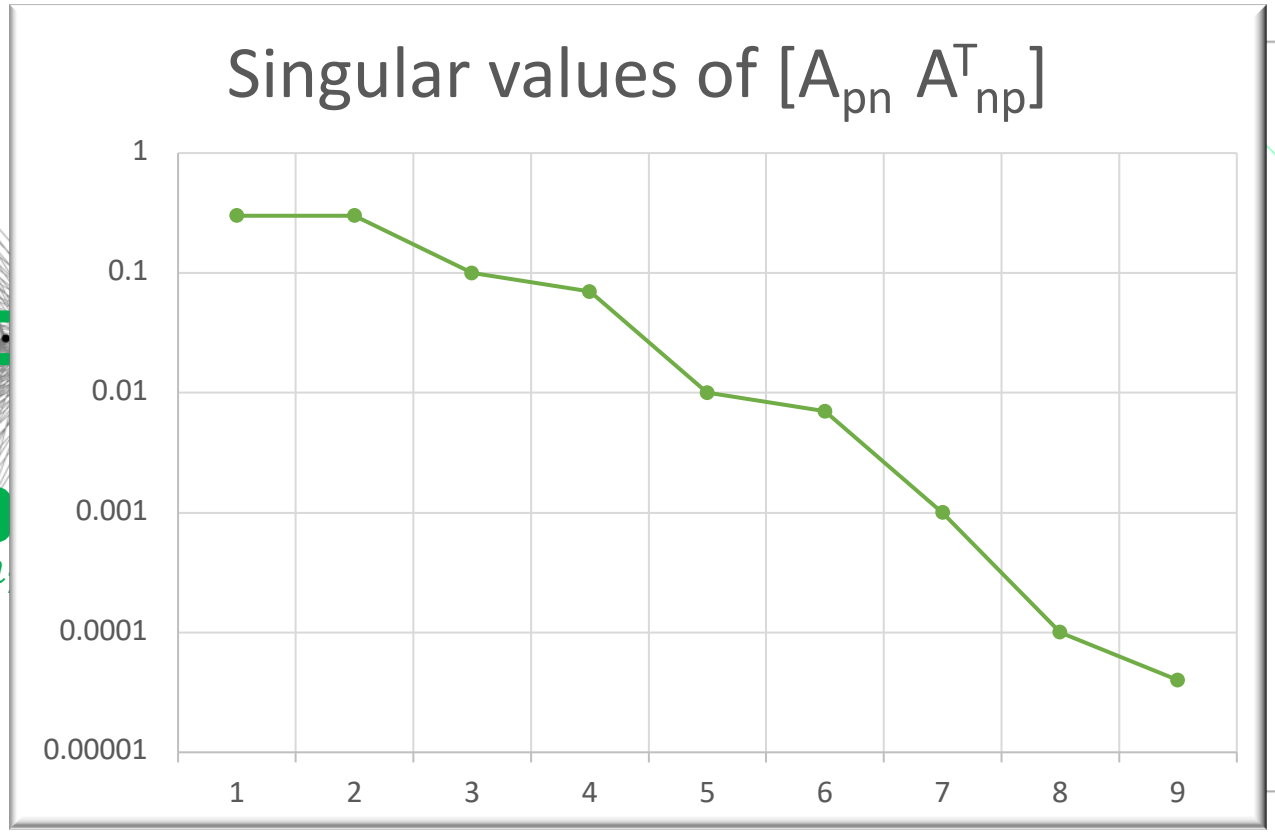


Fill-in is dense but low-rank

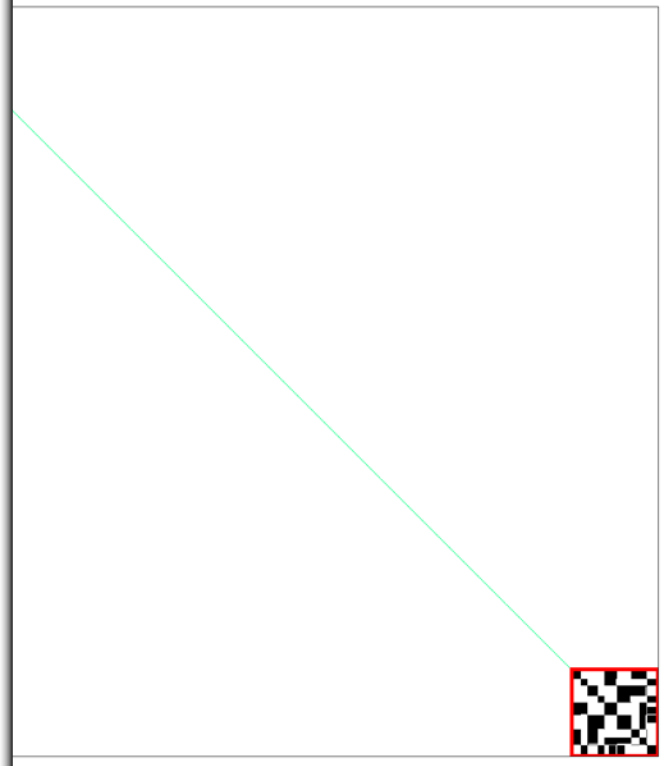
Graph of A



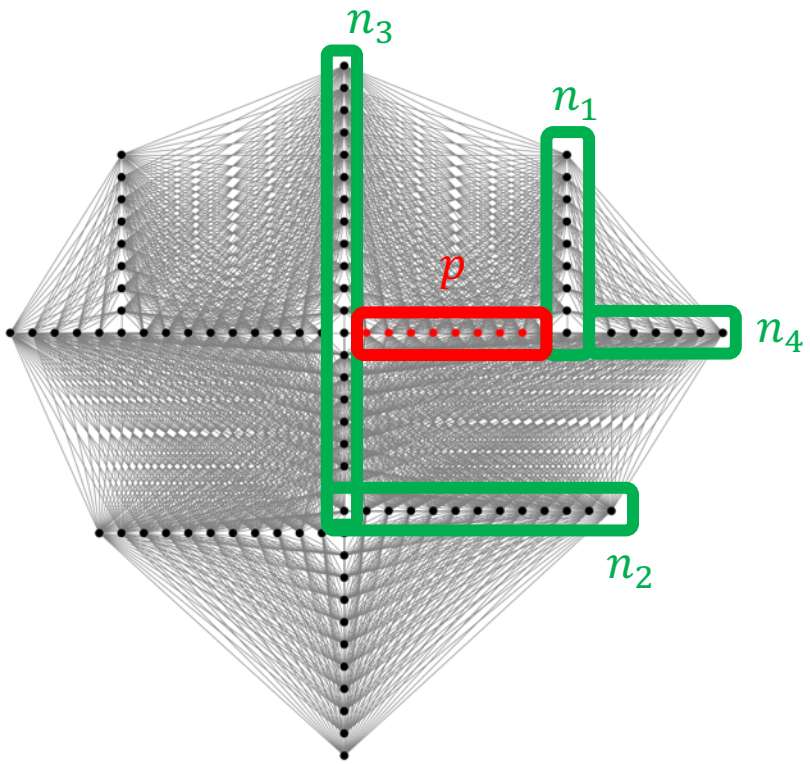
Zoom on trailing matrix



Trailing matrix



Sparsification: block scaling

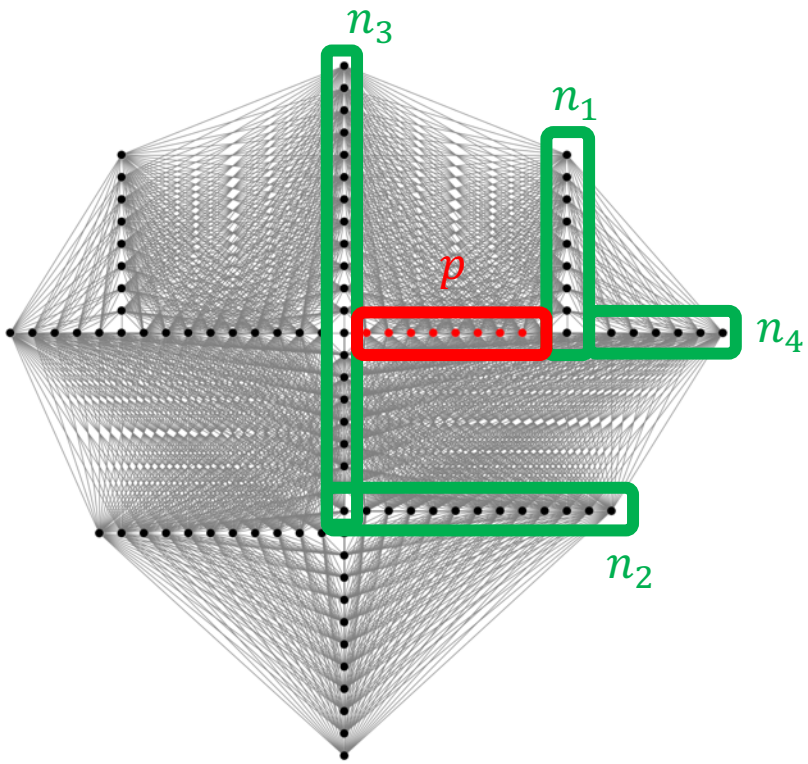


Not strictly needed

Very important for accuracy

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1} = \begin{bmatrix} I & B_{pn} & \\ B_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

Sparsification: low-rank approximation



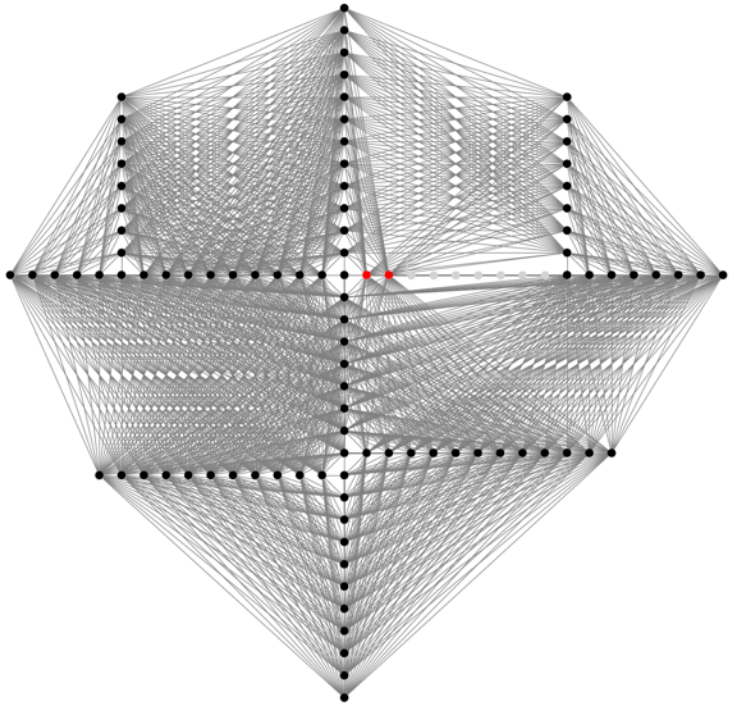
$$[A_{pn} \quad A_{np}^T] = [A_{pn_1} \quad A_{pn_2} \quad \dots \quad A_{n_1p}^T \quad A_{n_2p}^T \quad \dots]$$

$$= [Q_c \quad Q_f] \begin{bmatrix} W \\ O(\varepsilon) \end{bmatrix} \quad \text{(RRQR, SVD, ...)}$$

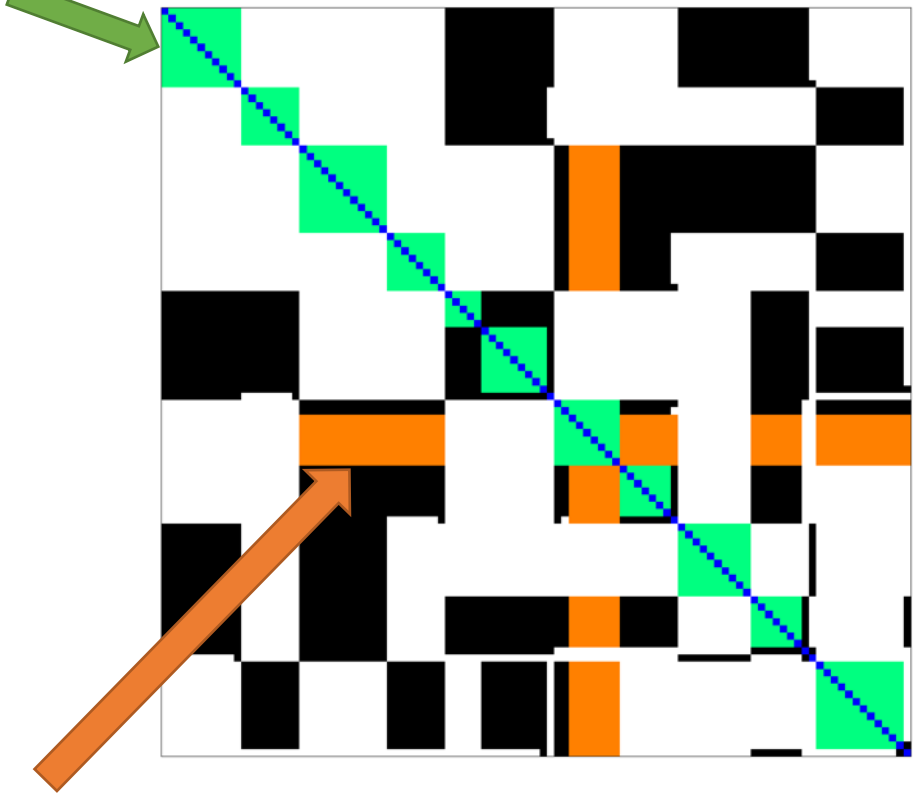
$$\begin{bmatrix} \begin{bmatrix} Q_f^T \\ Q_c^T \end{bmatrix} & \\ & I \end{bmatrix} \begin{bmatrix} I & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} \begin{bmatrix} [Q_f \quad Q_c] & \\ & I \end{bmatrix}$$

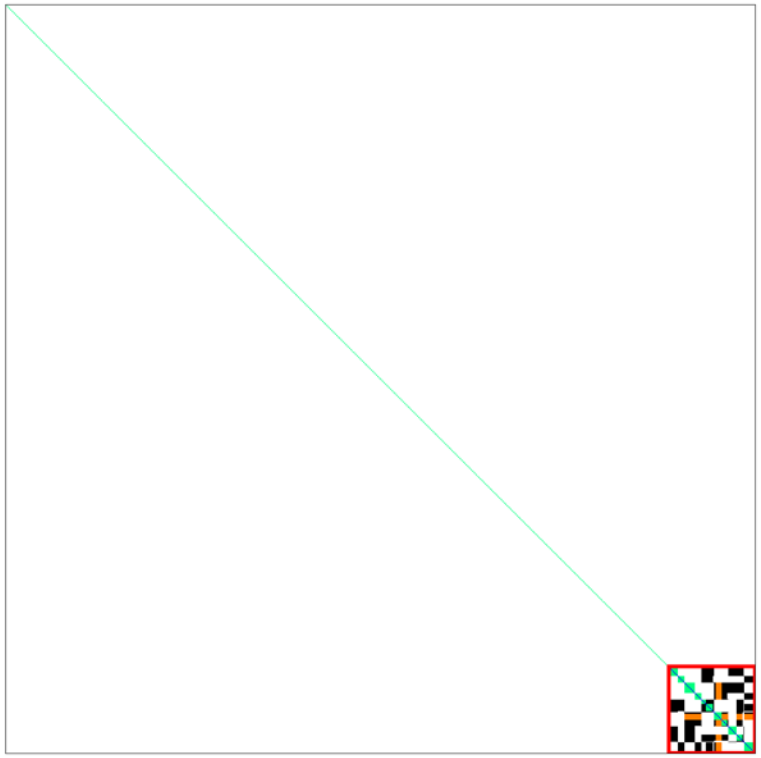
$$= \begin{bmatrix} I & \\ & \begin{bmatrix} I & W_{cn} & \\ W_{nc} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} \end{bmatrix}$$

