AllReduce Algorithms

Or Tall Skinny Algorithms if you are from Berkeley

AllReduce Algorithms

1) Tall Skinny matrices: Application
2) The CholeskyQR algorithm (see MATH6664)
3) AllReduce Householder factorization
4) Application to dense LU and dense QR factorizations

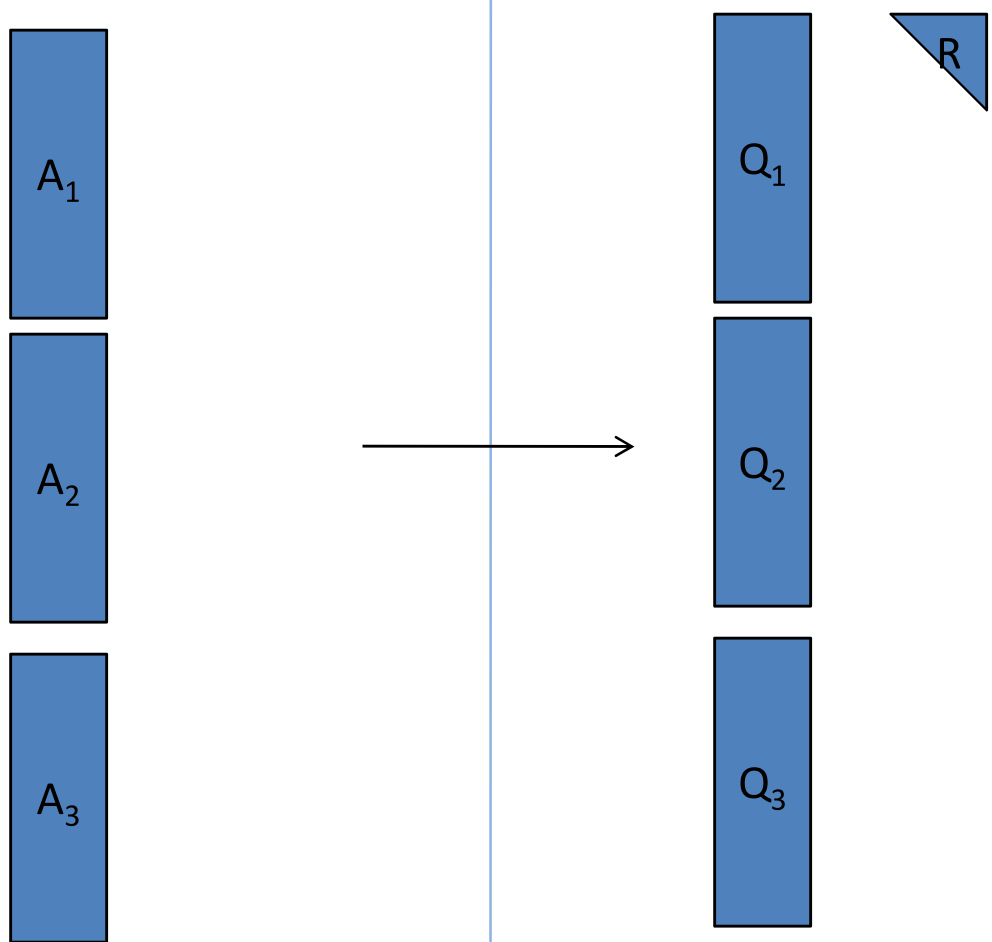
AllReduce Algorithms

1) **Tall Skinny matrices: Application**
2) The CholeskyQR algorithm (see MATH6664)
3) AllReduce Householder factorization
4) Application to dense LU and dense QR factorizations

Reduce Algorithms: Introduction

The QR factorization of a long and skinny matrix with its data partitioned vertically across several processors arises in a wide range of applications.

Input:	**Output:**
A is block distributed by rows	Q is block distributed by rows R is global



Example of applications: in block iterative methods.

a) **in iterative methods with multiple right-hand sides** (block iterative methods:)

   1) Trilinos (Sandia National Lab.) through Belos (R. Lehoucq, H. Thornquist, U. Hetmaniuk).
   2) BlockGMRES, BlockGCR, BlockCG, BlockQMR, …

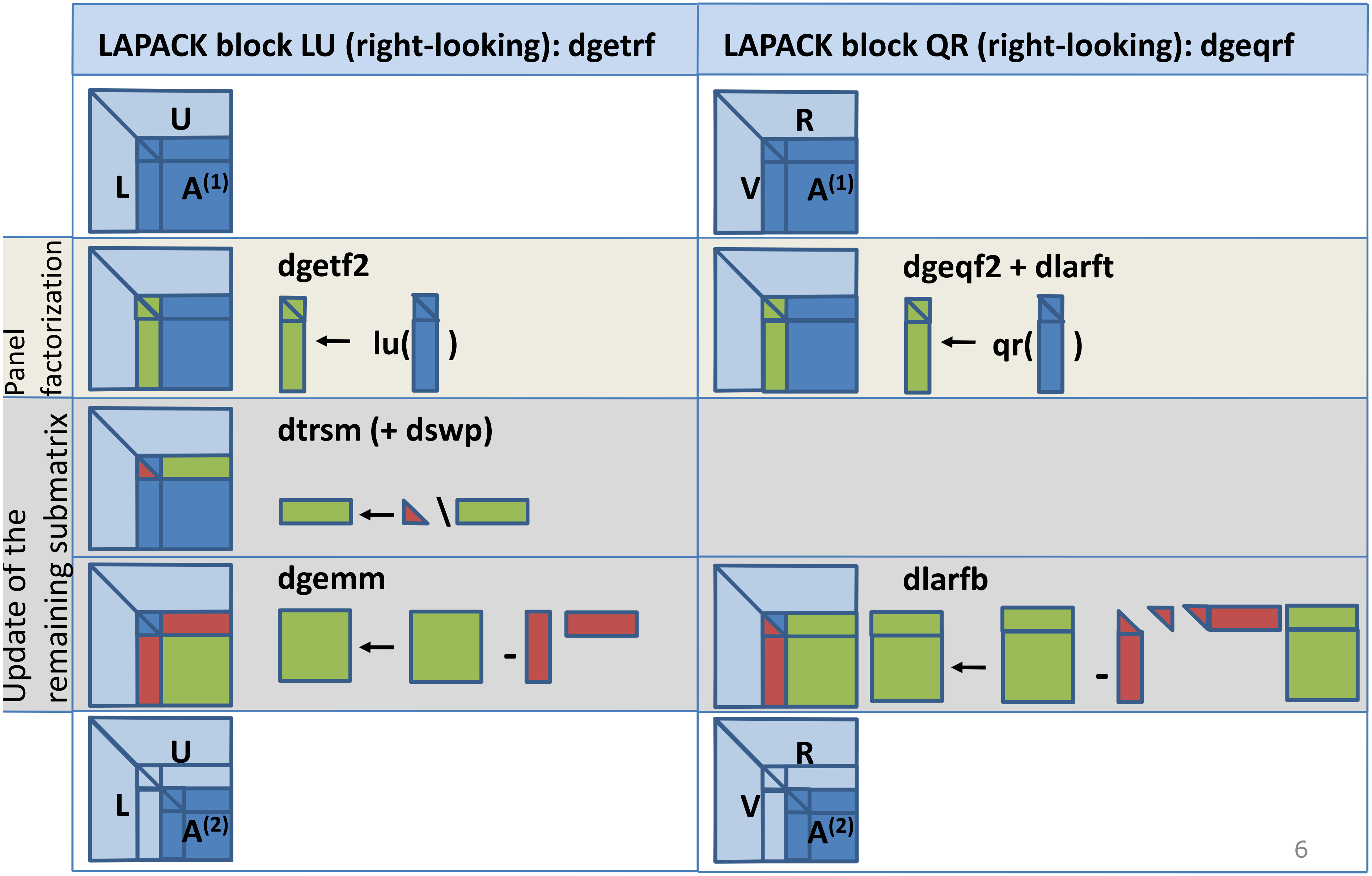
b) **in iterative methods with a single right-hand side**

   1) s-step methods for linear systems of equations (e.g. A. Chronopoulos),
   2) LGMRES (Jessup, Baker, Dennis, U. Colorado at Boulder) implemented in PETSc,
   3) Recent work from M. Hoemmen and J. Demmel (U. California at Berkeley).