Sparsification: low-rank approximation



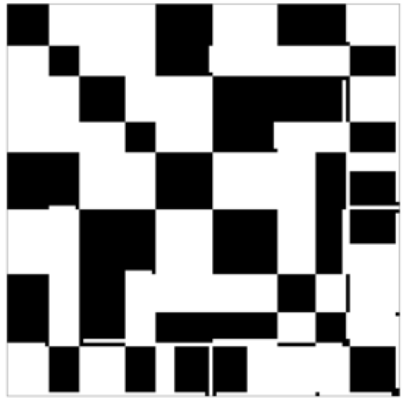
Identity

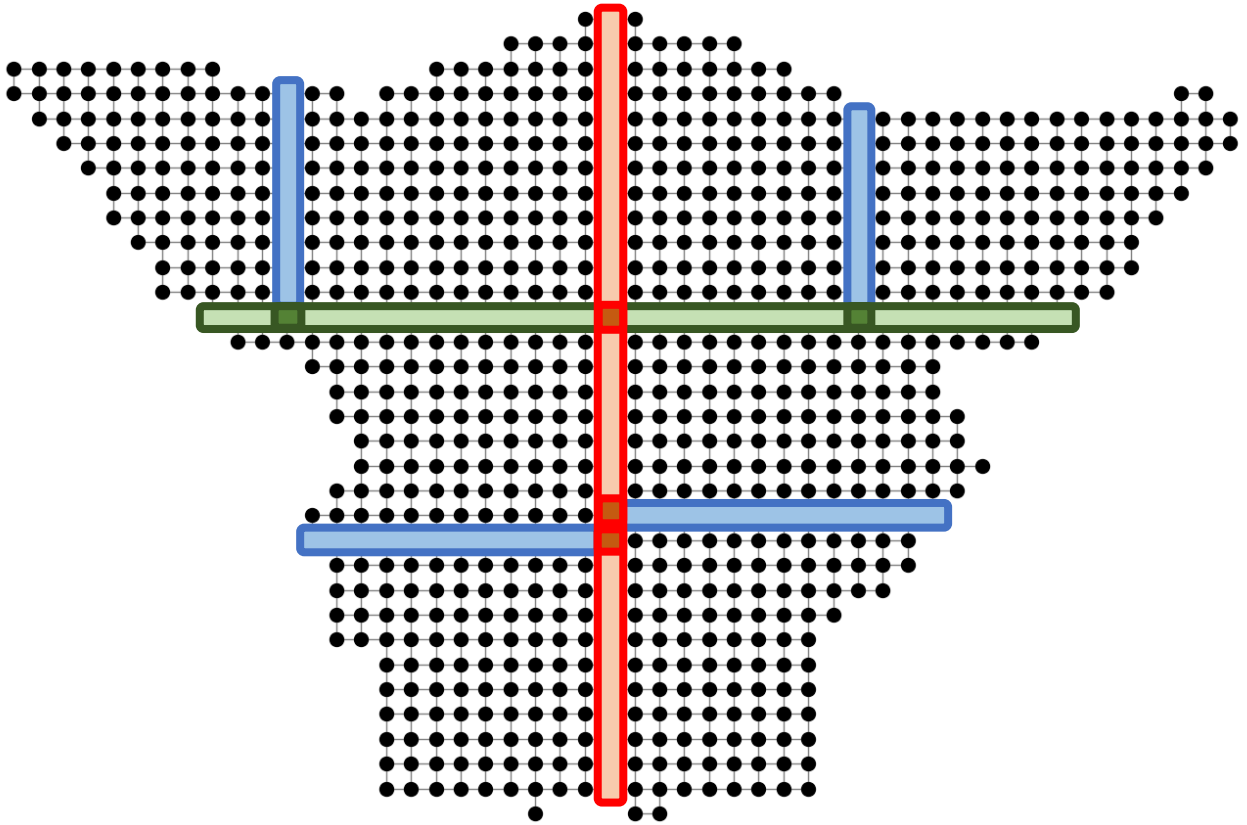


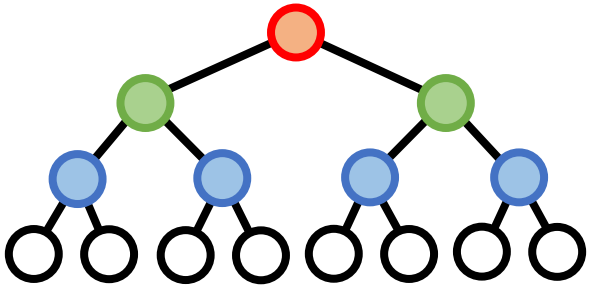


$O(\varepsilon) \approx 0 \mapsto$ ignore

Defining interfaces through overlapping separators







Eliminate ↦ Scale ↦ Sparsify ↺

For level k = 1, …, L

- Eliminate interiors (LL^T, LDL^T, PLU, PLUQ)

Fill-in;
limited by separators

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1} = \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

- Scale interfaces (LL^T, LDL^T, PLU, PLUQ)

No fill-in;
No approximations

$$\begin{bmatrix} L^{-1} & \\ & I \end{bmatrix}\begin{bmatrix} A_{pp} & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix}\begin{bmatrix} U^{-1} & \\ & I \end{bmatrix} = \begin{bmatrix} I & L^{-1}A_{pn} \\ A_{np}U^{-1} & A_{nn} \end{bmatrix}$$

- Sparsify interfaces (RRQR) (if left and right are eliminated)

No fill-in!
Approximation

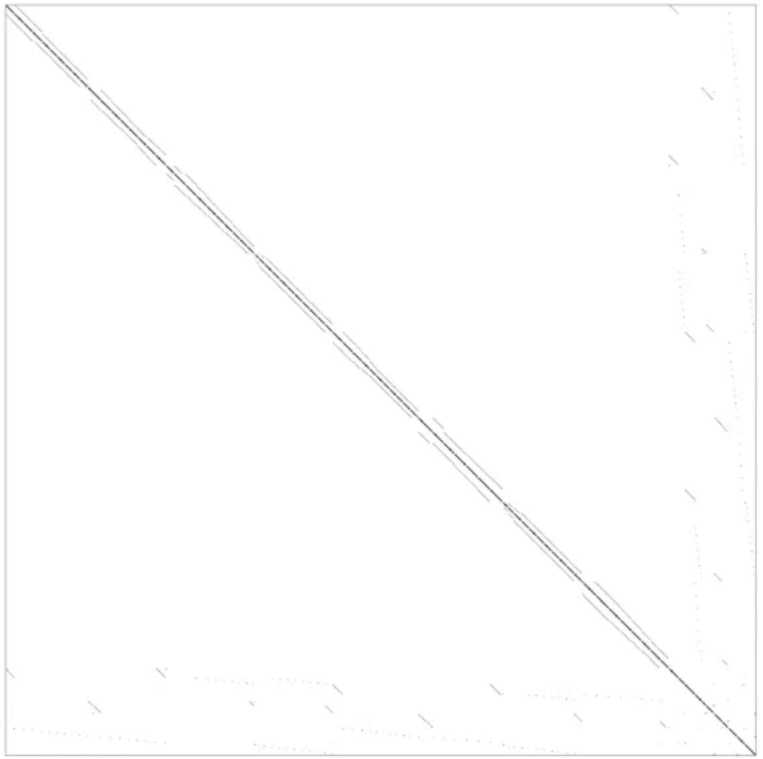
$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix}\begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix}\begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn} \end{bmatrix} \approx \begin{bmatrix} I & & \\ & I & W_{cn} \\ & W_{nc} & A_{nn} \end{bmatrix}$$

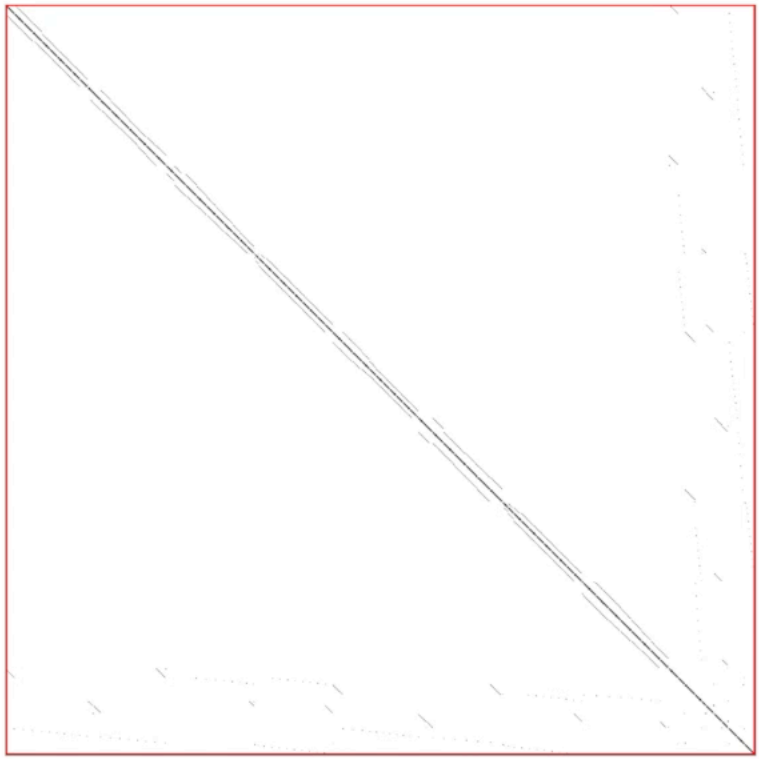
- Merge clusters

Sparse lower-triangular
and sparse orthogonal

Result: $A \approx \prod F_i \Rightarrow$ Preconditioner for CG or GMRES

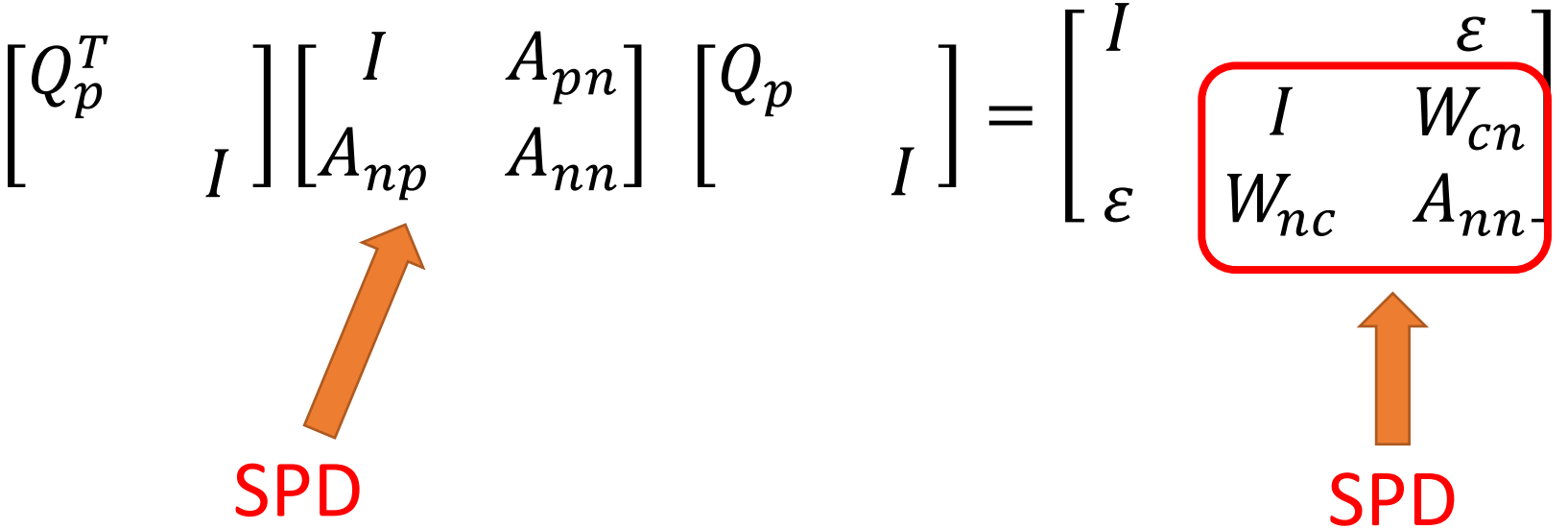
spaND in action



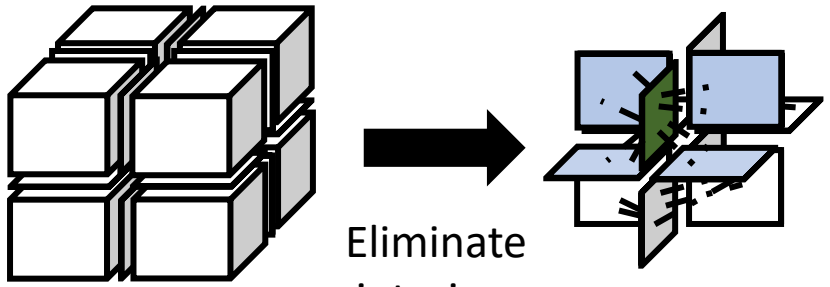


SPD remains SPD

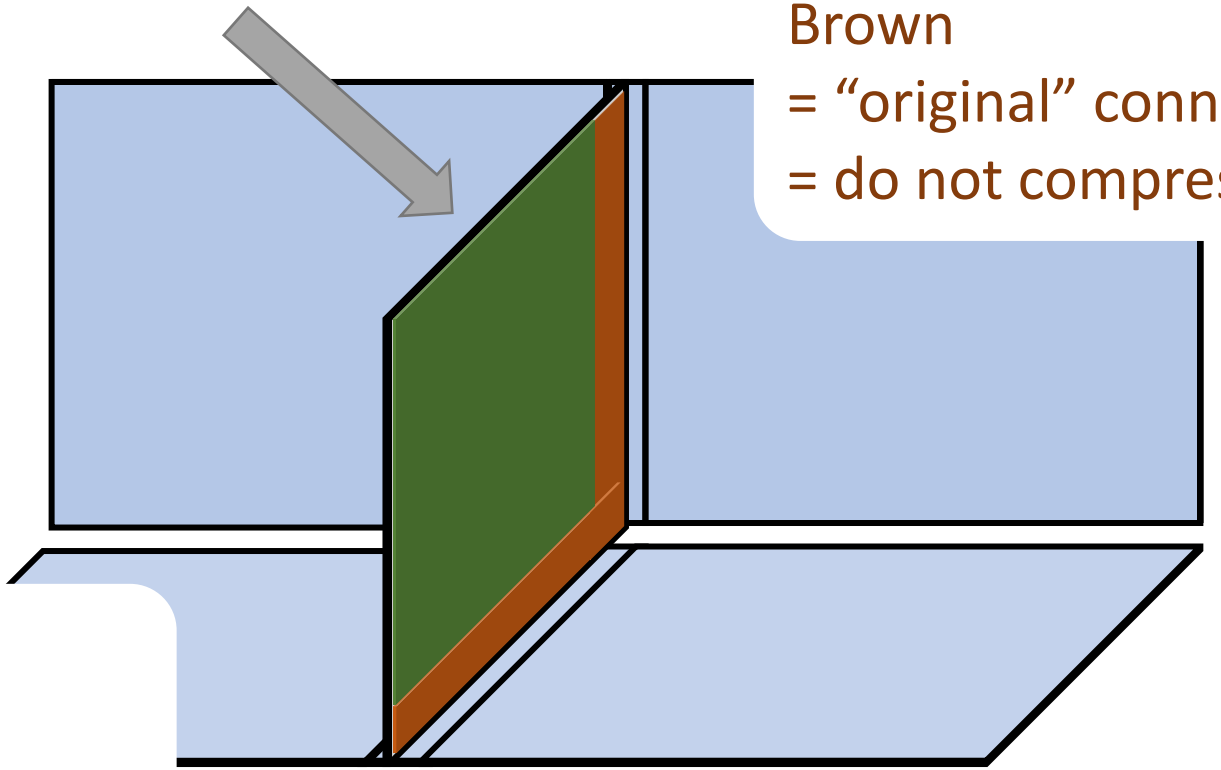
If using Cholesky for elimination and scaling



Separator sizes?



Interface (Brown + Green)



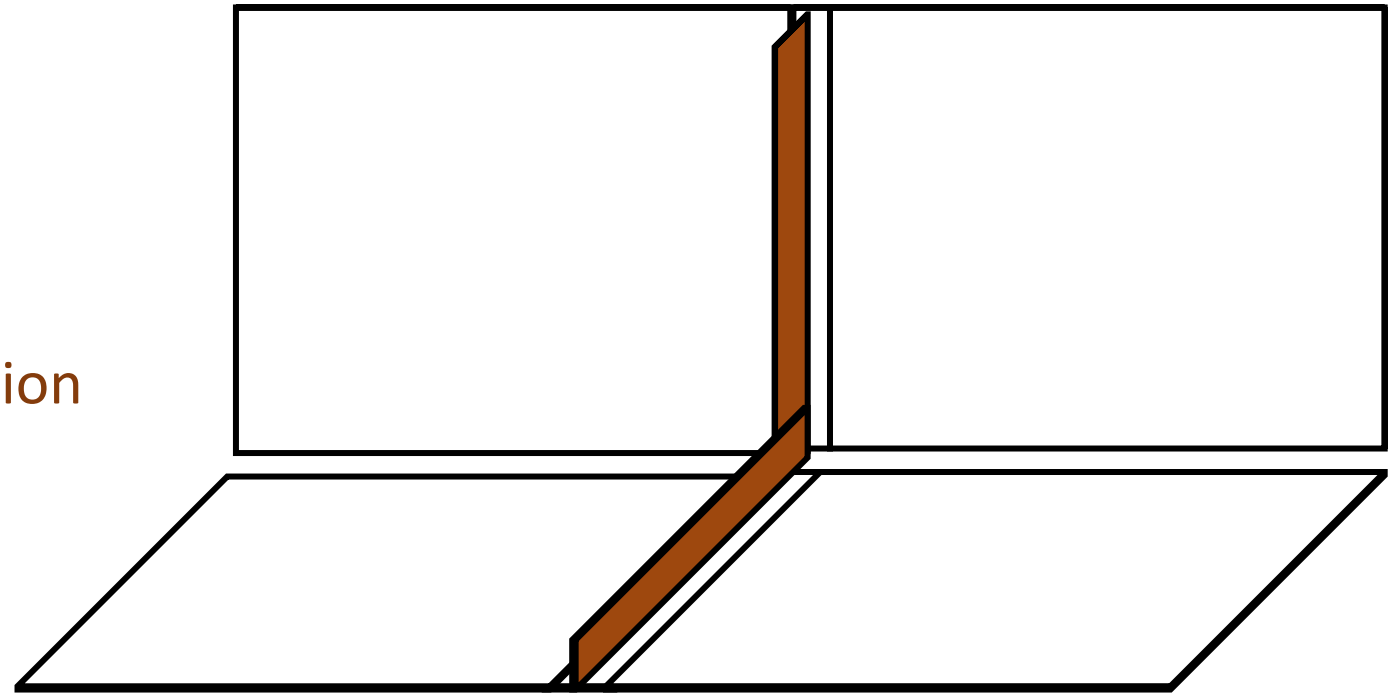
Brown
= “original” connections
= do not compress well

Green
⇒ fill-ins
⇒ compress well

Separator sizes decrease $\mathrm{O}\left(N^{\frac{2}{3}}\right) \Rightarrow \mathrm{O}\left(N^{\frac{1}{3}}\right)$

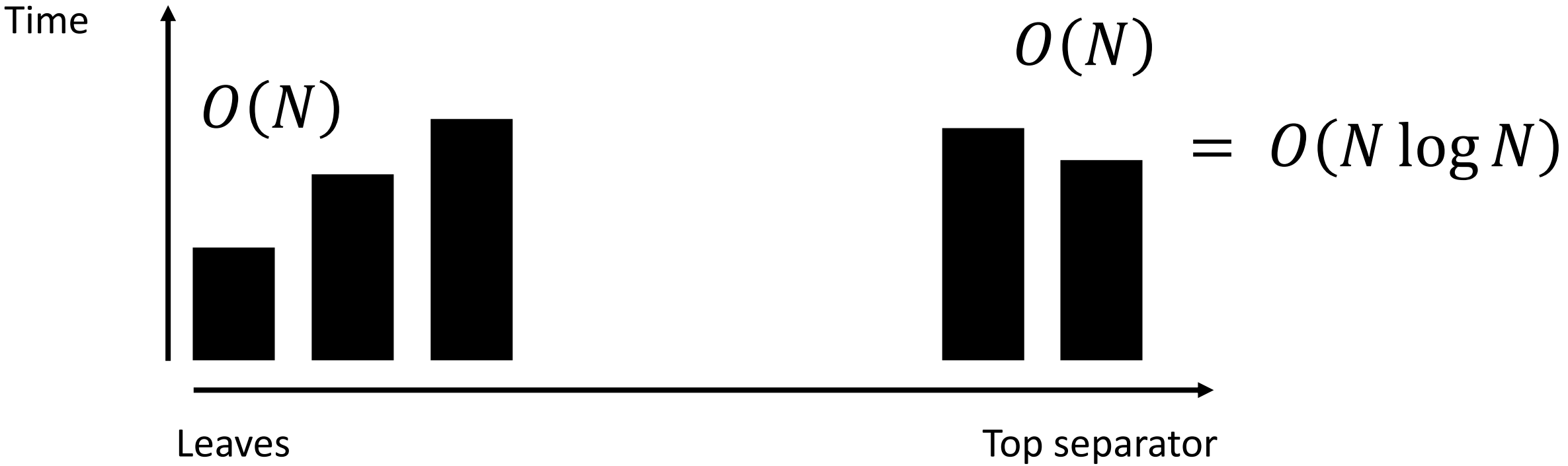
"planes" "lines"

Brown
⇒ what is left
after sparsification



spaND is $O(N \log N)$ in 3D

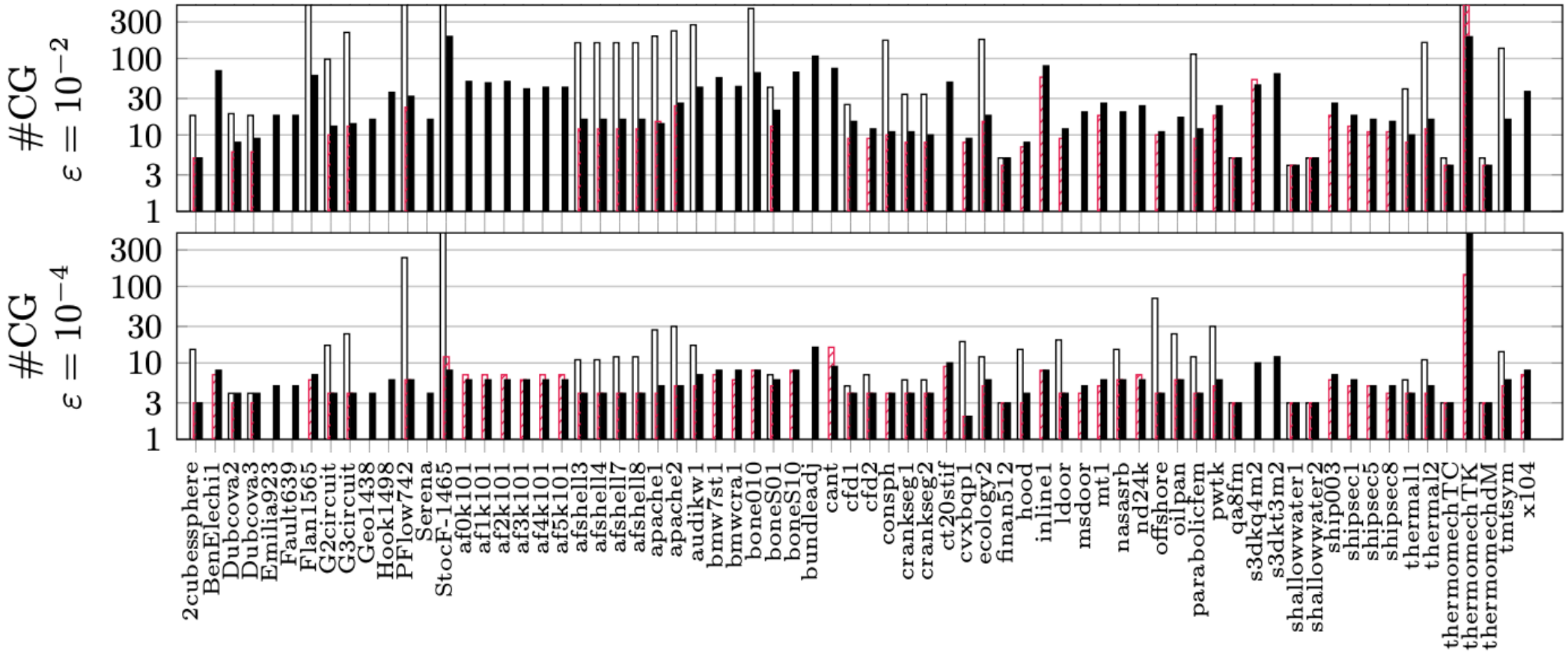
<u>If separators</u>* $N^{\frac{2}{3}} \rightarrow N^{\frac{1}{3}}$



* and few other assumptions

Orthogonal always works on SPD
Scaling improves accuracy

Black = spaND (Orthogonal, scaling)
Red = HIF* (Triangular, scaling)
White = HIF* (Triangular, no scaling)

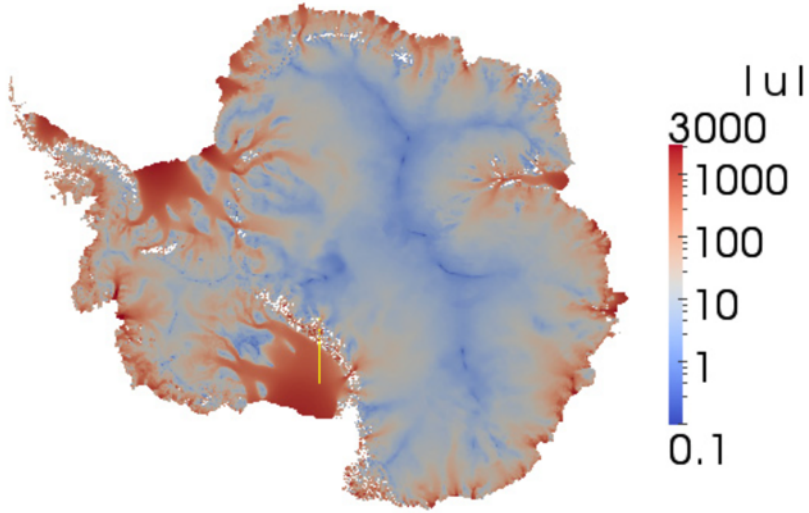


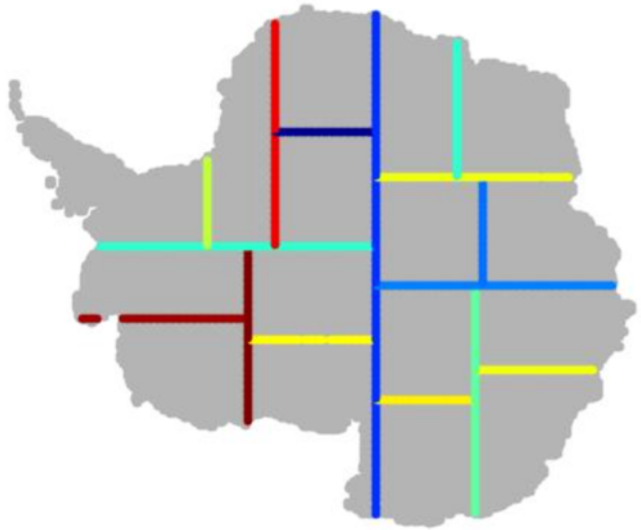
All >50k SPD problems from SuiteSparse, except 2 that didn't converge in <500

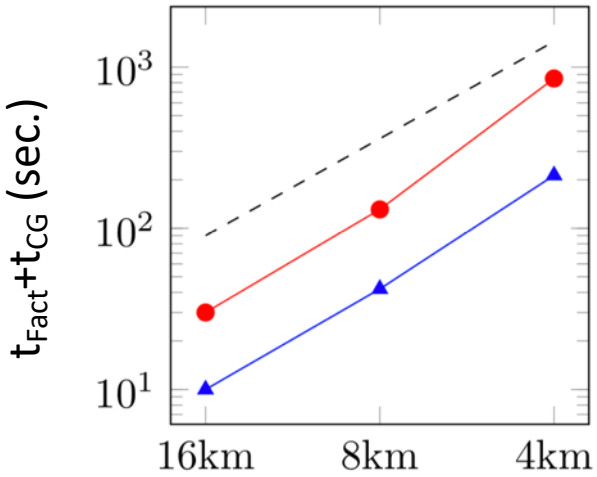
* Ho, K. L., & Ying, L. (2016). Hierarchical interpolative factorization for elliptic operators: differential equations. *Communications on Pure and Applied Mathematics*, *69*(8), 1415-1451.

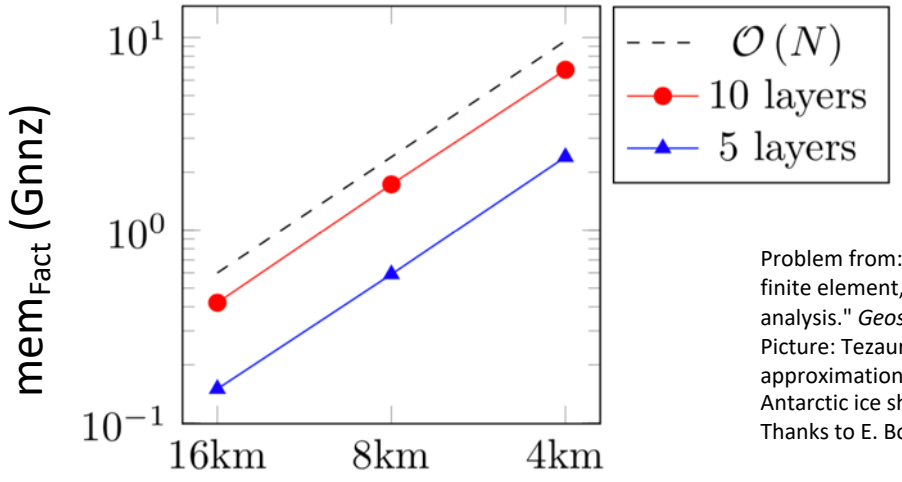
Ice-Sheet modeling $\kappa(A) > 10^{11}$

		spaND				Direct	ILU(0) (Trilinos)
	N	t_{Fact} (s.)	t_{CG} (s.)	n_{CG}	$size_{Top}$	$t_{Fact}+t_{Solve}$ (s.)	t_{Solve} (s.)
5 layers	0.6M (16km)	7	3	7	78	19	23
	2.5M (8km)	28	14	8	88	126	286
	10M (4km)	124	89	10	99	1036	7137
10 layers	1M (16km)	23	7	7	137	86	42
	4.6M (8km)	97	34	8	147	725	544
	18.5M (4km)	538	311	10	159	-	18680





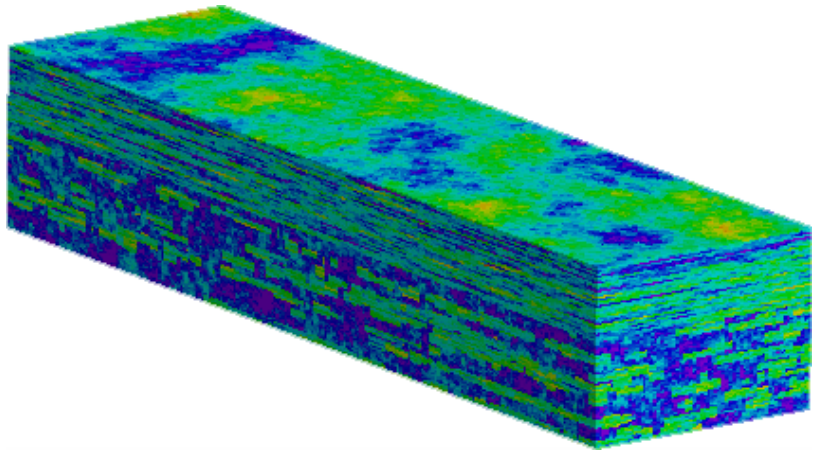


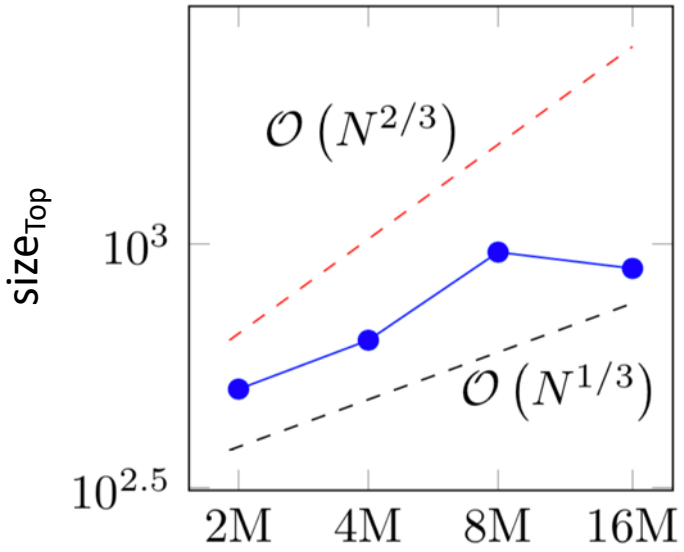


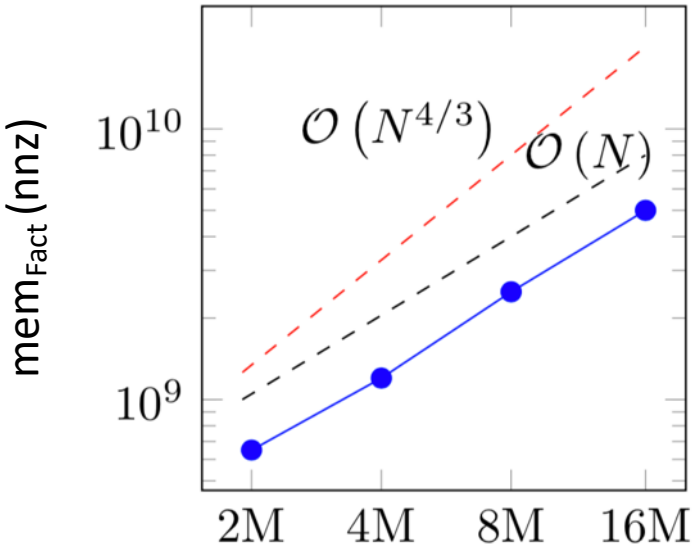
Problem from: Tezaur, Irina K., et al. "Albany/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis." *Geoscientific Model Development (Online)* 8.4 (2015).
Picture: Tezaur, Irina K., et al. "On the scalability of the Albany/FELIX first-order Stokes approximation ice sheet solver for large-scale simulations of the Greenland and Antarctic ice sheets." *Procedia Computer Science* 51 (2015): 2026-2035.
Thanks to E. Boman, R. Tuminaro, S. Rajamanickam & M. Perego

The SPE problem

	spaND				Direct
N	t_{Fact} (s.)	t_{CG} (s.)	n_{CG}	$size_{Top}$	$t_{Fact}+t_{Solve}$ (s.)
2M	55	18	13	504	743
4M	118	44	14	635	3677
8M	254	102	16	962	-
16M	650	256	14	891	-