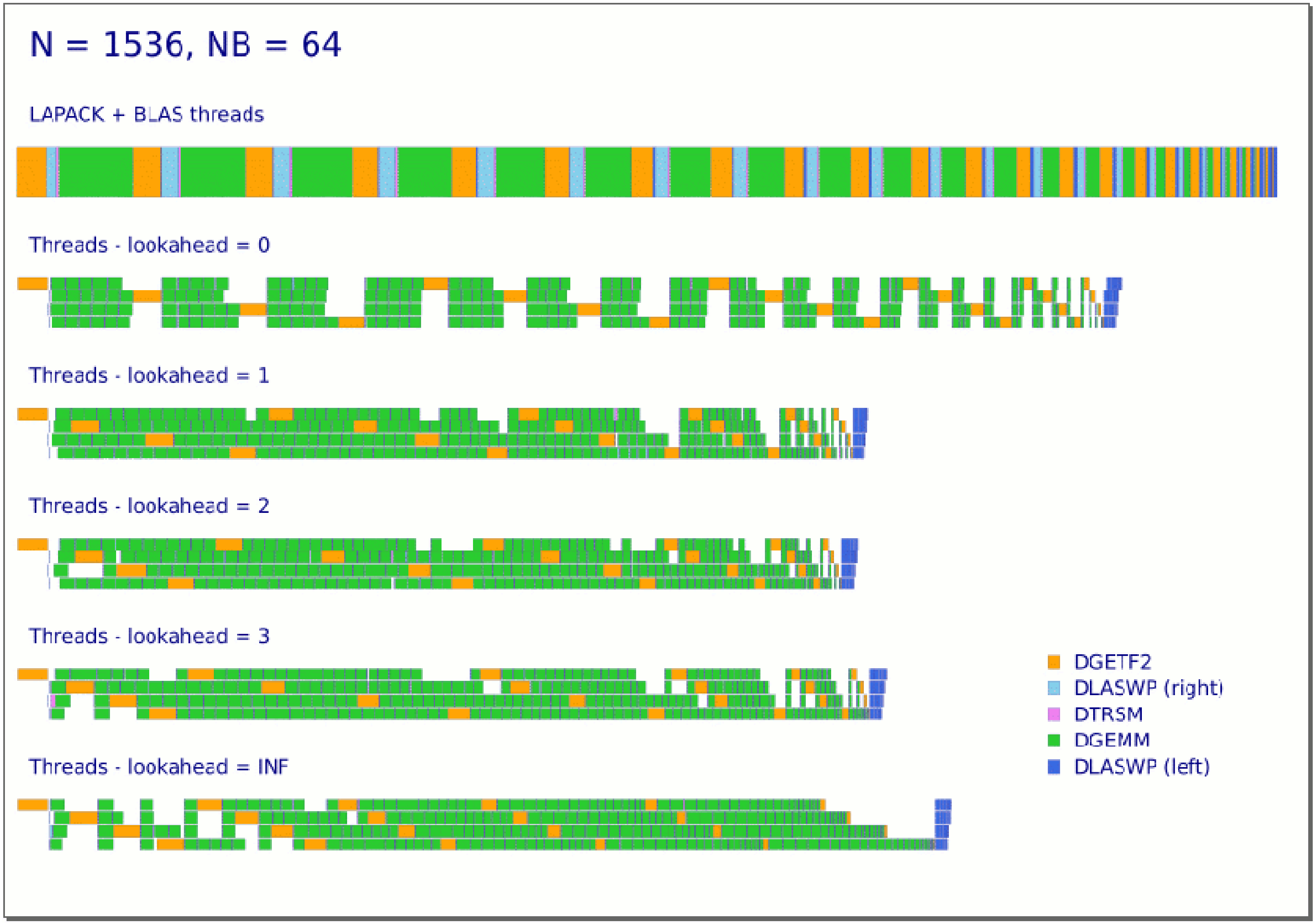
c) **in iterative eigenvalue solvers,**

   1) PETSc (Argonne National Lab.) through BLOPEX (A. Knyazev, UCDHSC),
   2) HYPRE (Lawrence Livermore National Lab.) through BLOPEX,
   3) Trilinos (Sandia National Lab.) through Anasazi (R. Lehoucq, H. Thornquist, U. Hetmaniuk),
   4) PRIMME (A. Stathopoulos, Coll. William & Mary),
   5) And also TRLAN, BLZPACK, IRBLEIGS.

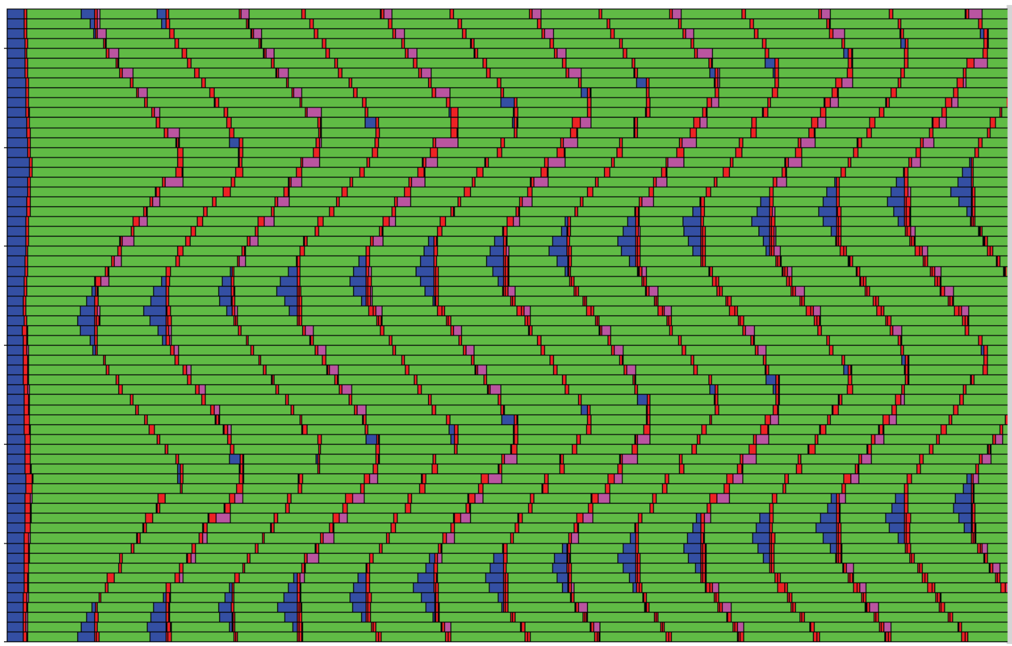
Example of applications:
panel factorization of dense blocked factorization



Example with PLASMA



Example with ScaLAPACK



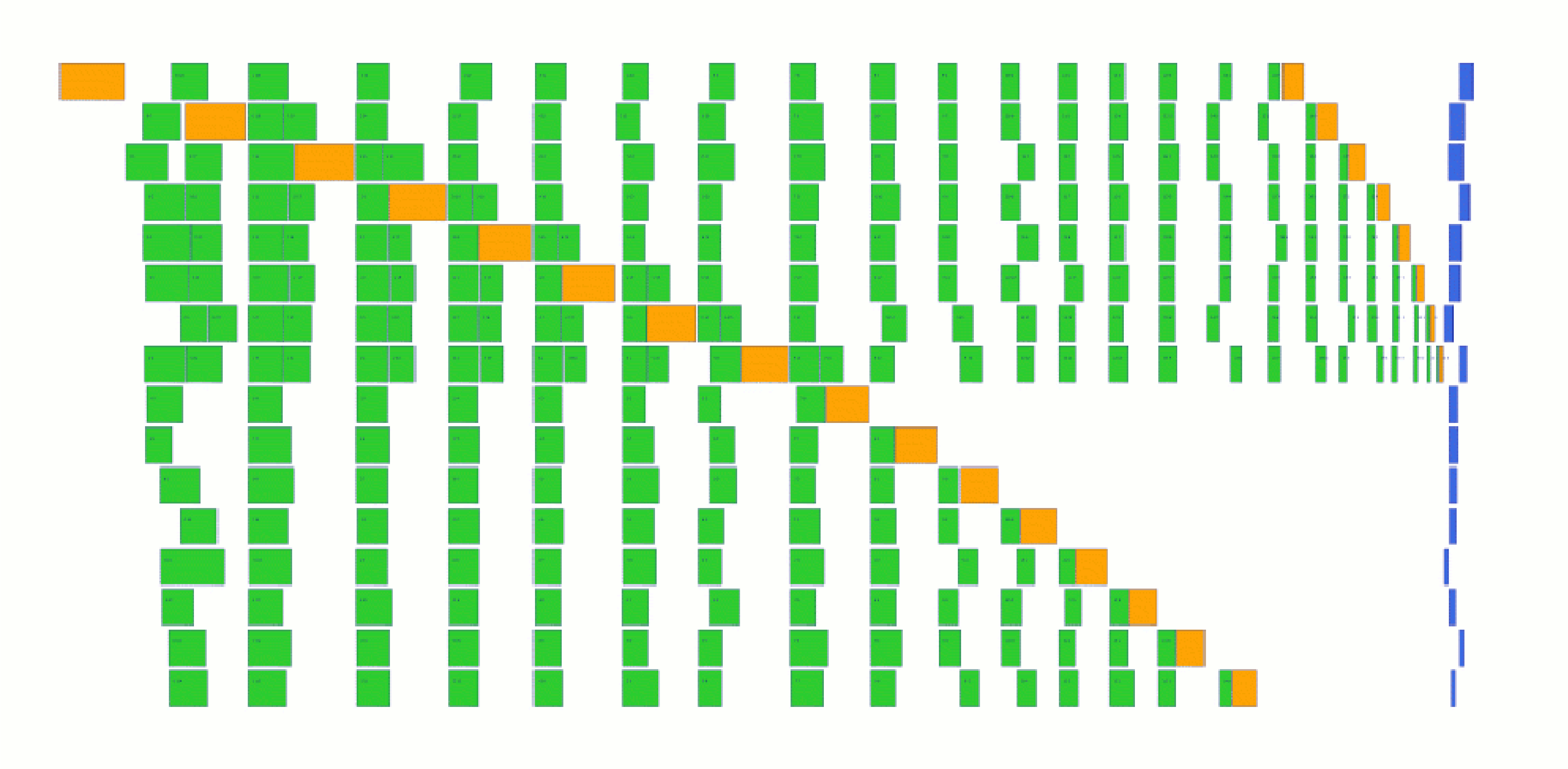
- green: pdgemm
- blue : pdgetf2
- red : pdlswap
- magenta : pdtrsm
- cyan : topset
- yellow: igamn2d