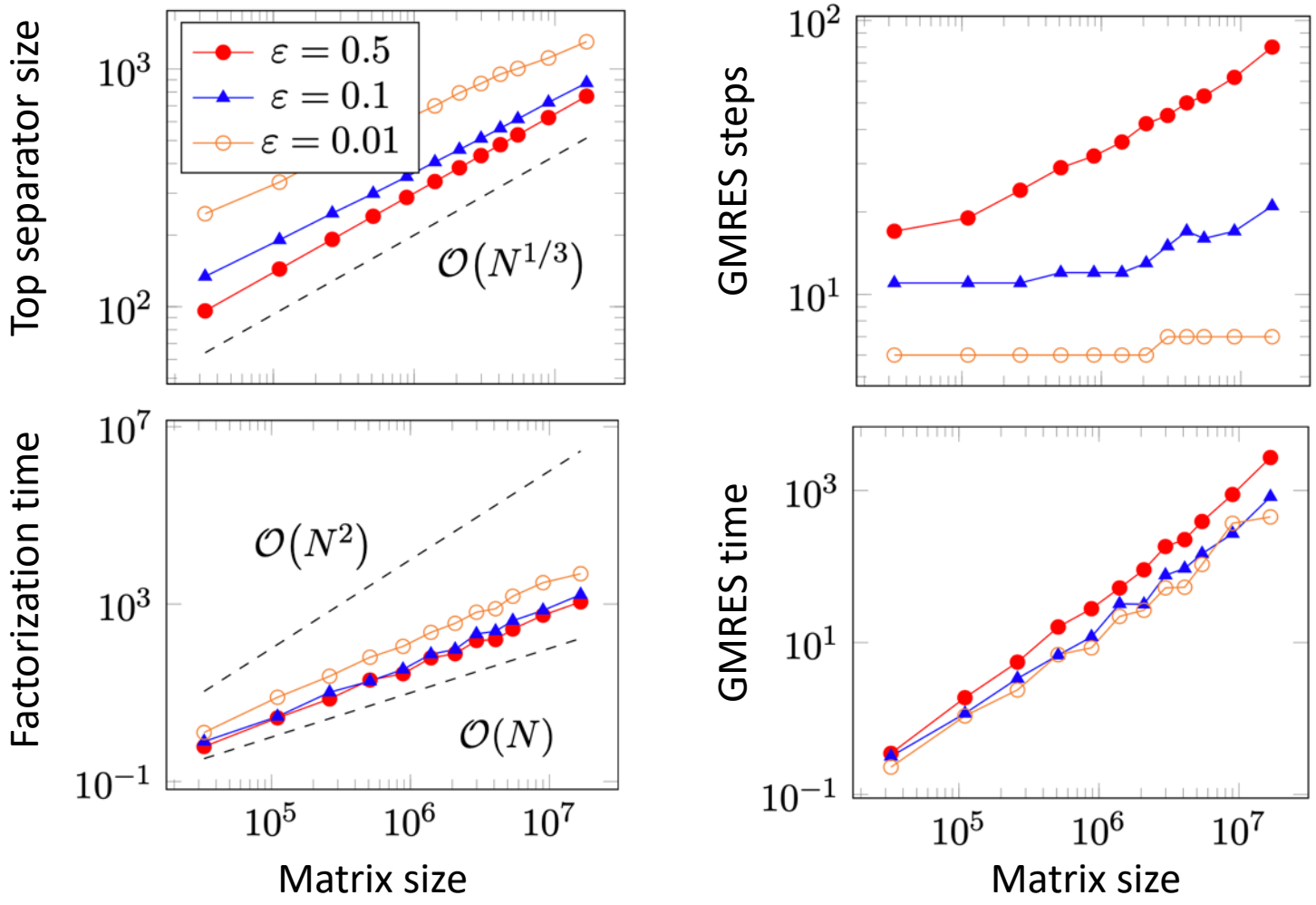


Top separator
32 GB → 6MB

Problem and picture from: Christie, Michael Andrew, and M. J. Blunt. "Tenth SPE comparative solution project: A comparison of upscaling techniques." *SPE reservoir simulation symposium*. Society of Petroleum Engineers, 2001. Thanks to Bazyli Klockiewicz for the matrix

spaND also works on unsymmetric problems

3D advection-diffusion



$-\nabla \cdot (a(x)\nabla u(x)) + b(x) \cdot \nabla u(x) = f(x)$, centered FD on $[0,1]^3$, $a = 10^{-2}, b_i = 1$

Conclusion

spaND is

- Much faster than direct methods
- Never breaks on SPD problems
- Robust and versatile

TaskTorrent

Cambier, L., Qian, Y. and Darve, E. "TaskTorrent: a Lightweight Distributed Task-Based Runtime System in C++." *To appear in Proceeding of the 2020 Parallel Application Workshop: Alternatives to MPI+X. arXiv preprint arXiv:2009.10697* (2020).

Runtime systems

Computations = tasks with dependencies

Runtime systems schedule tasks and

- avoid synchronization
- exploit all resources

```
for (k=0; k<n; k++) {
  POTRF(A[k,k]);
  for(i=k+1; i<n; i++) {
    TRSM(A[k,k],A[i,k]);
  }
  for (i=k+1; i<n; i++) {
    SYRK(A[i,k], A[i,i]);
    for (j=k+1; j<i; j++) {
        GEMM(A[i,k], A[j,k], A[i,j]);
    }
  }
}
```

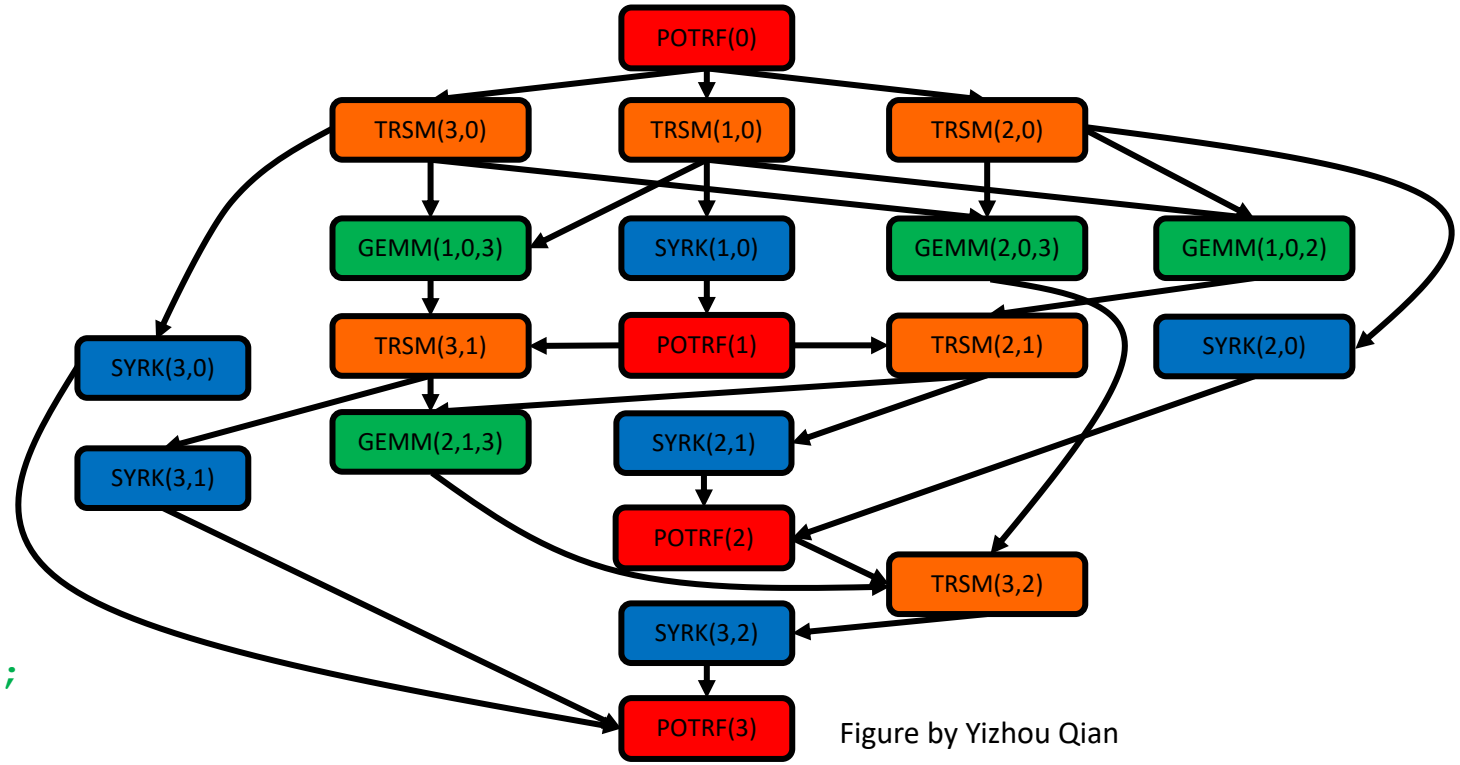


Figure by Yizhou Qian

Two approaches: STF and PTG

Sequential task flow (STF)

```
/** Define tasks **/
void task(A: in, B: out)
/** Register data **/
A = [ ... ]
B = [ ... ]
/** Process DAG **/
for (i ..., j ...)
    task(A[i], B[j])
```

- Data dependencies inferred through data sharing rules

Parametrized task graph (PTG)

```
/** Define DAG using
 *  functions of K
 **/
in_deps  = (K k) { ... }
task     = (K k) { ... }
out_deps = (K k) { ... }
/** Seed tasks **/
for (k in kinit)
    start(k)
```

- Task defined as functions over K
- Computation triggered by seeding initial tasks

Existing solutions

```
for k = 0, n do
  dpotrf(k, n, pA[f2d{i = k, j = k}])
  for i = k+1, n do
    dtrsm(i, k, n, pA[f2d{i = i, j = k}])
  end
  for i = k+1, n do
    dsyrk(i, k, n, pA[f2d{i = i, j = i}],
                   pA[f2d{i = i, j = k}])
    for (j=k+1; j<i; j++) {
      dgemm(i, j, k, n,
            pA[f2d{i = i, j = k}]),
            pA[f2d{i = i, j = k}]),
            pA[f2d{i = i, j = k}]))
    end
  end
end
```

Legion/Regent:

- Sequential semantic (STF)
- Custom language
- May need tuning for good performances
- Hard to use within legacy codes

```
TRSM(k, m)

// Execution space
k = 0   .. NT-1
m = k+1 .. NT-1

// Partitioning
: dataA(m, k)

// Flows & their dependencies
READ  A <- A POTRF(k) [type = LOWER]
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C GEMM(k-1, m, k)
        -> A SYRK(k, m)
        -> A GEMM(k, m, k+1..m-1)
        -> B GEMM(k, m+1..NT-1, m)
        -> dataA(m, k)

BODY
  trsm(A, C);
END
```

PaRSEC:

- Custom language (JDF)
- Cannot use other data structures

Goals of TaskTorrent

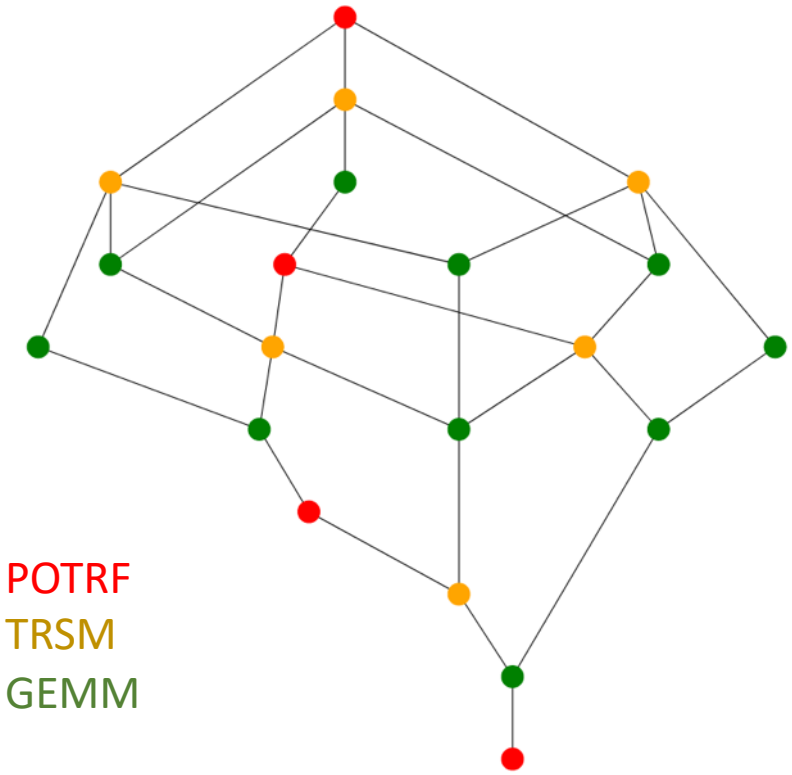
Current adoption of runtime systems is low

Wider adoption requires

- Easy to learn API
  - No new language
  - API with predictable behaviors
  - Good OOTB performances
- Good interaction with existing codebases
  - Plays well with MPI and message-passing codes
  - Incremental adoption
  - Standard tools (MPI and C++)
  - Any user data structures
  - Minimal overhead, no task refactoring

STF is easier to use but may not scale

Need to enumerate the DAG !



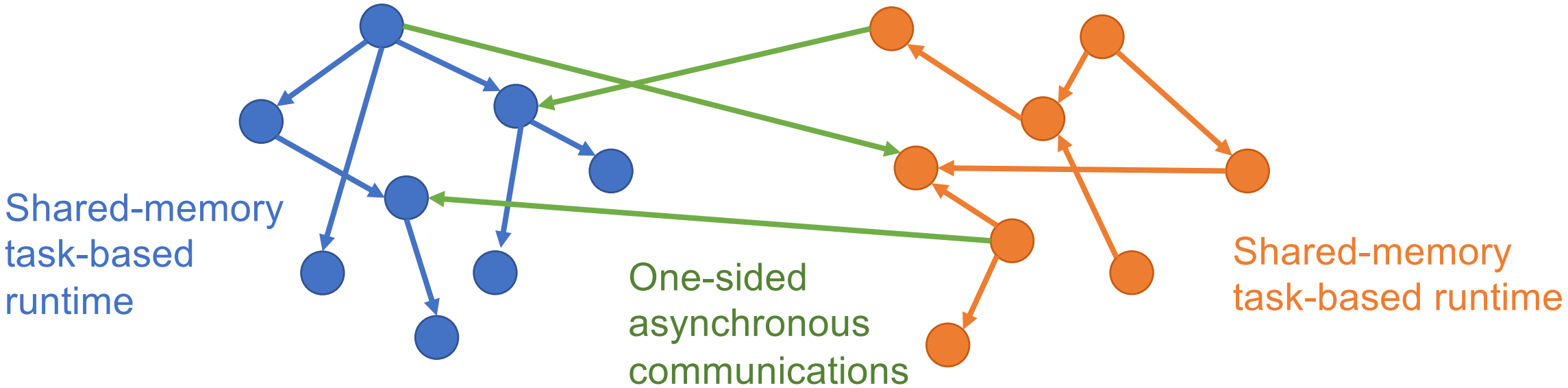
POTRF
TRSM
GEMM

DAG of 4x4 Cholesky (tiny)

DAG of 20x20 Cholesky (very small)

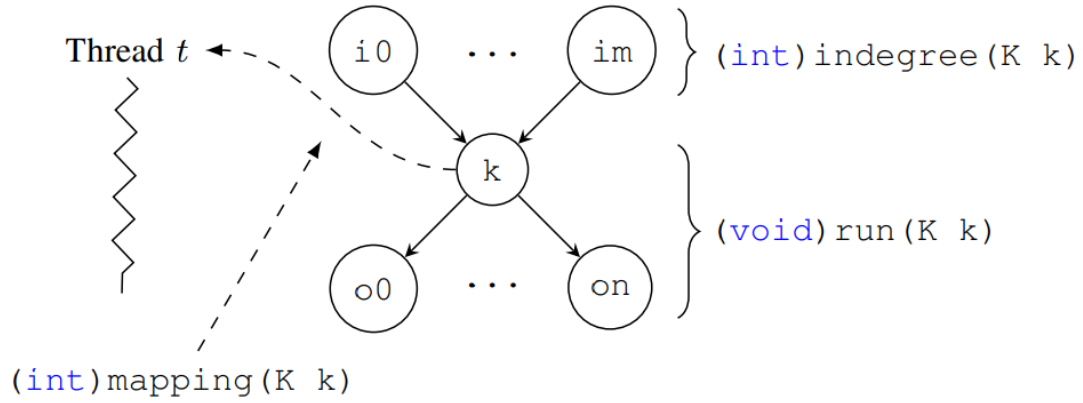
TaskTorrent combines PTG and Active messages

- Lightweight: PTG + AMs
- Message-passing, with MPI and C++ threads
- Any user data structures
- Good performances out-of-the-box



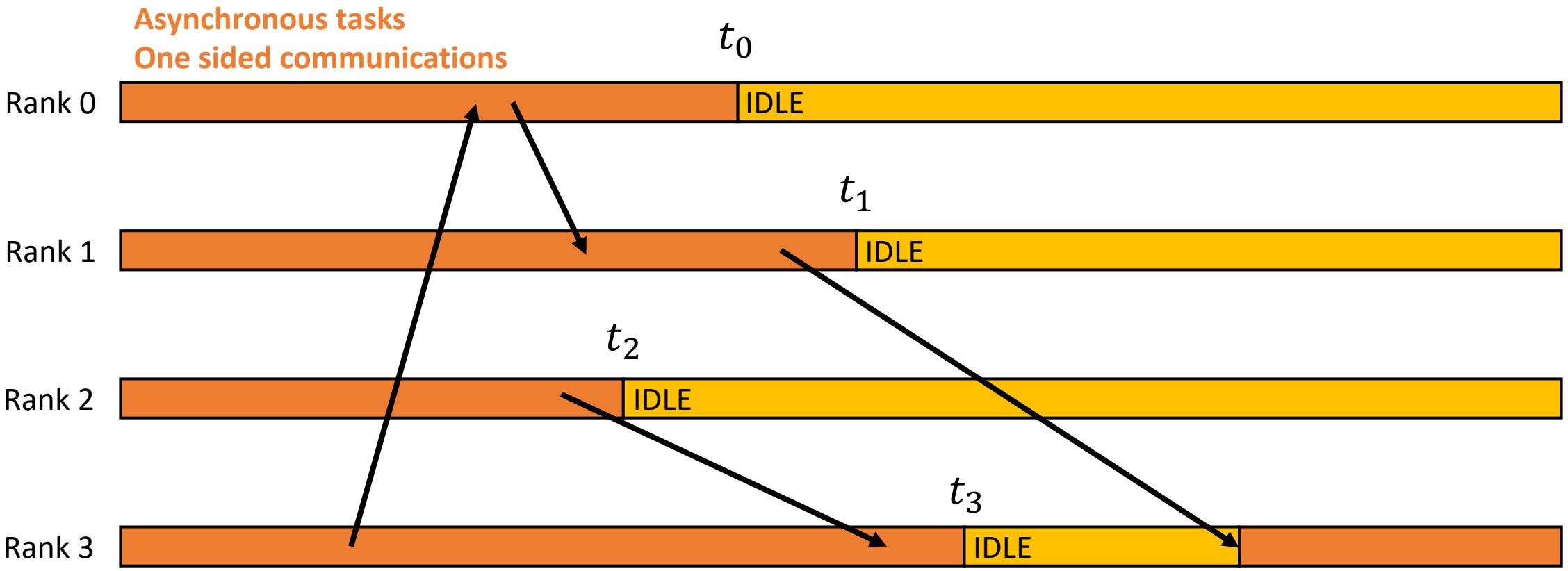
How to use TaskTorrent?

1. Tasks are `functions(K k)`
   - Mapping task to threads
   - # incoming deps
   - Computations + fulfill deps
2. Communications using AMs
   - Send data + fulfill deps
3. Seed + join

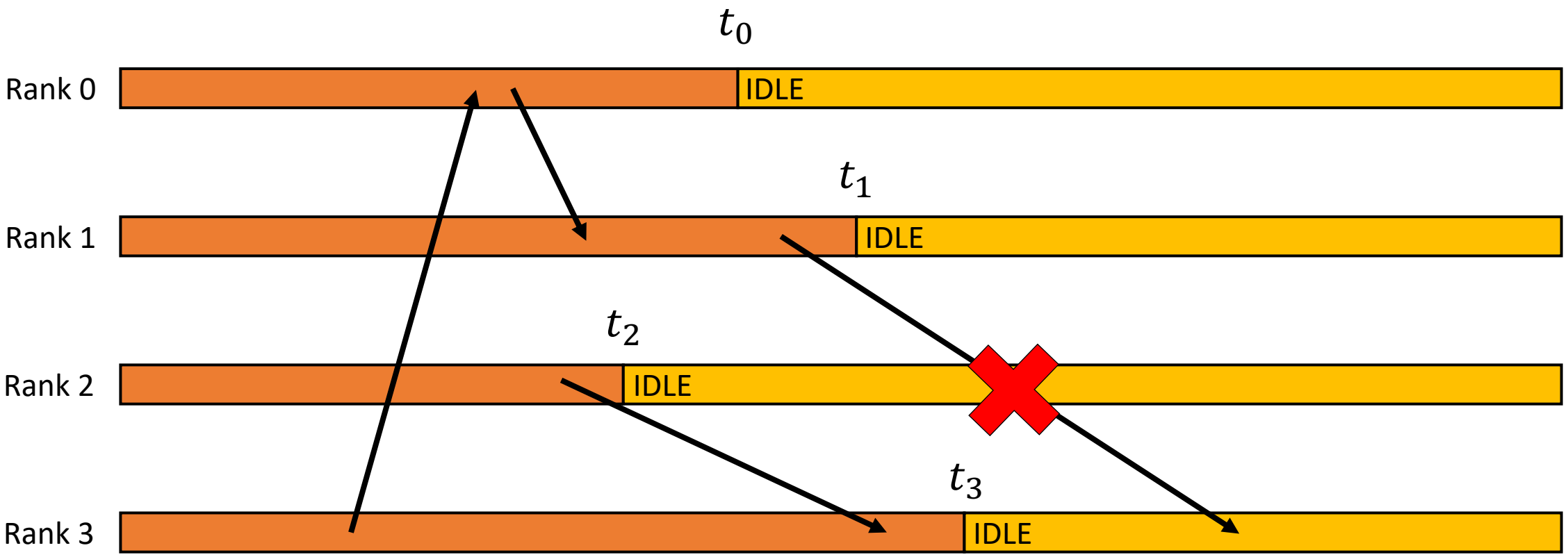



```
/** Initialize structures **/
Communicator comm(MPI_COMM_WORLD);
Threadpool tp(n_threads, &comm);
Taskflow<int> tf(&tp);
/** Create active message **/
am = comm.make_active_msg(
  [&](int d, int k, payload pk) {
    data[k] = pk;
    tf.fulfill_promise(d);
  });
/** Define Taskflow **/
tf.set_mapping(mapping)
.set_indegree(n_deps)
.set_run([&](int k) {
  compute(k);
  for (auto d : deps(k)) {
    int dest = task_2_rank(d);
    if (dest == my_rank) {
      tf.fulfill_promise(d);
    } else {
      am->send(dest, d, k, data[k])
    }
  }
});
/** Start initial tasks **/
for (auto k : initial_tasks)
  tf.fulfill_promise(k);
/** Wait for completion **/
tp.join();
```

Detecting completion is non-trivial

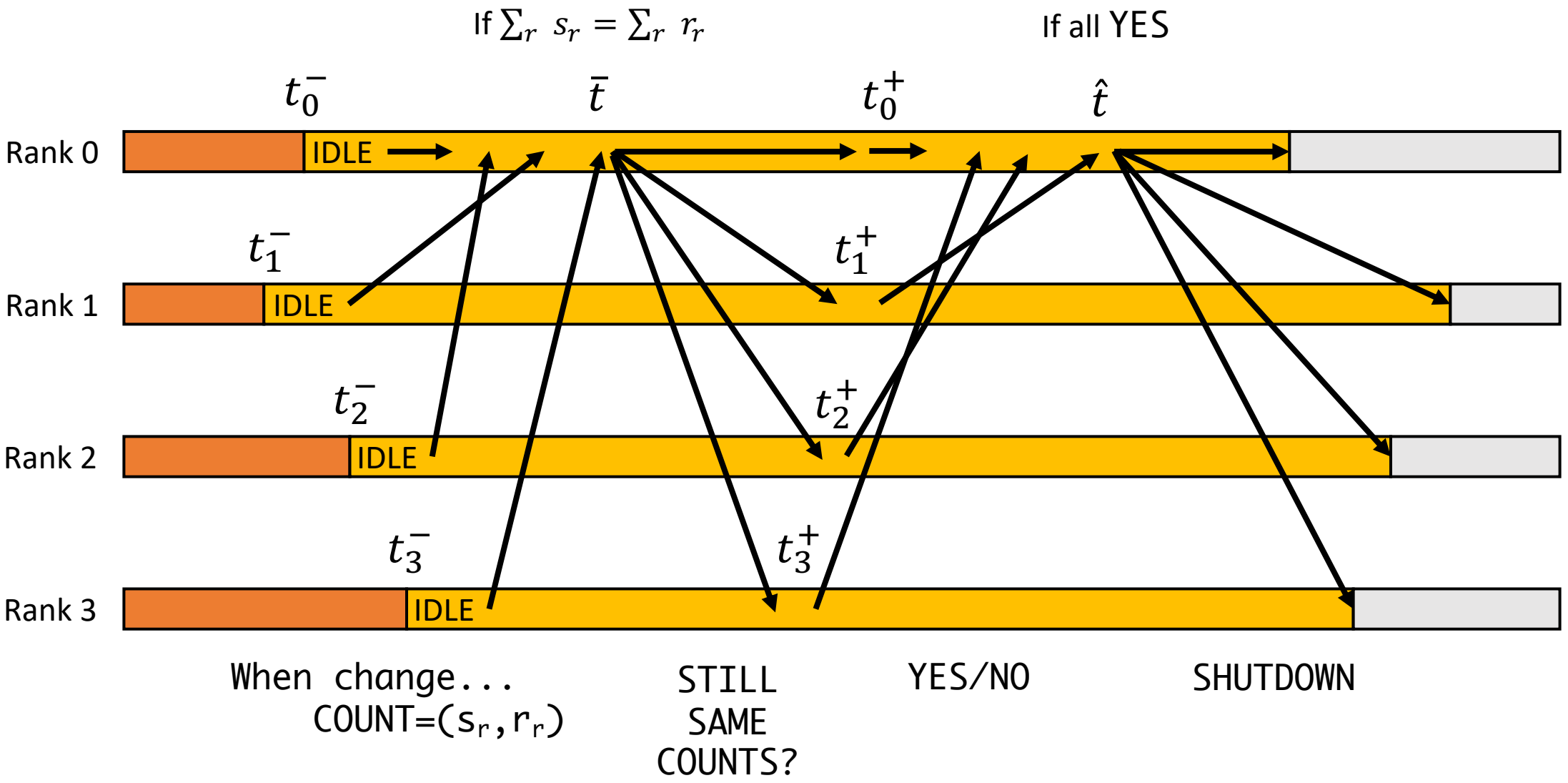


So what is “completion”?

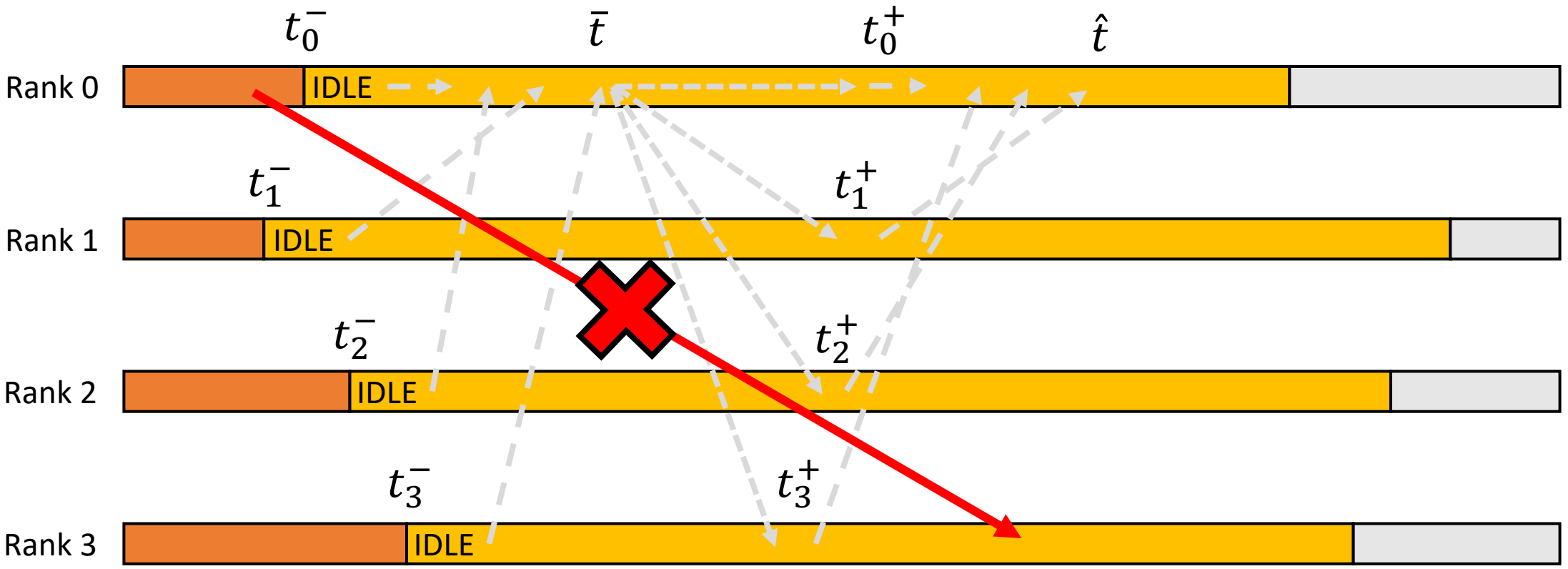


- Ranks are idle (no user tasks) at $\{t_r\}_r$
- All messages sent before $\{t_r\}_r$ are received before $\{t_r\}_r$

Detecting completion is done in two stages



No AM can cross the “envelope” ⇒ Idle forever



Properties of the completion algorithm

Theorem 1:

If the user creates a finite number of messages, the algorithm uses a finite number of messages

⇒ Finishes even if MPI is unfair

Theorem 2:

The algorithm terminates if and only if completion is reached

⇒ Correct

Benchmarks

We compare

- TaskTorrent
- StarPU (STF)
- Intel ScaLAPACK (MPI + OpenMP)

We show that

1. Runtimes outperform bulk-synchronous (ScaLAPACK)
2. TTOR (PTG) is competitive with other SOTA runtimes (StarPU, STF)
3. TTOR is easy to use
4. PTG scales better than STF when tasks are small

Distributed GEMM

Strong and weak scalings

A

0	1	0	1
2	3	2	3
0	1	0	1
2	3	2	3

B

0	1	0	1
2	3	2	3
0	1	0	1
2	3	2	3

=

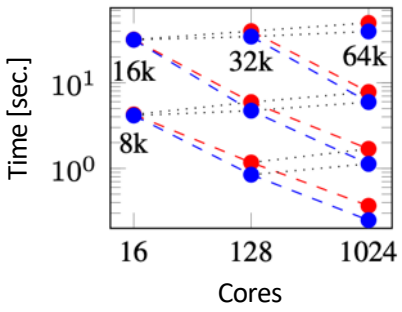
C

0	1	0	1
2	3	2	3
0	1	0	1
2	3	2	3

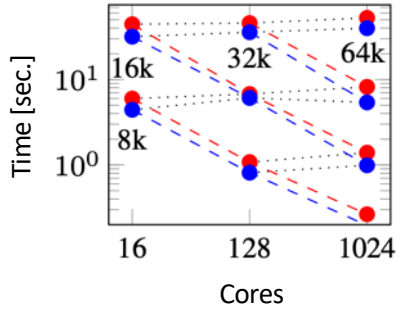
Block ↦ node mapping

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

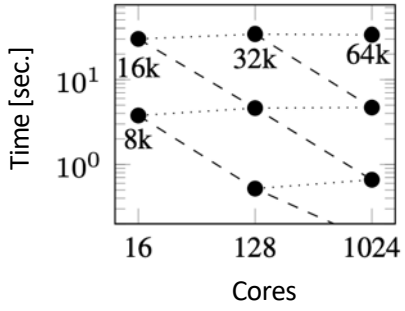
```
gemm_Cikj.set_task([&](int3 ikj){
  int i = ikj[0];
  int k = ikj[1];
  int j = ikj[2];
  C_ij[i + j * num_blocks].noalias() +=
      A_ij[i + k * num_blocks] *
      B_ij[k + j * num_blocks];
  if(k < num_blocks-1) {
    gemm_Cikj.fulfill_promise({i,k+1,j});
  }
}).set_indegree([&](int3 ikj) {
  return (ikj[1] == 0 ? 2 : 3);
}).set_mapping([&](int3 ikj) {
  return (ikj[0] / nprows + ikj[2] / npcols
      * (num_blocks / nprows)) % n_threads;
});
```



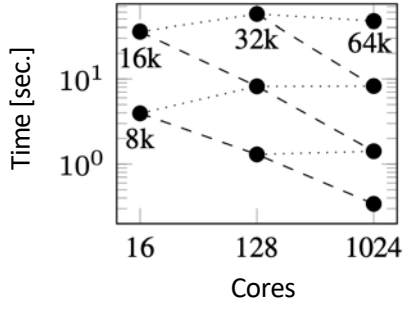
TaskTorrent 2D



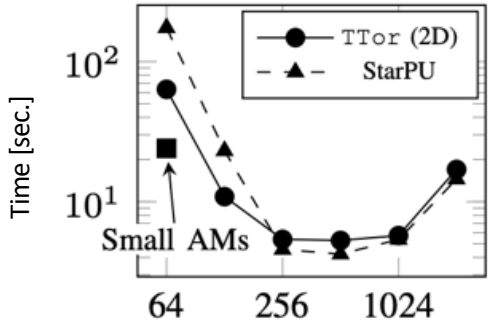
TaskTorrent 3D



StarPU 2D



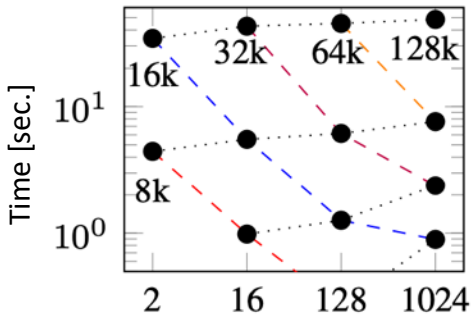
ScaLAPACK 2D



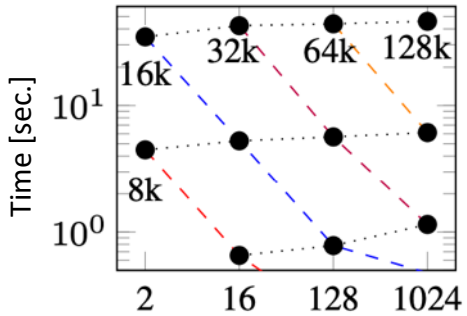
Block size impact
(N=32k, 1024 CPUs)