What about strong scalability?

N = 1536

NB = 64

procs = 16



Reduce Algorithms: Introduction

Example of applications:

a) in block iterative methods (iterative methods with multiple right-hand sides or iterative eigenvalue solvers),

b) in dense large and more square QR factorization where they are used as the panel factorization step, or more simply

c) in linear least squares problems which the number of equations is extremely larger than the number of unknowns.

Reduce Algorithms: Introduction

Example of applications:

a) in block iterative methods (iterative methods with multiple right-hand sides or iterative eigenvalue solvers),

b) in dense large and more square QR factorization where they are used as the panel factorization step, or more simply

c) in linear least squares problems which the number of equations is extremely larger than the number of unknowns.

The main characteristics of those three examples are that

a) there is **only one column of processors involved** but several processor rows,

b) **all the data is known from the beginning**,

c) and **the matrix is dense**.

Reduce Algorithms: Introduction

Example of applications:

a) in block iterative methods (iterative methods with multiple right-hand sides or iterative eigenvalue solvers),

b) in dense large and more square QR factorization where they are used as the panel factorization step, or more simply

c) in linear least squares problems which the number of equations is extremely larger than the number of unknowns.

The main characteristics of those three examples are that

a) there is **only one column of processors involved** but several processor rows,

b) **all the data is known from the beginning**,

c) and **the matrix is dense**.

Various methods already exist to perform the QR factorization of such matrices:

a) Gram-Schmidt (**mgs(row),cgs**),

b) Householder (**qr2**, **qrf**),

c) or **CholeskyQR**.

We present a new method:

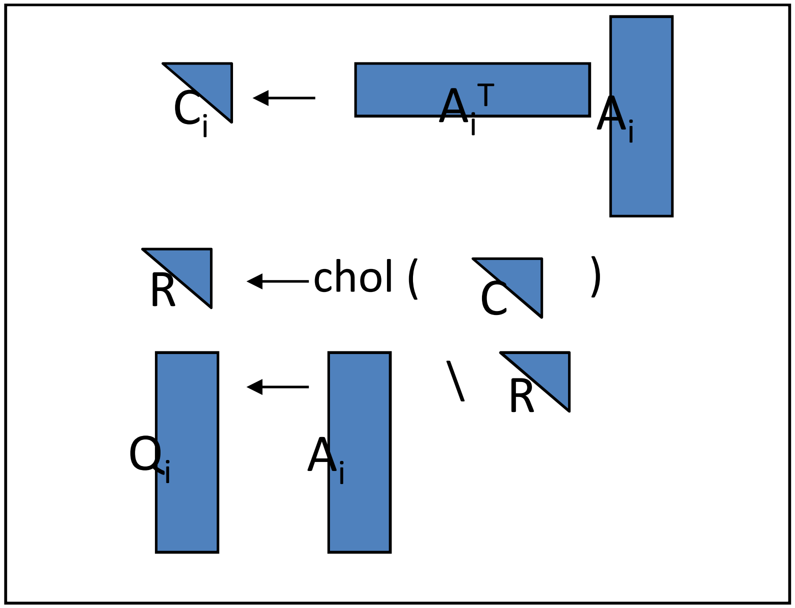
Allreduce Householder (**rhh_qr3**, **rhh_qrf**).

AllReduce Algorithms

1) Tall Skinny matrices: Application
2) **The CholeskyQR algorithm (see MATH6664)**
3) AllReduce Householder factorization
4) Application to dense LU and dense QR factorizations

The CholeskyQR Algorithm

SYRK:	$C := A^T A$	(mn^2)
CHOL:	$R :=$ chol(C)	$(n^3/3)$
TRSM:	$Q := A \backslash R$	(mn^2)



Bibliography

- First reference in an ETHZ tech. report from W. Gander (1980):

  In fact even the method, although we don't recommend it, of computing Q via the Cholesky decomposition of A^TA,

  $$A^TA = R^TR$$

  and to put

  $$Q=AR^{-1}$$

  seems to be superior than Schmidt.

- Then Å. Björck (p.67, 1997):

  As pointed out by Gander (1980), even computing Q via the Cholesky decomposition of A^TA seems to be superior to CGS.

Bibligraphy (cont.)

- A. Stathopoulos and K. Wu, **A block orthogonalization procedure with constant synchronization requirements**, *SIAM Journal on Scientific Computing*, 23(6):2165-2182, 2002.

- Popularized by iterative eigensolver libraries:
  1) **PETSc** (Argonne National Lab.) through **BLOPEX** (A. Knyazev, UCDHSC),
  2) **HYPRE** (Lawrence Livermore National Lab.) through **BLOPEX**,
  3) **Trilinos** (Sandia National Lab.) through **Anasazi** (R. Lehoucq, H. Thornquist, U. Hetmaniuk),
  4) **PRIMME** (A. Stathopoulos, Coll. William & Mary).

Fall 2007: MATH 6664 – Numerical Linear Algebra project

Abstract

The goal of this project is to analyze, program and experiments the *Cholesky QR* orthogonalization scheme then write a small report about it. You will be evaluated on the quality of the report, the quality of your experiments and your coding performance. This consists in the first part of the project, it is worth 75 points (over 150 of the project, over 400 of the total).

The orthogonalization scheme to be studied is *Cholesky QR* and it can be described as follows:

algorithm: *Cholesky QR*.
input data: A is m-by-n and full rank.
output data: Q and R, such that: $A = QR$, Q is m-by-n with orthonormal columns, R is n-by-n upper triangular with positive diagonal elements.

1. (`SYRK`) $C \leftarrow A^T A$,
2. (`POTRF`) $R \leftarrow$ `chol`(C),
3. (`TRSM`) $Q \leftarrow A/R$.

The first time (to my knowledge) this algorithm has been mentionned is in Gander [3], unfortunatley Gander forgets to give further reference. Stathopoulos and Wu have recently presented a new method (`SVDQB`) closely related see [7].

Part I: Analysis
The first part of your report will consist in an analysis of the algorithm. The analysis of the algorithm needs to have:

1. explanation on why this algorithm performs the reduced QR-factorization of A,
2. FLOPS count for this algorithm,
3. stability analysis with some experiments to assess the results (Matlab experiments),
4. scalability analysis in a parallel distributed framework.

Part II: Implementation
You need to provide me with three different flavors for the implementation of the algorithm: Matlab, sequential code (optional) and parallel distributed code.

Part III: Experimentation
Finally, explanation of the codes performance based on the analysis made in the first part is expected.

General remarks

1. When it is asked to bound a quantity, it is implicitly assumed that the tighter the better ...

1 Analysis

1.1 Theoretical analysis

Question 1 *why the algorithm Cholesky QR generates a QR factorization?*

1.2 FLOP count

You can get FLOP counts for standard operations in for example [9, p.82] or [2, p.120]. In our case, we have:

`SYRK`	mn^2
`POTRF`	$1/3n^3$
`TRSM`	mn^2

Question 2 *rederive the FLOP count ($= mn^2$) for SYRK operation as accuratley as possible.*

Question 3 *how does the FLOP count of* `Cholesky QR` *compare with Gram-Schmidt QR? with Householder QR? when the R-factor only is needed, or when the Q-factor and the R-factor are needed? when $m \gg n$, or when $m = n$?*

1.3 Stability analysis

1.3.1 SYRK operation

Defining $\mathbf{u}$ as the unit roundoff, assuming that

$$m\mathbf{u} < 0.1, \tag{1}$$

baring overflow and underflow, and assuming that the standard floating-point arithmetic model is respected (see [9, Eq. (13.7)]), reusing the error analysis of a scalar product $fl(x^T y)$, we get that the elements $\bar{c}_{ij}$ of $\bar{C} = fl(A^T A)$ are such that

$$\bar{c}_{ij} = fl\left(\sum_{k=1}^{m} a_{ki} a_{jk}\right) = \sum_{k=1}^{m} a_{ki} a_{jk}(1+\delta), \text{ where } |\delta| < 1.06m\mathbf{u}.$$

(See [1, p.43] or [6, p.387], for more explanations.)
It follows that

Theorem 1 (see [1, p.43] or [6, p.387]).

$$\bar{C} = A^T A + E^{(1)}, \text{ where } |e_{ij}^{(1)}| < 1.06m\mathbf{u}|a_i|^T|a_j| \leq 1.06m\mathbf{u}\|a_i\|_2\|a_j\|_2. \tag{2}$$

The `SYRK` operation computes only half of the matrix $\bar{C}$. Consequently the matrix $\bar{C}$ is symmetric.

Question 4 *Do you think Equation (1) represents a reasonnable assumption?*

Question 5 *Does Equation (2) imply backward stability of the* `SYRK` *operation? Make a comment on the*

Question 6 *Give $c_1(m,n)$ such that*

$$\|E^{(1)}\|_2 < c_1(m,n)\mathbf{u}\|A\|_2^2. \tag{3}$$

(Hint: you might want to use Appendix A.)

Question 7 *Prove that for all $i = 1$ to n,*

$$|\lambda_i(\bar{C}) - \sigma_i(A)^2| < c_2(m,n)\mathbf{u}\|A\|_2^2.$$

(Hint: We have seen in class something called Weyl's theorem and it is given in Appendix B.)

Question 8 *Using Question 7, give an assumption on A of the form*

$$c_3(m,n)\mathbf{u}\kappa(A)^2 < 1,$$

so that we can guarantee $\bar{C}$ to be symmetric positive definite. This enables you to compute a upper bound for the condition number of $\bar{C}$ in term of A. Give γ_1 such that:

$$\kappa(\bar{C}) \leq \gamma_1\kappa(A)^2.$$

1.3.2 POTRF operation

The following result on `POTRF` is extracted from [1, Th. 2.2.2, p.49].

Theorem 2 (see [1, Th. 2.2.2, p.49] or [6, Th 10.3, p.197]). *Let $\bar{C}$ be a n-by-n symmetric positive definite matrix. Provided that*

$$c_4(m,n)\mathbf{u}\kappa(\bar{C}) < 1, \quad \text{where } c_4(n) = 20n^{3/2}, \tag{4}$$

the Cholesky factor of $\bar{C}$ can be computed without breakdown, and the computed $\bar{R}$ satisfy

$$\bar{R}^T\bar{R} = \bar{C} + E^{(2)}, \quad \|E^{(2)}\|_2 < c_5(n)\mathbf{u}\|\bar{R}\|_2^2, \quad \text{where } c_5(n) = 2.5n^{3/2}. \tag{5}$$

Question 9 *Explain what the assumption made with Equation (4) means.*

Question 10 *Does Equation (5) imply backward stability of Cholesky factorization?*

Question 11 *Give an assumption on A such that Assumption (4) holds.*

Using Weyl's theorem (see Appendix B) again, we can prove that the singular value of $\bar{R}$ and the singular value of A are essentially the same. This means that we can prove something like:

$$\sigma_i(A)(1 - c_7(m,n)\mathbf{u}\kappa(A)^2) \leq \sigma_i(\bar{R}) \leq \sigma_i(A)(1 + c_7(m,n)\mathbf{u}\kappa(A)^2). \tag{6}$$

To keep things easy in the following, we will assume:

1.3.3 TRSM operation

The following result for `TRSM` is extracted from [6, Chap. 8], [5], or [9, Lecture 17].

Theorem 3 (see [6, Chap. 8],[5]). *Let the triangular systems $Rx = b$ where R is nonsingular n-by-n matrix be solved by back substitution (any ordering). Then the computed solution $\bar{x}$ satisfies*

$$(T + \Delta T)\bar{x} = b, \text{ where } |\Delta T| \leq 1.12n\mathbf{u}|T|.$$

Higham does not use $1.12n\mathbf{u}$ (old style error analysis) but rather γ_n (modern error analysis) which is defined as $\gamma_n = n\mathbf{u}(1-n\mathbf{u})^{-1}$ (see [6, Lemma 3.1, p.63]). You can check that under the Assumption (1): $m\mathbf{u} < 0.1$, we have $\gamma_n < 1.12n\mathbf{u}$.
In `TRSM`, there are m triangular solves (one per line of A). Higham's theorem gives us that, for any $i = 1,\ldots,m$,

$$\bar{Q}_{i,:}(\bar{R} + \Delta R_i) = A_{i,:}, \text{ where } |\Delta R_i| \leq 1.12n\mathbf{u}|\bar{R}|.$$

In our context, we rather seek an error perturbation in the right-hand side $A_{i,:}$ rather than on the matrix $\bar{R}$. First we note that if $|\Delta R_i| \leq 1.12n\mathbf{u}|\bar{R}|$, then $\|\Delta R_i\|_2 \leq 1.12n\mathbf{u}\sqrt{n}\|\bar{R}\|_2$. (We have used the fact that if $|A| \leq |B|$ then $\|A\|_2 \leq \sqrt{\text{rank}(B)}\|B\|_2$, see [6, Lemma 6.6 (c), p.111].)
This last equality rewrites:

$$\bar{Q}_{i,:}\bar{R} = A_{i,:} + E_{i,:}^{(3)}, \text{ where } \|E_{i,:}^{(3)}\|_2 \leq 1.12n\mathbf{u}\sqrt{n}\|\bar{Q}_{i,:}\|_2\|\bar{R}\|_2.$$

Combining all those line from $i = 1$ to m, we get

$$\bar{Q}\bar{R} = A + E^{(3)}, \quad \|E^{(3)}\|_2 \leq 1.12n^2\mathbf{u}\|\bar{Q}\|_2\|\bar{R}\|_2$$

(see once more [6, Lemma 6.6 (a), p.111]).
Now it is possible to prove that:

$$\|\bar{Q}\|_2| = 1 + c_8(m,n)\mathbf{u}\kappa^2(A),$$

(this is tedious) but once more for simplicity, we skip it and take the result of this analysis to be:

$$\bar{Q}\bar{R} = A + E^{(3)}, \quad \|E^{(3)}\|_2 \leq \phi(m,n)\mathbf{u}\|\bar{R}\|_2 \tag{8}$$

Question 12 *Using Equation (2), Equation (5) and Equation (8), prove that*

$$\bar{R}^T(I - \bar{Q}^T\bar{Q})\bar{R} = E \quad \text{where } E = E^{(1)} + E^{(2)} - A^T E^{(3)} - E^{(3)T}A - E^{(3)T}E^{(3)}$$

Question 13 *Using Equation (3), Equation (5), Equation (7) and Equation (8), we can finally prove that the Q-factor computed by the QR Cholesky, $\bar{Q}$, is such that*

$$\|I - \bar{Q}^T\bar{Q}\|_2 \leq \psi(m,n)\mathbf{u}\kappa^2(A). \tag{9}$$

1.4 scalability analysis in a parallel distributed framework

Question 14 *For each of those algorithms: CGS, MGS, and Cholesky QR, count the total number of* `MPI_Allreduce` *performed and the quantity of data involved in the all* `MPI_Allreduce`.

Since our messages are small, MPI is likely to use a binary tree to perform the `MPI_Allreduce`, therefore a model for one `MPI_Allreduce` on P processes is:

$$t_c = 2\log_2(P)(\alpha \cdot \texttt{message\_size} + \beta).$$

where α is the inverse of the bandwidth and β the latency.

Question 15 *Calling γ the floating-point ratio of one processor, give the scalability of our three algorithms. Discuss.*

2 Implementation

You need to provide me with three different flavors for the implementation of the algorithm:

1. Matlab
2. sequential code
3. parallel distributed code running on the cluster

All codes need to be checked. The sequential code is optionnal since one can use without (much) penalty the parallel one.

3 Experimentation

3.1 Verification in Matlab of the sharpness of the bound obtained in Equation (8) and Equation (9)

3.2 Compare experimentally the quality of *Cholesky QR* with CGS, MGS, and Householder.

3.3 Analyse the performance of your parallel code and compare with theory you have developed in section 1.4

4 Conclusion

You write whatever you feel like at this point.

References

[1] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996. ISBN 0-89871-360-9. xvii+408 pp.