Distributed dense Cholesky

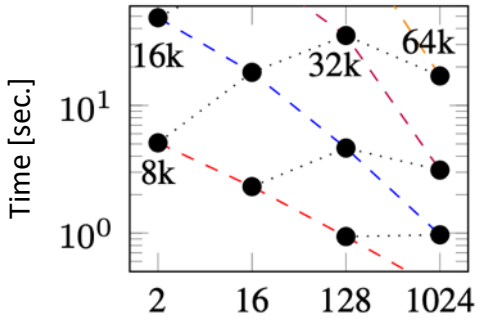
Strong and weak scalings



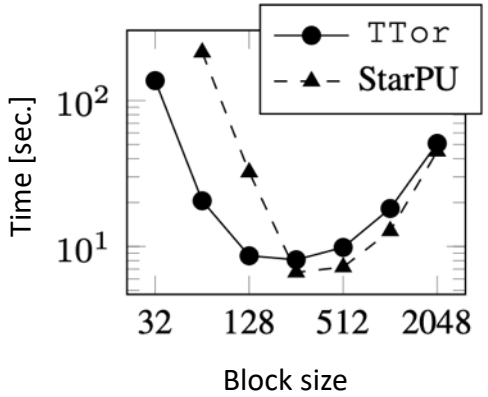
TaskTorrent



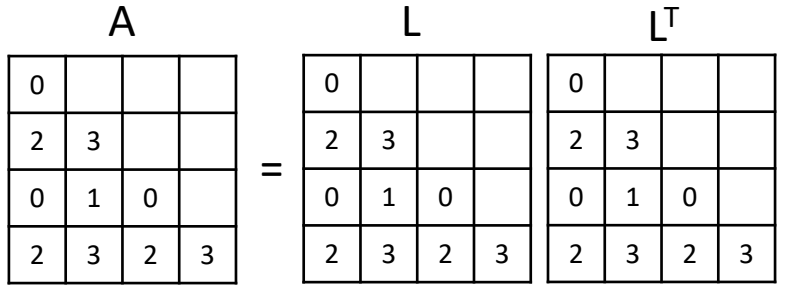
StarPU

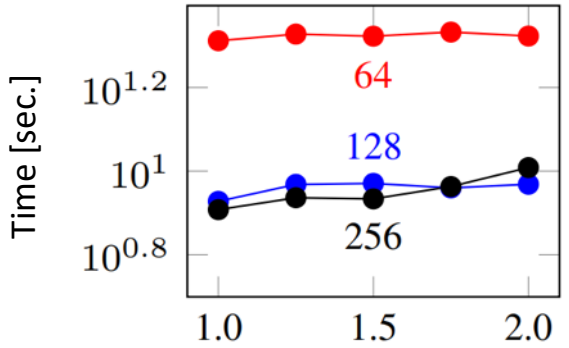


ScaLAPACK



Block size impact
(N=65k, 1024 CPUs)





Load balancing test
Various average block sizes
N=65k, 1024 CPUs

Conclusion

- TTOR combines PTG + AMs
- Its design makes it
  - Lightweight, minimal overhead, parallel DAG exploration
  - Easy to use and integrate into an existing code
  - Good out-of-the-box performances

`https://github.com/leopoldcambier/tasktorrent`

Task-based distributed spaND

With TaskTorrent

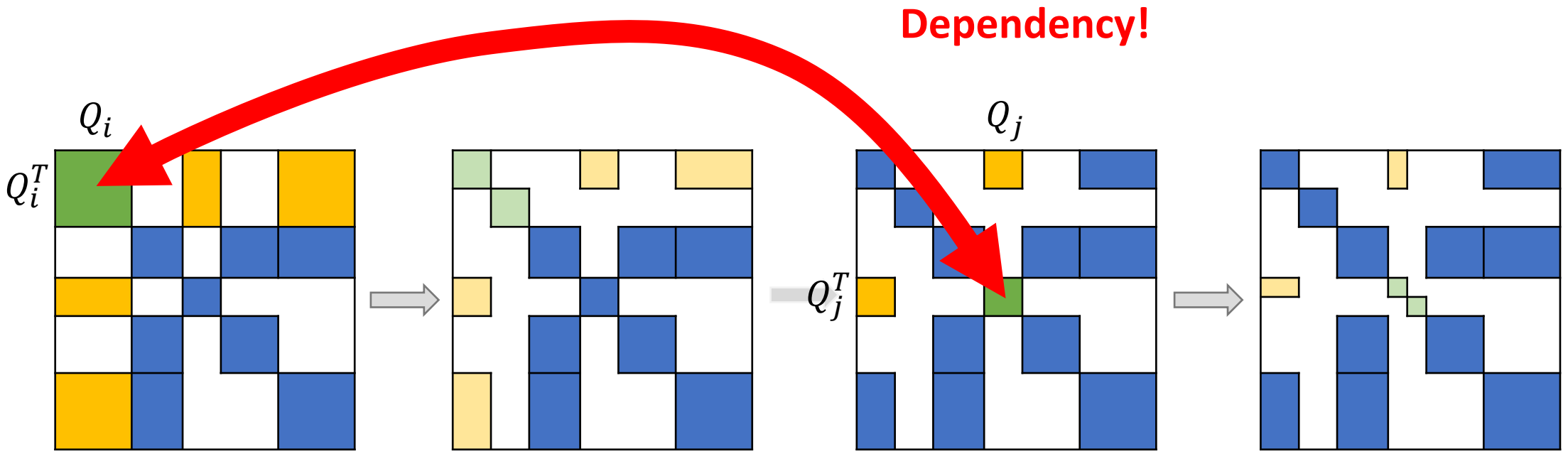
spaND sparsification is usually sequential

For level k = 1, …, L

- Eliminate interiors
- Scale interfaces
- Sparsify interfaces

  Compute Q_p (depends on A_{pn}, A_{np} for all n)

  Update $A_{nc} \leftarrow A_{np}Q_{pc}, \quad A_{pc} \leftarrow Q_{pc}^T A_{pn}$



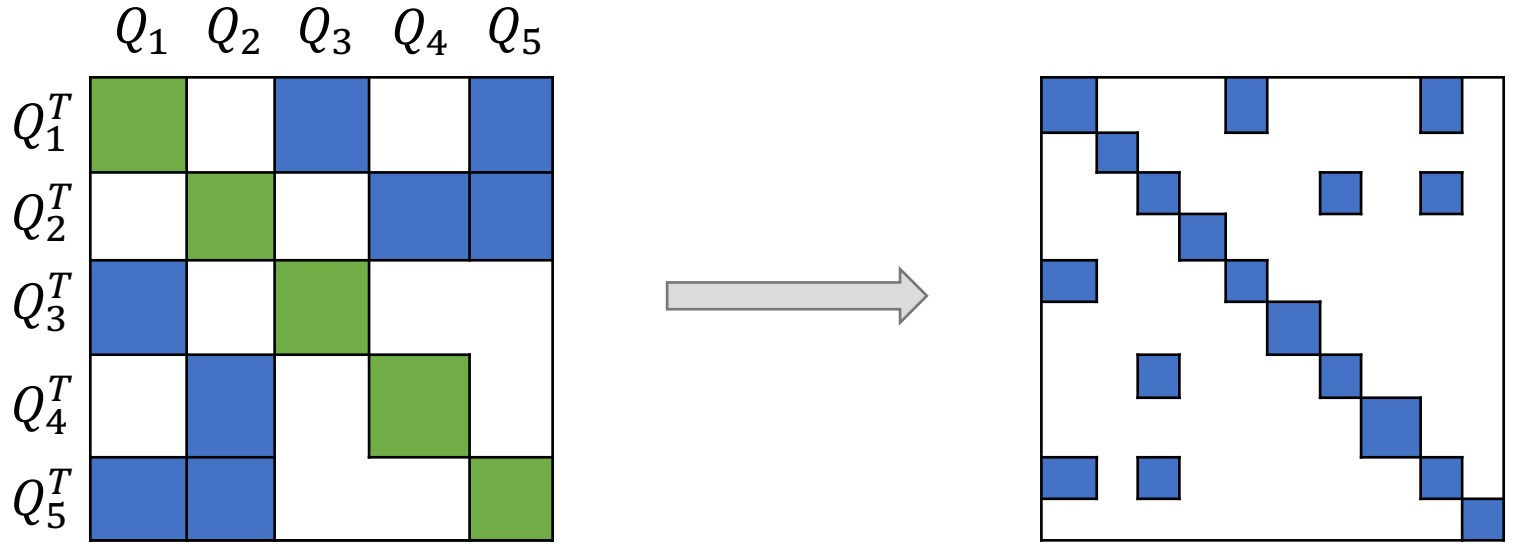
Simultaneous sparsification is more concurrent…

For level k = 1, …, L
- Eliminate interiors
- Scale interfaces
- Sparsify interfaces

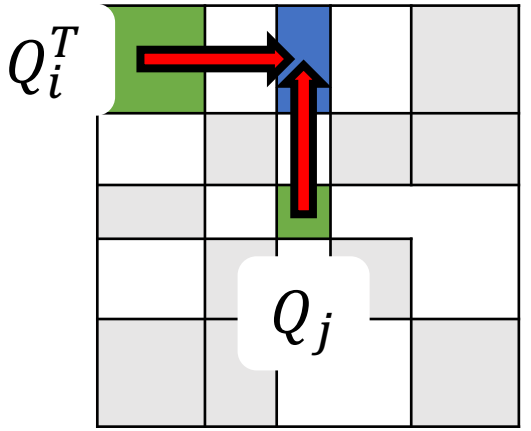
  For all p, compute Q_p (w/ original A_{pn}, A_{np})

  Update $A_{ij}^{+} = Q_{ic}^{T} A_{ij} Q_{jc}$

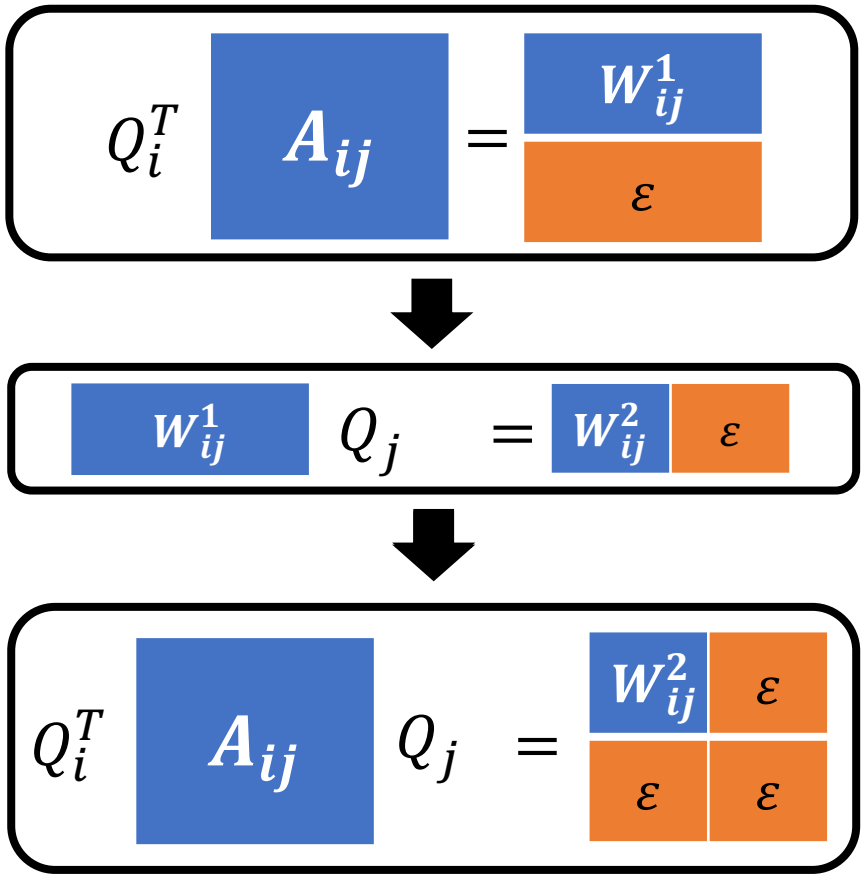
- More concurrent
- More flops
- Less accurate ?



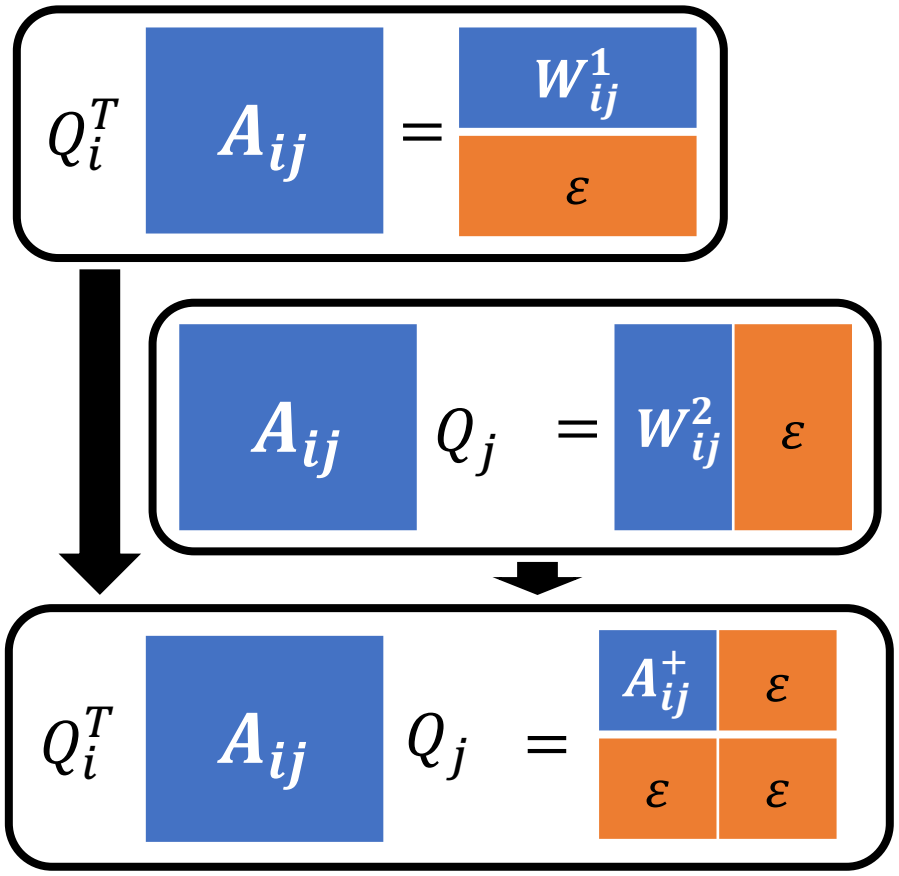
… and as accurate



Sequential sparsification



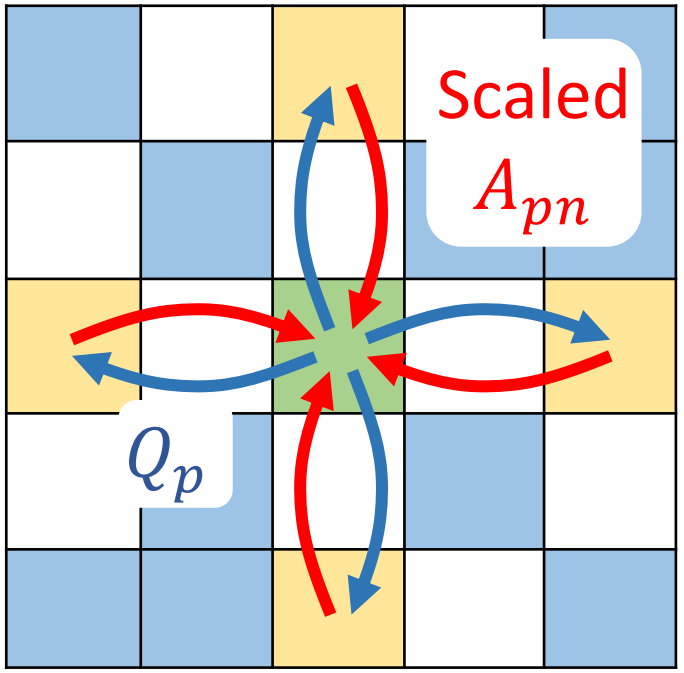
Simultaneous sparsification



How to express the DAG?