[2] S. Blackford and J. Dongarra. LAWN 41: installation guide for LAPACK. LAPACK Working Note UT-CS-92-151, University of Tennessee, Mar. 1992.

[3] W. Gander. Algorithms for the QR decomposition. Tech. Report 80-02, ETH, Zürich, Switzerland, 1980.

[4] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996. ISBN 0-8018-5413-X (hardback), 0-8018-5414-8 (paperback). xxvii+694 pp.

[5] N. J. Higham. The accuracy of solutions to triangular systems. *SIAM Journal on Numerical Analysis*, 26(5):1252–1265, Oct. 1989.

[6] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002. ISBN 0-89871-521-0. xxx+680 pp.

[7] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM Journal on Scientific Computing*, 23(6):2165–2182, 2002.

[8] G. W. P. Stewart. *Matrix Algorithms. Volume I: Basic Decompositions*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998. ISBN 0-89871-414-1. xx+458 pp.

[9] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997. ISBN 0-89871-361-7. xii+361 pp.

A Absolute value of matrices and norm – Higham [6] Lemma 6.6.

Lemma 1 ([6, Lemma 6.6, p111]). *Let A and B be m-by-n symmetric matrices. Then*

1. *If $\|a_j\|_2 \leq \|b_j\|_2$, $j = 1:n$, then*
$$\|A\|_F \leq \|B\|_F, \quad \|A\|_2 \leq \sqrt{rank(B)}\|B\|_2, \quad |A| \leq ee^T|B|.$$
2. *If $|A| \leq B$ then $\|A\|_2 \leq \|B\|_2$.*
3. *If $|A| \leq |B|$ then $\|A\|_2 \leq \sqrt{rank(B)}\|B\|_2$.*
4. *$\|A\|_2 \leq \||A|\|_2 \leq \sqrt{rank(A)}\|A\|_2$.*

B Weyl's theorem (or more accuratley a corollary of Weyl's theorem)

Corollary 1 (e.g. [8, Corollary 4.3.1, p.69] or [4, Corollary 8.6.3, p.449]). *Let A and E be m-by-n symmetric matrices. Then*

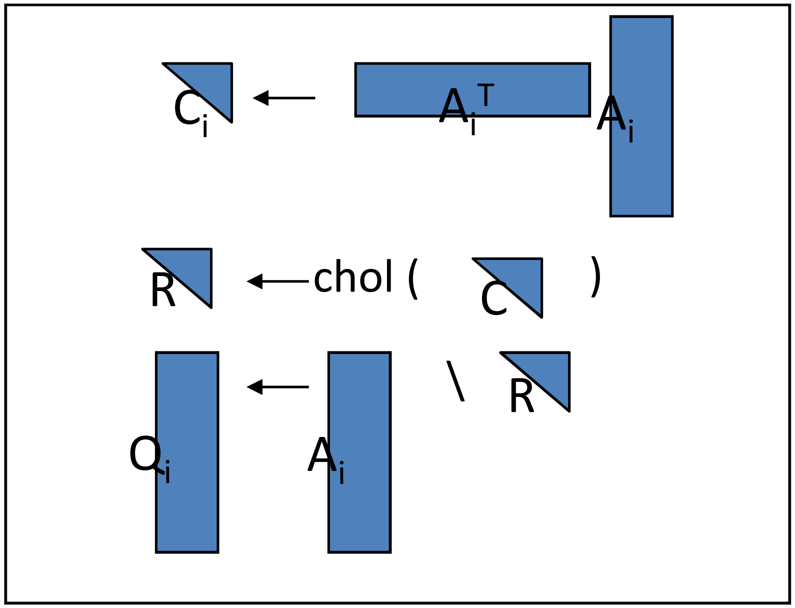
$$|\sigma_i(A+E) - \sigma_i(A)| \leq \|E\|_2, \quad i = 1,2,\ldots,p.$$

Corollary 2 (e.g. [8, Theorem 4.34(4), p.72] or [4, Corollary 8.1.6, p.396]). *Let A and E be n-by-n symmetric matrices. Then*

$$|\lambda_i(A+E) - \lambda_i(A)| \leq \|E\|_2, \quad i = 1,2,\ldots,p.$$

Question 1.

SYRK:	$C := A^T A$	(mn^2)
CHOL:	$R := \mathrm{chol}(C)$	($n^3/3$)
TRSM:	$Q := A \backslash R$	(mn^2)



- Why does Cholesky QR generates a QR factorization of the matrix A?

Questions 2 and 3.

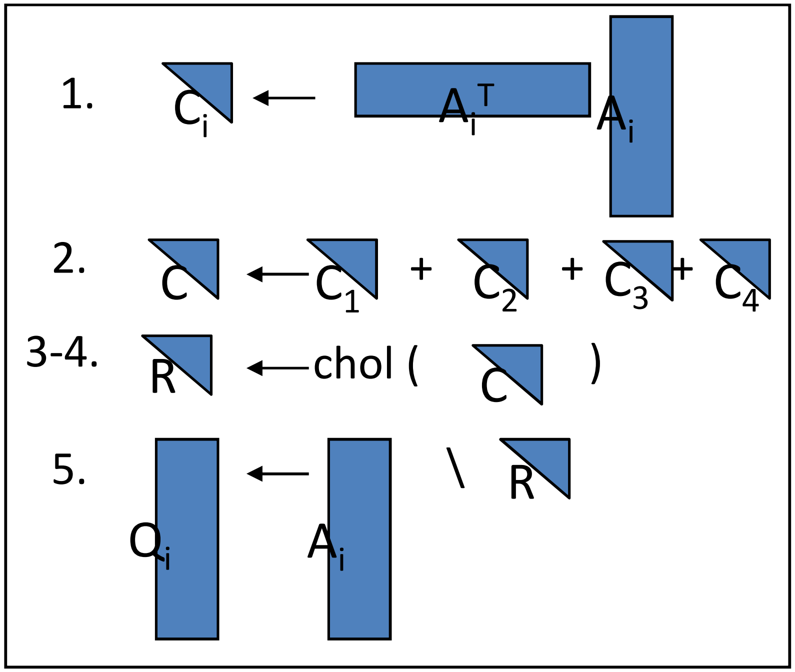
When Only R is needed (R on all the processes)				
	FLOPs (total)			
CholeskyQR	$mn^2 + n^3/3$			
CGram-Schmidt	$2mn^2$			
Mgram-Schmidt	$2mn^2$			
Householder	$2mn^2 - 2/3n^3$			

Q and R are needed				
	FLOPs (total)			
CholeskyQR	$2mn^2 + n^3/3$			
CGram-Schmidt	$2mn^2$			
Mgram-Schmidt	$2mn^2$			
Householder	$4mn^2 - 4/3n^3$			

Parallel distributed CholeskyQR

The CholeskyQR method in the parallel distributed context can be described as follows:

1: SYRK:	C:= A^TA	(mn^2)
2: MPI_Reduce:	C:= $\text{sum}_{\text{procs}}$ C	(on proc 0)
3: CHOL:	R := chol(C)	($n^3/3$)
4: MPI_Bdcast	Broadcast the R factor on proc 0 to all the other processors	
5: TRSM:	Q := A\R	(mn^2)



This method is extremely fast. For two reasons:

1. first, there is **only one or two communications phase**,
2. second, **the local computations are performed with fast operations**.

Another advantage of this method is that the resulting code is exactly four lines,

3. so the method is **simple** and relies heavily on other libraries.

Despite all those advantages,

4. this method is **highly unstable**.

Operations/Latency/Bandwidth

Questions 14 and 15

When Only R is needed (R on all the processes)				
	FLOPs (total)	# msg	Vol data exchanged	FLOPs
CholeskyQR	$mn^2 + n^3/3$	$2\log_2(p)$	$2\log_2(p)$ ($n^2/2$)	$(mn^2)/p + n^3/3$
CGram-Schmidt	$2mn^2$	$2n \log_2(p)$	$2\log_2(p)$ ($n^2/2$)	$(2mn^2)/p$
MGram-Schmidt	$2mn^2$	$n^2 \log_2(p)$	$2\log_2(p)$ ($n^2/2$)	$(2mn^2)/p$
Householder	$2mn^2 - 2/3n^3$	$2n \log_2(p)$	$2\log_2(p)$ ($n^2/2$)	$(2mn^2 - 2/3n^3)/p$

Q and R are needed				
	FLOPs (total)	# msg	Vol data exchanged	FLOPs
CholeskyQR	$2mn^2 + n^3/3$	$2 \log_2(p)$	$2 \log_2(p)$ ($n^2/2$)	$(2mn^2)/p + n^3/3$
CGram-Schmidt	$2mn^2$	$4 n \log_2(p)$	$4 \log_2(p)$ ($n^2/2$)	$(2mn^2)/p$
MGram-Schmidt	$2mn^2$	$2 n^2 \log_2(p)$	$4 \log_2(p)$ ($n^2/2$)	$(2mn^2)/p$
Householder	$4mn^2 - 4/3n^3$	$4 n \log_2(p)$	$4 \log_2(p)$ ($n^2/2$)	$(4mn^2 - 4/3n^3)/p$

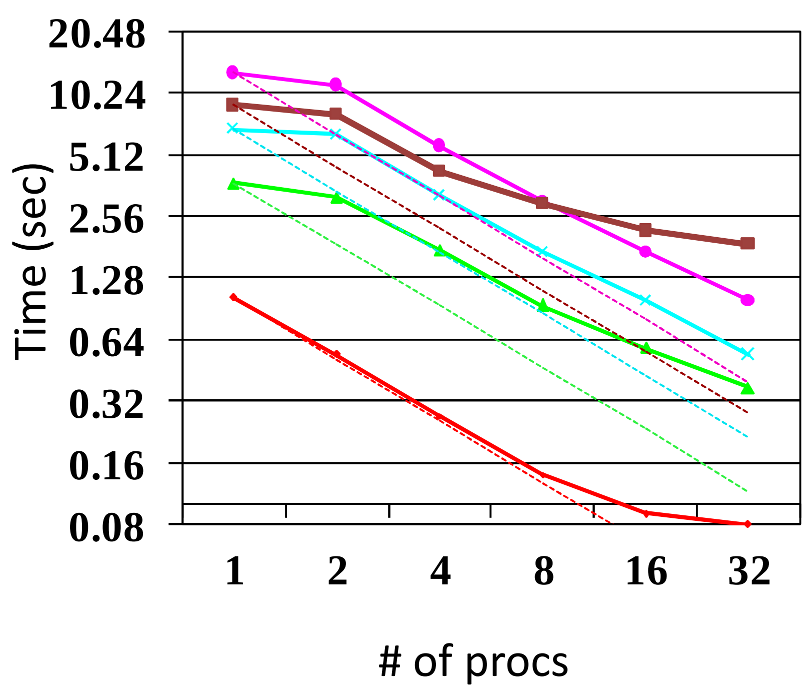
The total time is

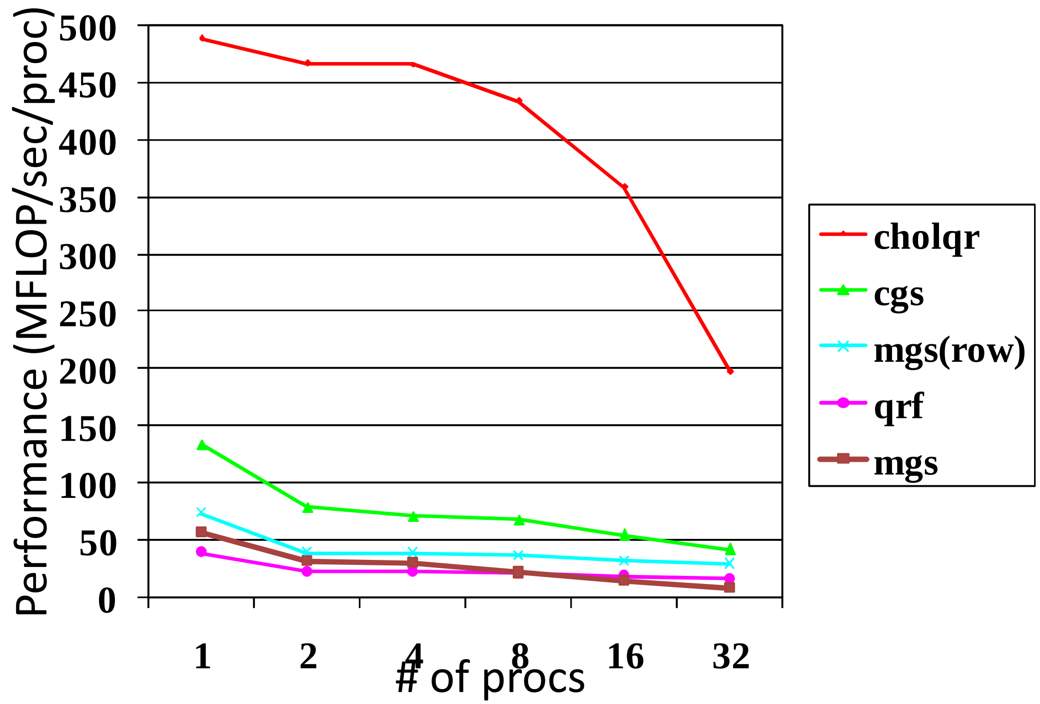
α * (# msg) + β * (vol data exchanged) + γ * (FLOPs)

In this experiment, we fix the problem: **m=100,000** and **n=50**.

Efficient enough?

Question 18



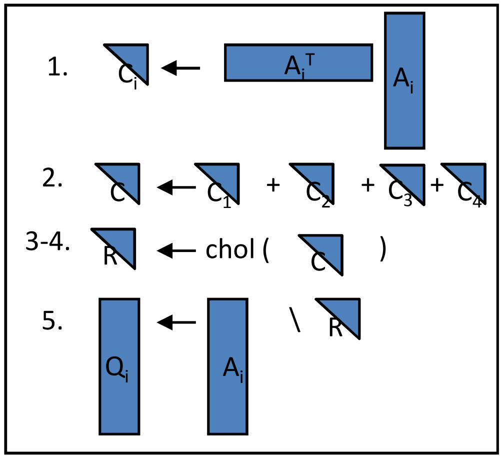


# of procs	cholqr		cgs		mgs(row)		qrf		mgs	
1	489.2	(1.02)	134.1	(3.73)	73.5	(6.81)	39.1	(12.78)	56.18	(8.90)
2	467.3	(0.54)	78.9	(3.17)	39.0	(6.41)	22.3	(11.21)	31.21	(8.01)
4	466.4	(0.27)	71.3	(1.75)	38.7	(3.23)	22.2	(5.63)	29.58	(4.23)
8	434.0	(0.14)	67.4	(0.93)	36.7	(1.70)	20.8	(3.01)	21.15	(2.96)
16	359.2	(0.09)	54.2	(0.58)	31.6	(0.99)	18.3	(1.71)	14.44	(2.16)
32	197.8	(0.08)	41.9	(0.37)	29.0	(0.54)	15.8	(0.99)	8.38	(1.87)

MFLOP/sec/proc

Time in sec

Simple enough?



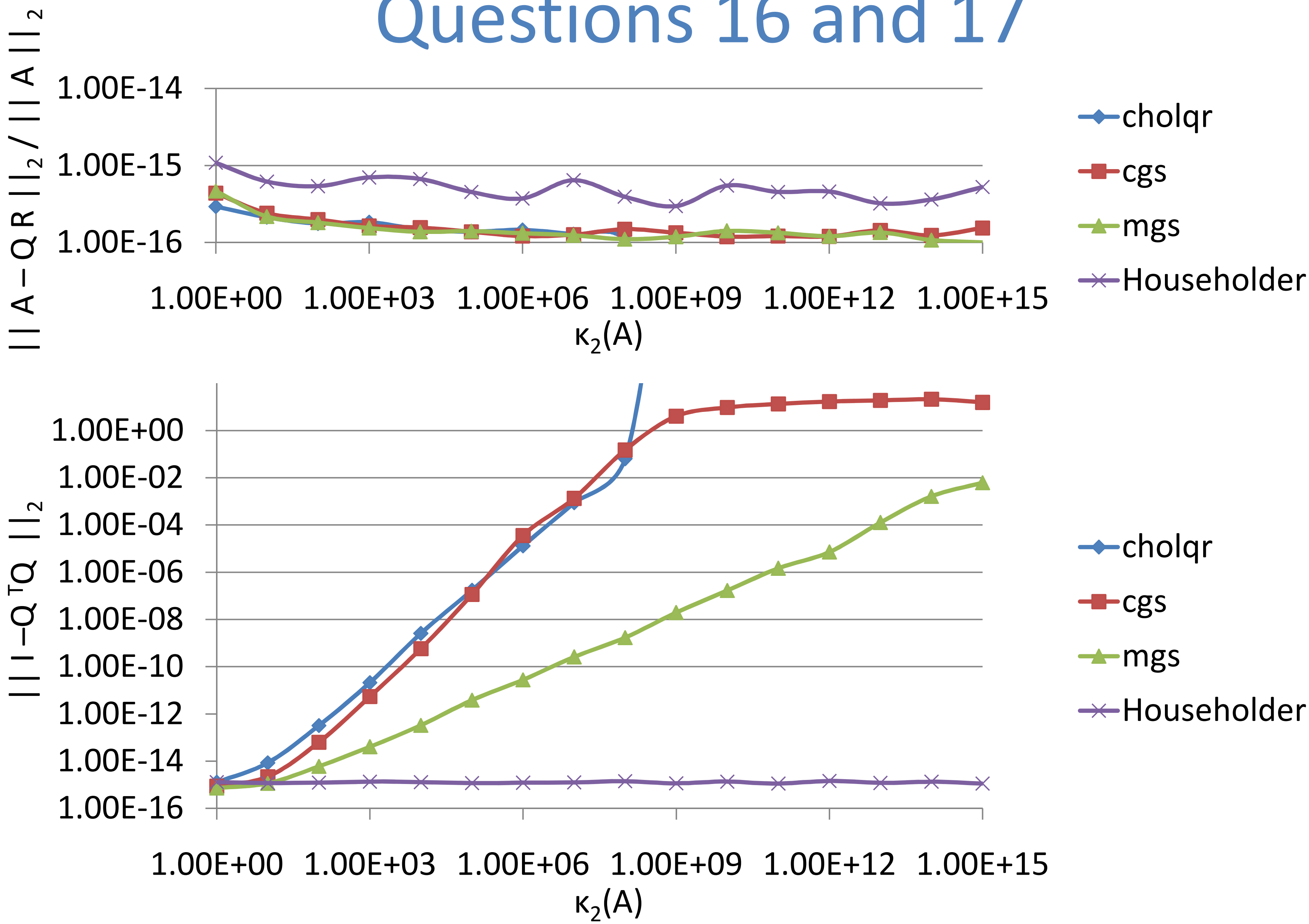

```
int choleskyqr_A_v0(int mloc, int n, double *A, int lda, double *R, int ldr,
            MPI_Comm mpi_comm){

            int info;
            cblas_dsyrk( CblasColMajor, CblasUpper, CblasTrans, n, mloc,
                        1.0e+00, A, lda, 0e+00, R, ldr );
            MPI_Allreduce( MPI_IN_PLACE, R, n*n, MPI_DOUBLE, MPI_SUM, mpi_comm );
            lapack_dpotrf( lapack_upper, n, R, ldr, &info );
            cblas_dtrsm( CblasColMajor, CblasRight, CblasUpper, CblasNoTrans, CblasNonUnit,
                        mloc, n, 1.0e+00, R, ldr, A, lda );
            return 0;

}
```