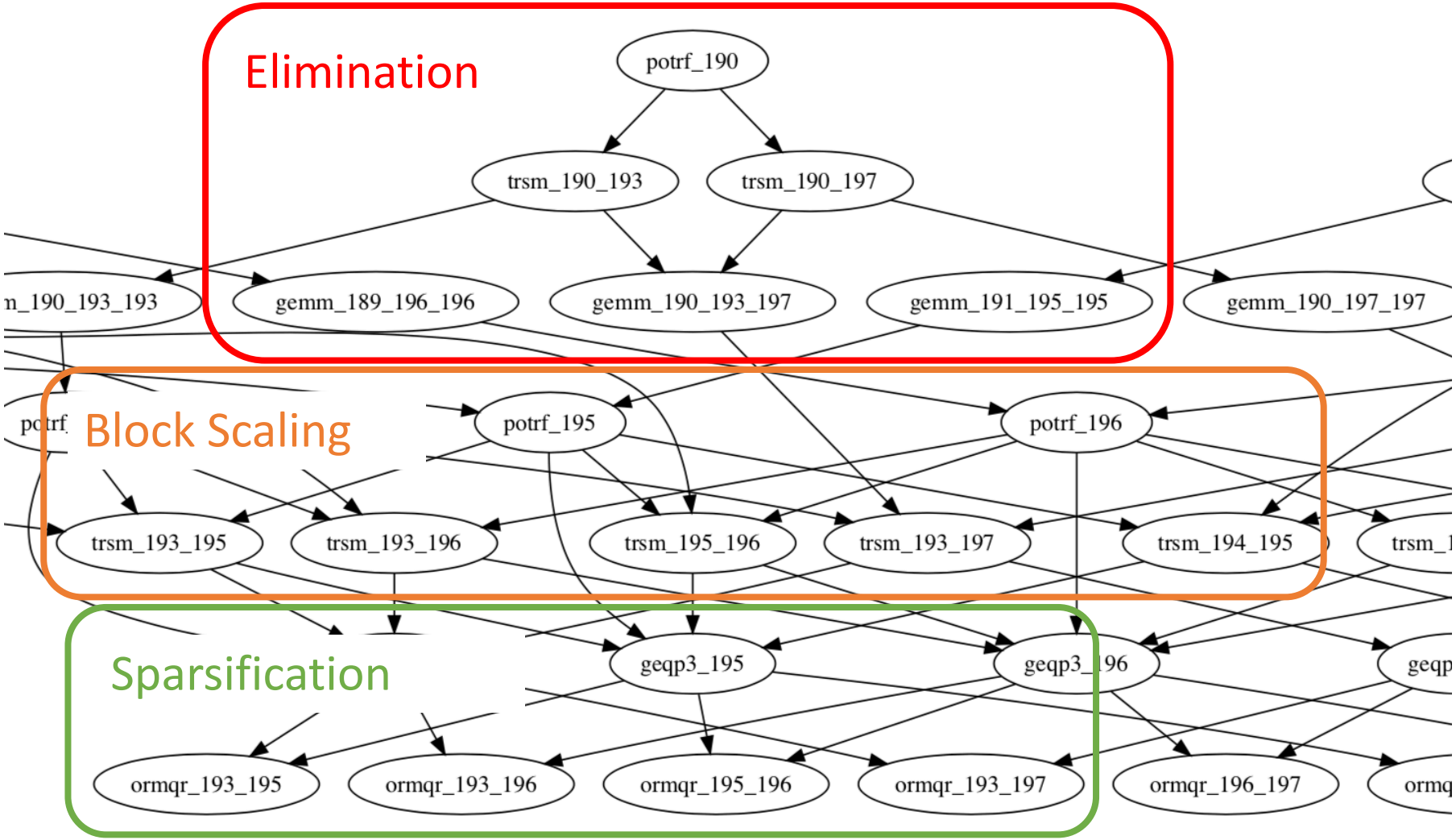
- 1 block operation ⇔ 1 task
- Granularity = interfaces
- Level-per-level DAGs

Rank revealing QR...



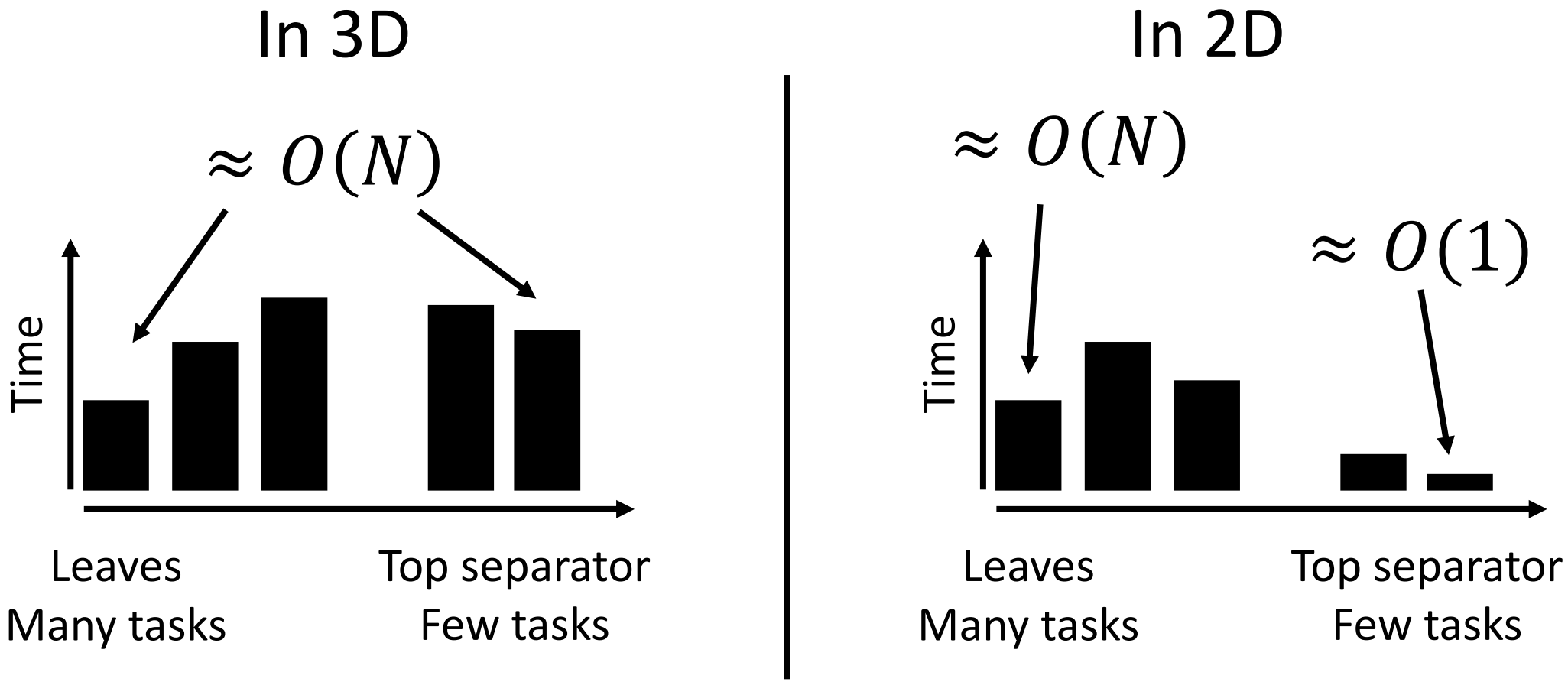
- Inputs: scaled (A_{pn}, A_{np}) from block scaling
- Task: RRQR, $Q_p = \mathrm{RRQR}(\begin{bmatrix} A_{pn} & A_{np}^T \end{bmatrix})$
- Output: sends Q_p, triggers compressions $Q_p^T A_{pq} Q_q$ and $Q_q^T A_{qp} Q_p$

Per-level DAG is wide

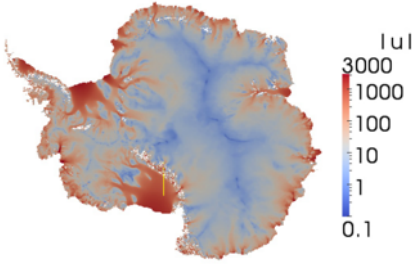


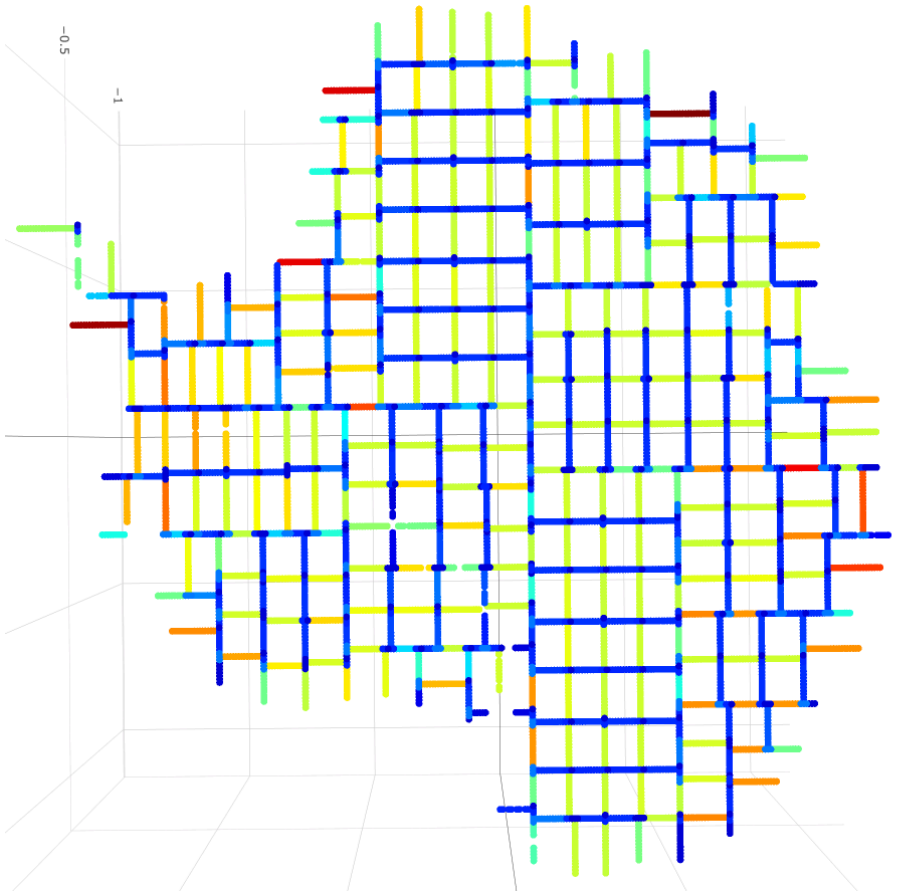
Still some limitations in 3D

Since Tasks ⇔ Block operation between interfaces

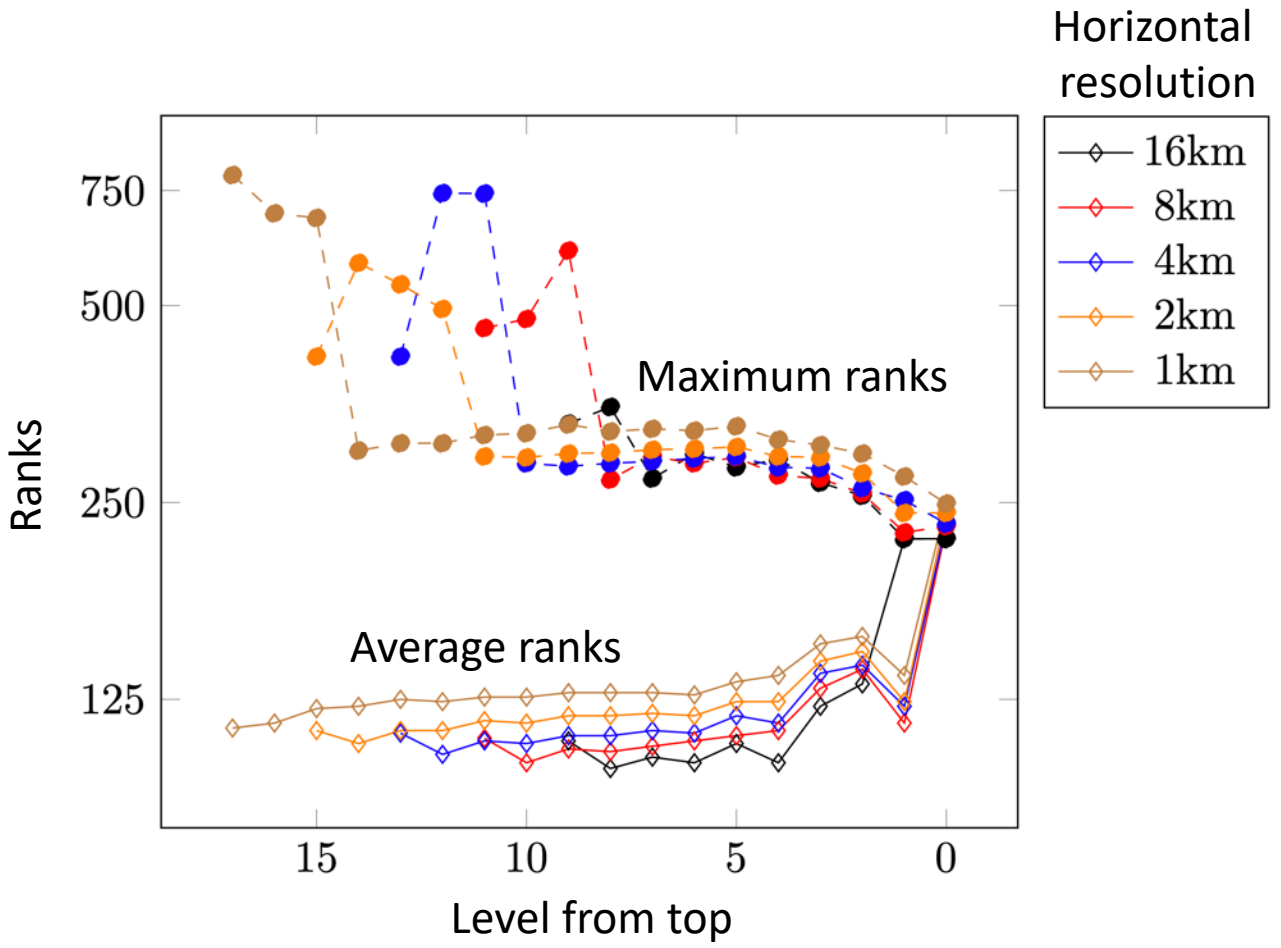


Ice-Sheet
Ranks are (fairly) uniform throughout the domain





Minimum = 22, Average = 98, Maximum = 350



Ice-Sheet: good weak scalings

cores	N	spaND				AMG (Hypre)	
		t_{fact}	t_{app}	n_{CG}	t_{total}	n_{CG}	t_{total}
36	1M	6.2	0.14	6	7.1	427	15
144	4M	7.3	0.15	6	8.5	456	16
576	18M	8.9	0.15	7	10.5	527	22
2304	74M	9.8	0.17	8	11.7	627	29
9216	296M	13.2	0.21	12	16.9	623	39

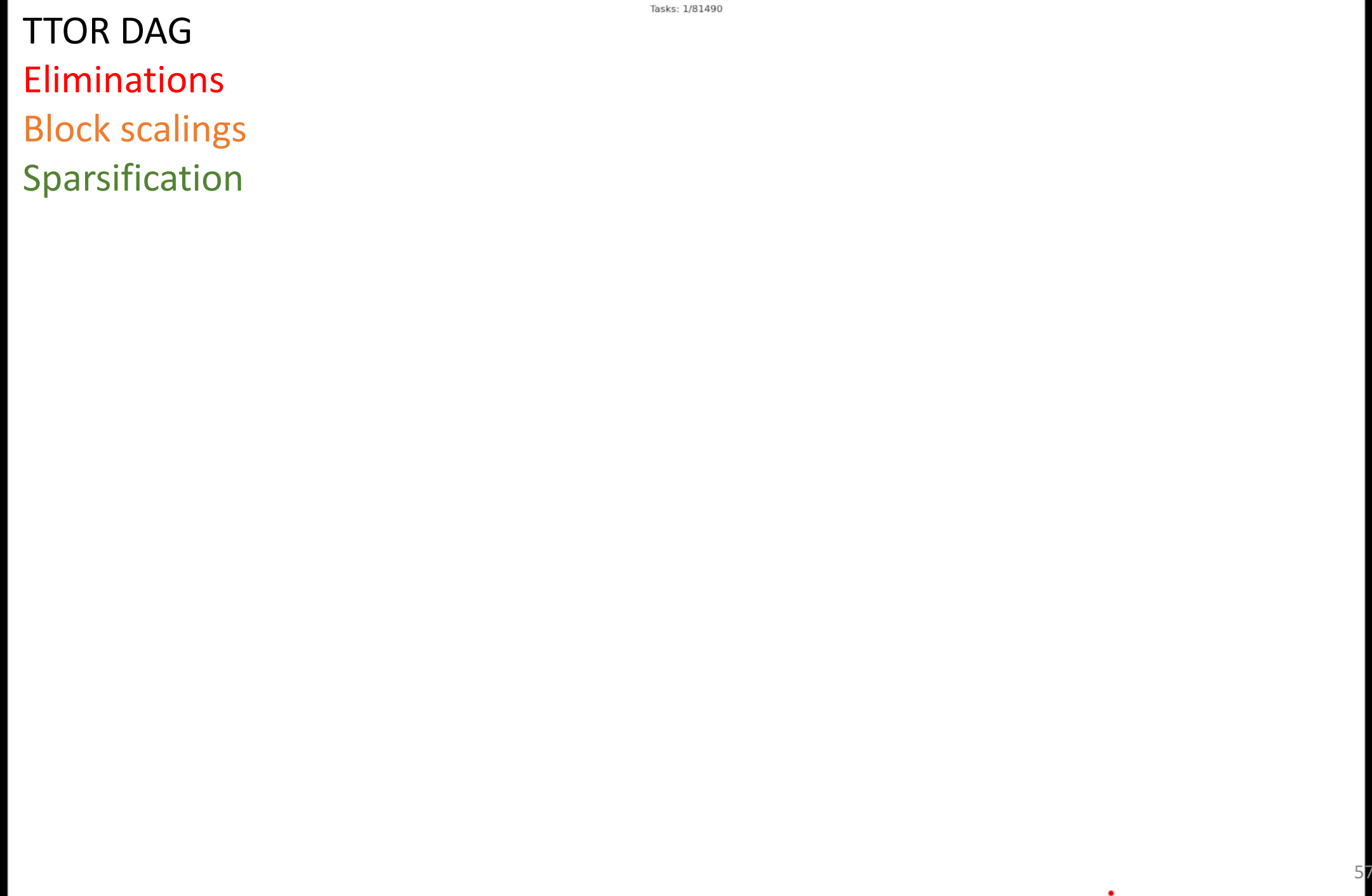
Weak scalings. 36 cores (N=1M) to 9216 cores (N=296M).
$\varepsilon = 10^{-2}$. CG stops at 10^{-8}. Hypre using Boomer AMG. On Quartz (LLNL)