(And OK, you might want to add an MPI user defined datatype to send only the upper part of R)

Stable enough?

m=100, n=50

Questions 16 and 17



SYRK:	$C := A^T A$	(mn^2)
CHOL:	$R := \text{chol}(C)$	$(n^3/3)$
TRSM:	$Q := A \backslash R$	(mn^2)

Stable enough?

Questions 4 to 14

Theorem 1 (see [Björck, 1997, p.43] or [Higham, 2002, p.387]).

The computed C (normal equations) is such that

$$C = A^T A + E^{(1)}, \text{ where } |e_{ij}^{(1)}| < 1.06\, m\, \mathbf{u}\, |a_i|^T |a_j|.$$

Theorem 2 (see [Björck, 1997, Th. 2.2.2, p.49] or [Higham, 2002, Th 10.3, p.197]).

Let C be a n-by-n symmetric positive definite matrix. Provided that

$$c_4(m,n)\mathbf{u}\kappa(C) < 1, \text{ where } c_4(n) = 20n^{3/2},$$

the Cholesky factor of C can be computed without breakdown, and the computed satisfy

$$R^T R = C + E^{(2)},\ \| E^{(2)} \|_2 < c_5(n)\mathbf{u}\|R\|_2, \text{ where } c_5(n) = 2.5n^{3/2}.$$

Theorem 3 (see [Higham, 2002, Chap. 8]).

Let the triangular systems Rx = b where R is nonsingular n-by-n matrix be solved by back substitution (any ordering). Then the computed solution x satisfies

$$(T + \Delta T)\, x = b, \text{ where } |\Delta T| < 1.12 n \mathbf{u} |T|.$$

SYRK:	$C := A^T A$	(mn^2)
CHOL:	$R := \mathrm{chol}(C)$	$(n^3/3)$
TRSM:	$Q := A \backslash R$	(mn^2)

Stable enough?

Questions 4 to 14

Final Theorem (see [MATH6664]).

Let A be a m-by-n matrix. Provided that

$$\pi(m,n)\mathbf{u}\kappa(A)^2 < 1,$$

then

the Cholesky factor of C of the computed normal equations can be computed without breakdown and the computed Q and R satisfy

$$QR = A + E^{(3)}, \text{ where } \|E^{(3)}\|_2 < \phi(m,n)\,\mathbf{u}\,\|A\|_2$$

$$\|I - Q^TQ\|_2 < \psi(m,n)\,\mathbf{u}\,\kappa(A)^2$$

Summary of stability results

Method	$\|\| I - Q^T Q \|\|_2$	Reference
Householder/Givens	$\psi(m,n)\,\mathbf{u}$	Wilkinson (1965-1966)
Iterated Gram-Schmidt	$\psi(m,n)\,\mathbf{u}$	many references
Modified Gram-Schmidt	$\psi(m,n)\,\mathbf{u}\,\kappa(A)$	Björck (1967)
Cholesky QR	$\psi(m,n)\,\mathbf{u}\,\kappa(A)^2$	Stathopoulos and Wu (2002)
Classical Gram-Schmidt (P)	$\psi(m,n)\,\mathbf{u}\,\kappa(A)^2$	Giraud, Langou, Rozložník, van den Eshof (2005) Barlow, Smoktuniwicz, Langou (2006)
Classical Gram-Schmidt (S)	$\psi(m,n)\,\mathbf{u}\,\kappa(A)^{n-1}$	Kiełbasiński (1974) Barlow, Smoktuniwicz, Langou (2006)

Goal for the next slides

- The goal is to derive a new technique, fairly general, that follows the idea and principle of the CholeskyQR algorithm. The resulting code will be simple and efficient. We will only use Householder transformation so our method will be **stable**.

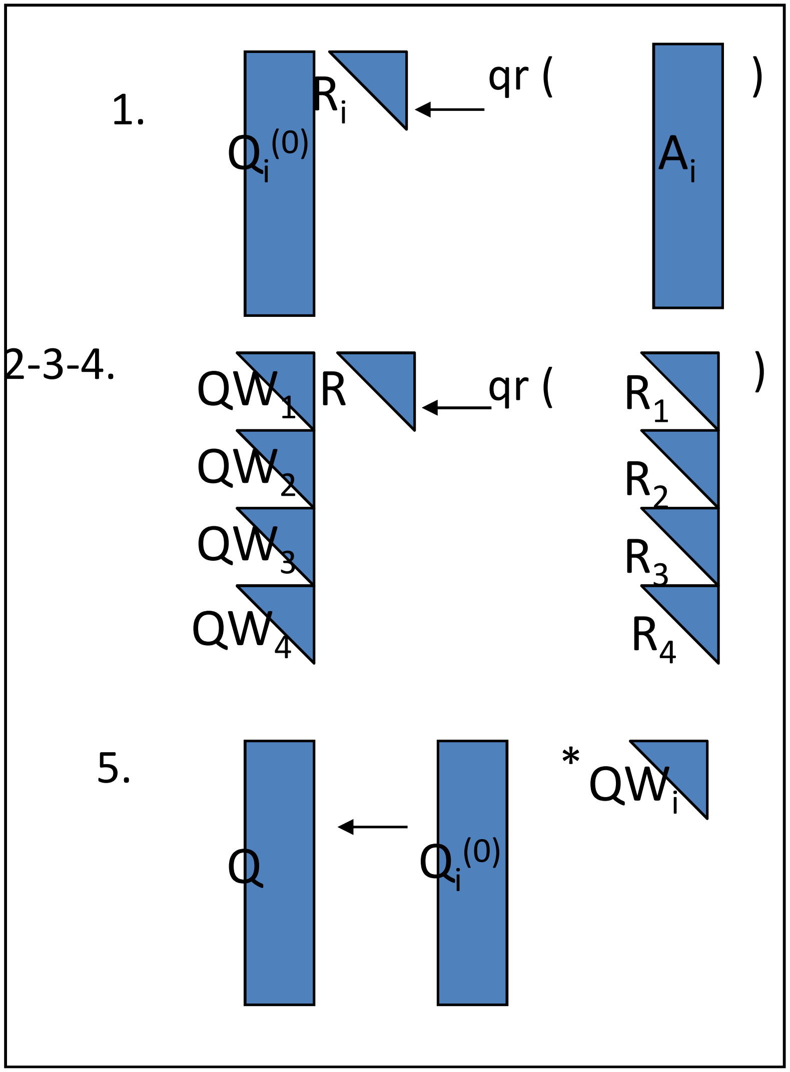
AllReduce Algorithms

1) Tall Skinny matrices: Application
2) The CholeskyQR algorithm (see MATH6664)
3) **AllReduce Householder factorization**
4) Application to dense LU and dense QR factorizations

Reduce Algorithms

The gather-scatter variant of our algorithm can be summarized as follows:

1. perform local QR factorization of the matrix A
2. gather the p R factors on processor 0
3. perform a QR factorization of all the R put the ones on top of the others, the R factor obtained is the R factor
4. scatter the the Q factors from processor 0 to all the processors
5. multiply locally the two Q factors together, done.