TTOR DAG

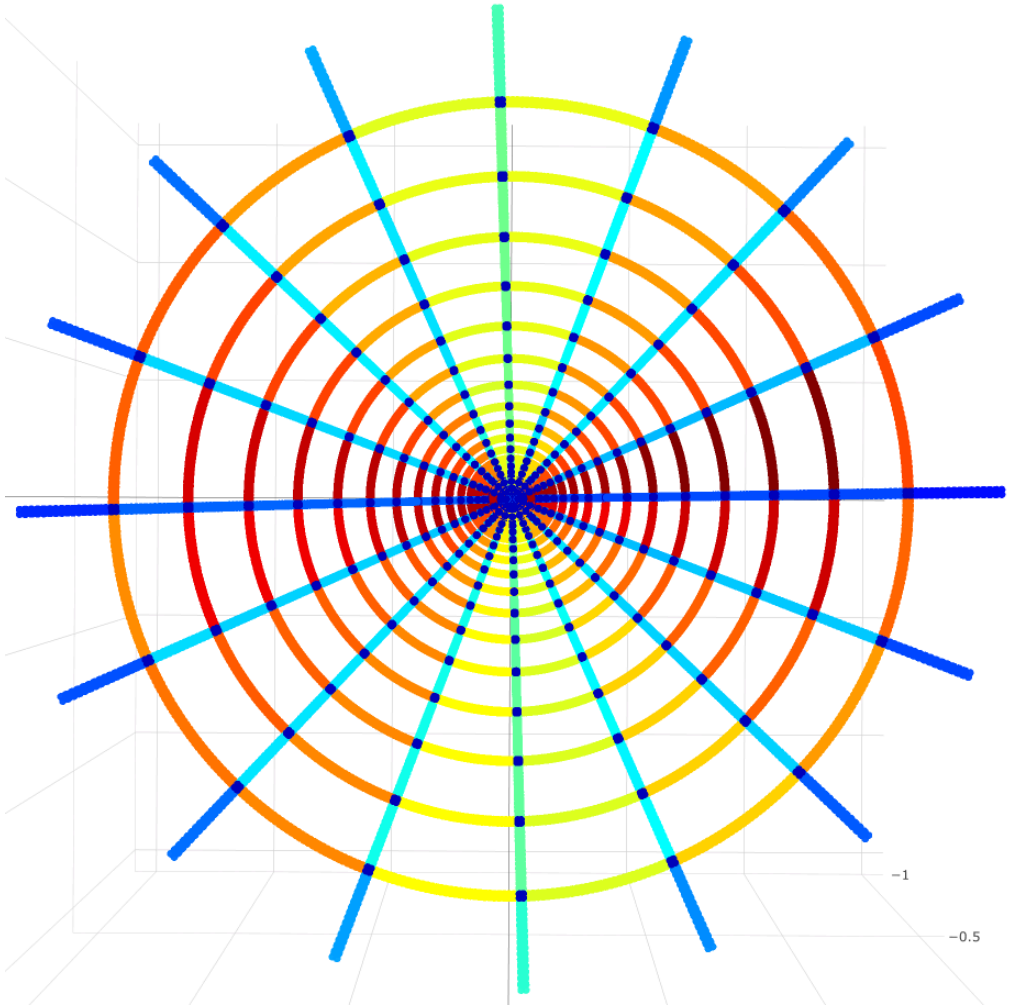
Eliminations

Block scalings

Sparsification

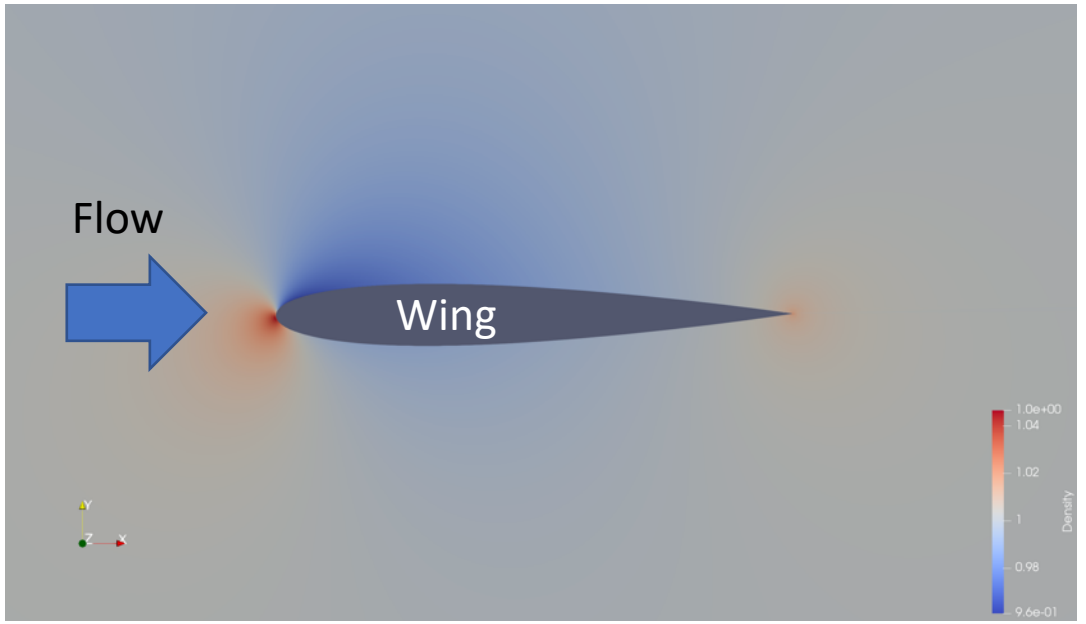


spaND in SU2: the NACA airfoil



Ranks from 40 to 296
Strong directionality in the ranks

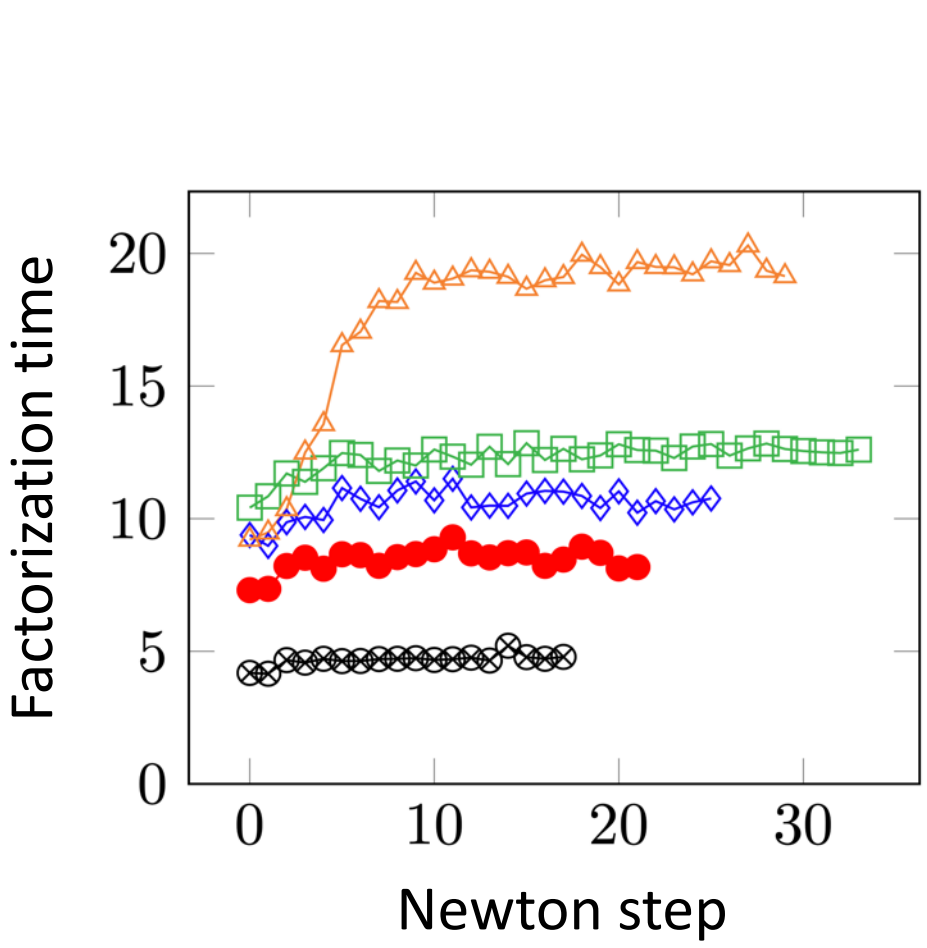
Inviscid flow around a wing
Euler equations

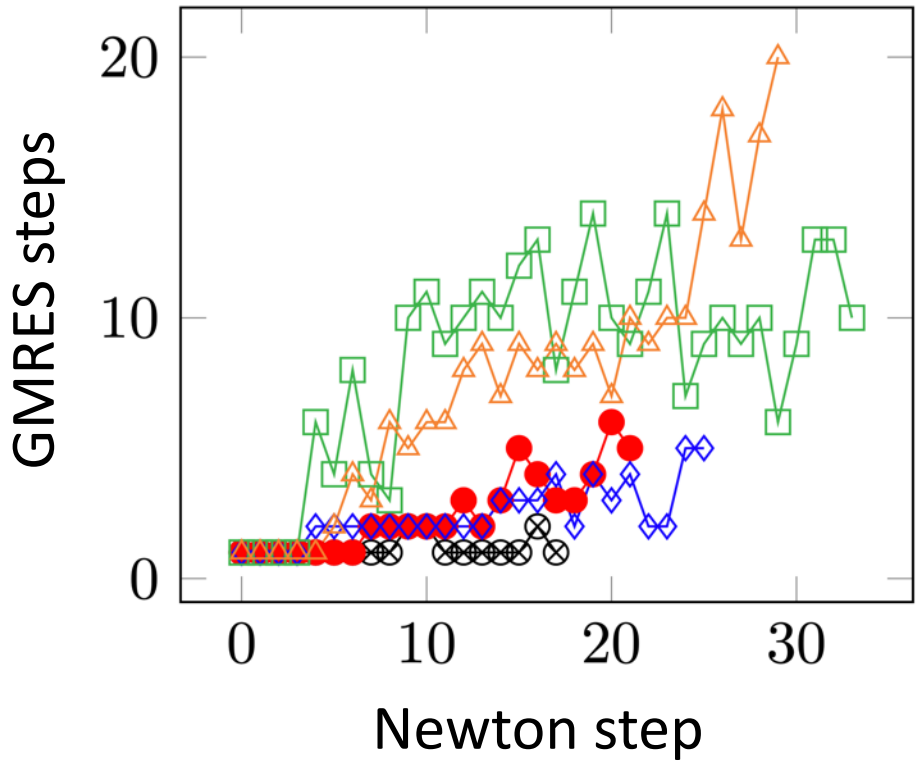


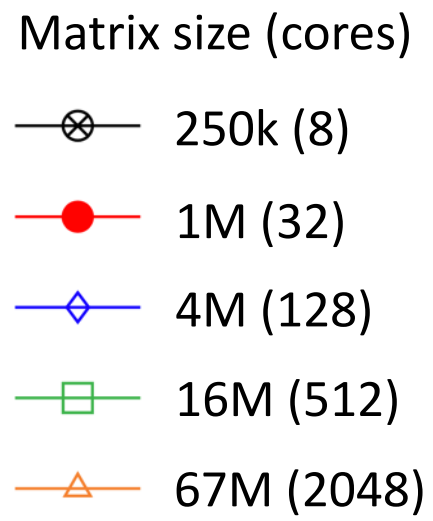
Density

SU2: Economon, T. D., Palacios, F., Copeland, S. R., Lukaczyk, T. W., & Alonso, J. J. (2016). SU2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, *54*(3), 828-846.
Thanks to Zan Xu and Juan Alonso

Some difficulties with irregular ranks







Weak scalings. 8 cores (N = 250k) to 2048 cores (N = 67M).
$\varepsilon = 10^{-3}$. GMRES stops at 10^{-3}. Newton stops at 10^{-14}. On Armstrong (Stanford HPCC)

Conclusion and perspectives

Hierarchical solvers are versatile

spaND

- Can be used on many matrices
- For most problems, complexity is about $O(N \log N)$
- Guaranteed to work on SPD

- More studies on unsymmetric problems are needed
- Full, general 3D task-based parallel require distributed RRQRs or other sparsification approach

Task-based runtimes *have* to be easy to use

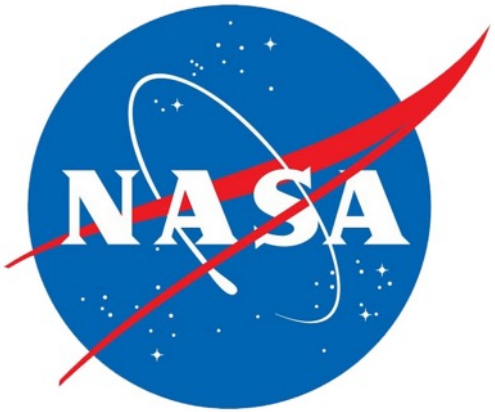
- Runtime systems need to be easy to use for adoption
- A simple approach like TTOR scales well in practice
- It is easy to combine with block algorithms such as spaND and leads to excellent performances

References

- spaND
  - HIF: Ho, Kenneth L., and Lexing Ying. "Hierarchical interpolative factorization for elliptic operators: differential equations." *Communications on Pure and Applied Mathematics* 69.8 (2016): 1415-1451.
  - spaND: Cambier, Léopold, et al. "An algebraic sparsified nested dissection algorithm using low-rank approximations." *SIAM Journal on Matrix Analysis and Applications* 41.2 (2020): 715-746.
  - 2nd order spaND: Klockiewicz, Bazyli, et al. "Second Order Accurate Hierarchical Approximate Factorization of Sparse SPD Matrices." *arXiv preprint arXiv:2007.00789* (2020).
  - Suitesparse: Davis, Timothy A., and Yifan Hu. "The University of Florida sparse matrix collection." *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011): 1-25.
  - Ice-sheet: Tezaur, Irina K., et al. "Albany/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis." *Geoscientific Model Development (Online)* 8.4 (2015).
  - SPE: Christie, Michael Andrew, and M. J. Blunt. "Tenth SPE comparative solution project: A comparison of upscaling techniques." *SPE reservoir simulation symposium*. Society of Petroleum Engineers, 2001.
  - SU2: Economon, Thomas D., et al. "SU2: An open-source suite for multiphysics simulation and design." *Aiaa Journal* 54.3 (2016): 828-846.
- TaskTorrent & runtimes
  - StarPU: Augonnet, Cédric, et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures." *Concurrency and Computation: Practice and Experience* 23.2 (2011): 187-198
  - Legion: Bauer, Michael, et al. "Legion: Expressing locality and independence with logical regions." *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012.
  - Regent: Slaughter, Elliott, et al. "Regent: a high-productivity programming language for HPC with logical regions." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015.
  - PaRSEC: Bosilca, George, et al. "Parsec: Exploiting heterogeneity to enhance scalability." *Computing in Science & Engineering* 15.6 (2013): 36-45.
  - TaskTorrent: Cambier, Léopold, Yizhou Qian, and Eric Darve. "TaskTorrent: a Lightweight Distributed Task-Based Runtime System in C++." *arXiv preprint arXiv:2009.10697* (2020).

Funding

- Fellowship from Total S.A.
- Research grants from
  - DOE (E-NA0002373-1)
  - Sandia National Lab (LDRD)
  - NASA (80NSSC18M0152)



Thank you for attending!

Now is a great time to ask questions