Reduce Algorithms

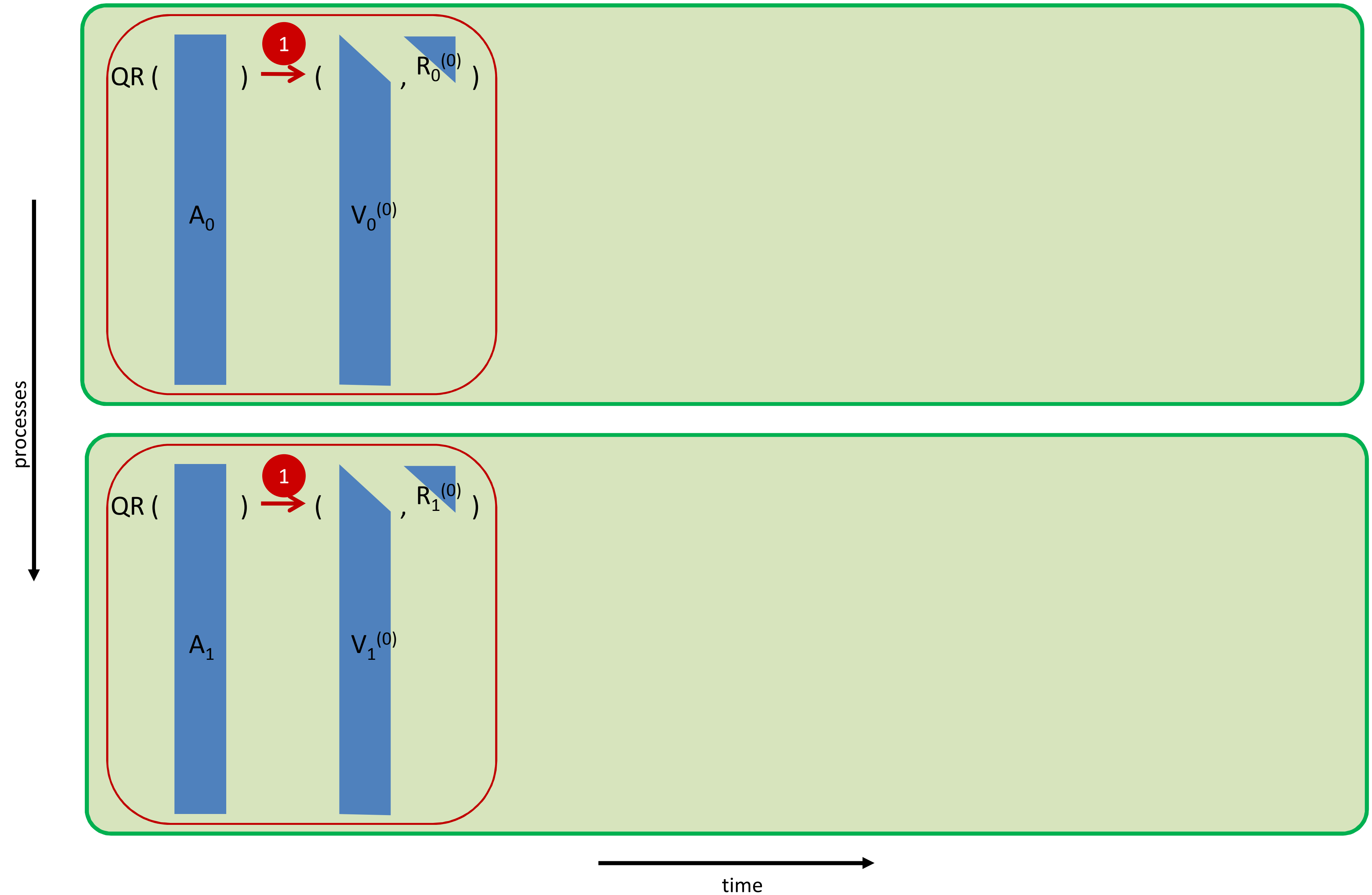
- This is the scatter-gather version of our algorithm.

- This variant is not very efficient for two reasons:
  - first the communication phases 2 and 4 are highly involving processor 0 ;
  - second the cost of step 3 is $p/3*n^3$, so can get prohibitive for large p.

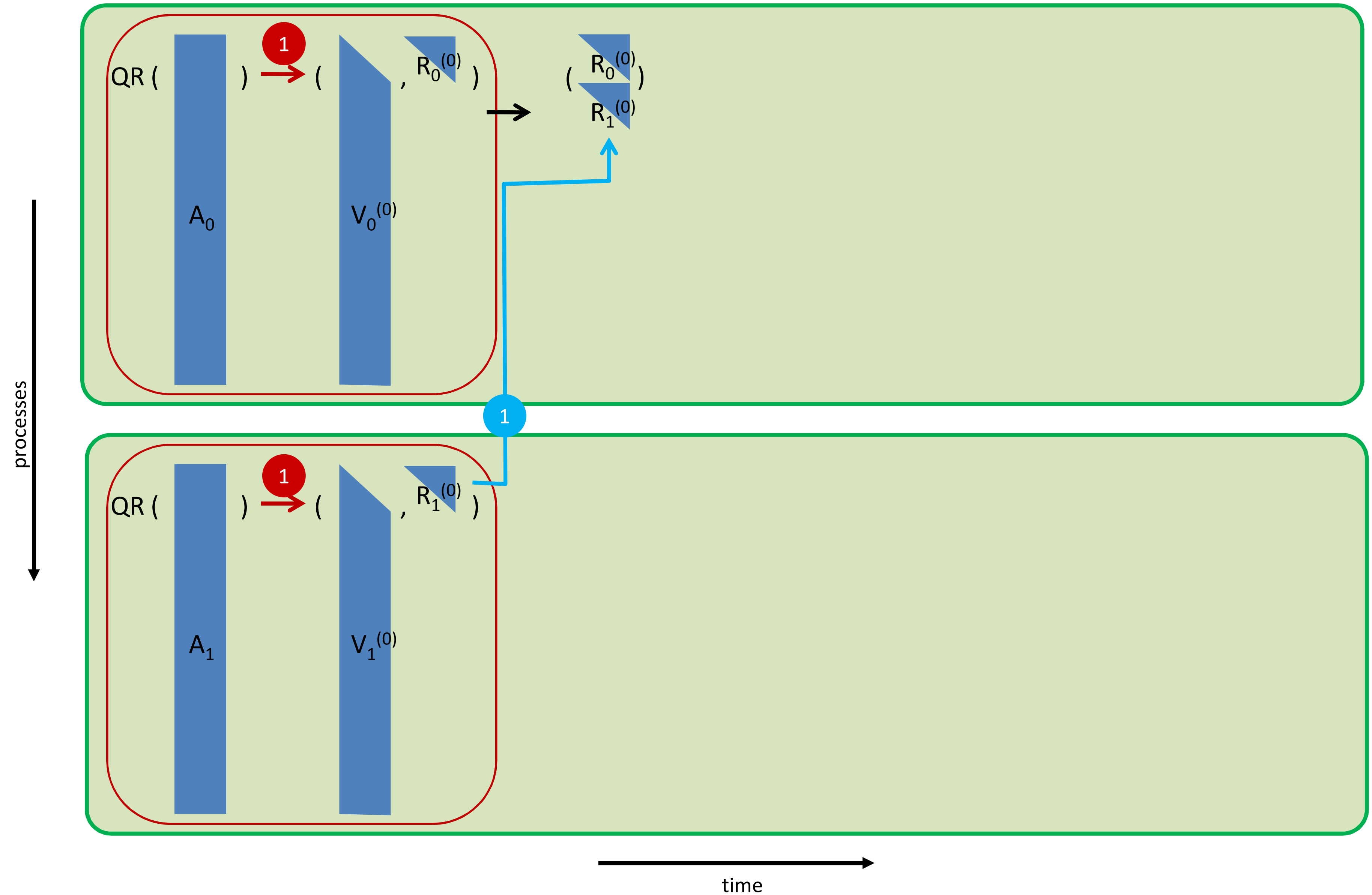
- Note that the CholeskyQR algorithm can also be implemented in a **scatter-gather** way but **reduce-broadcast**. This leads naturally to the algorithm presented below where a reduce-broadcast version of the previous algorithm is described. This will be our final algorithm.

On two processes

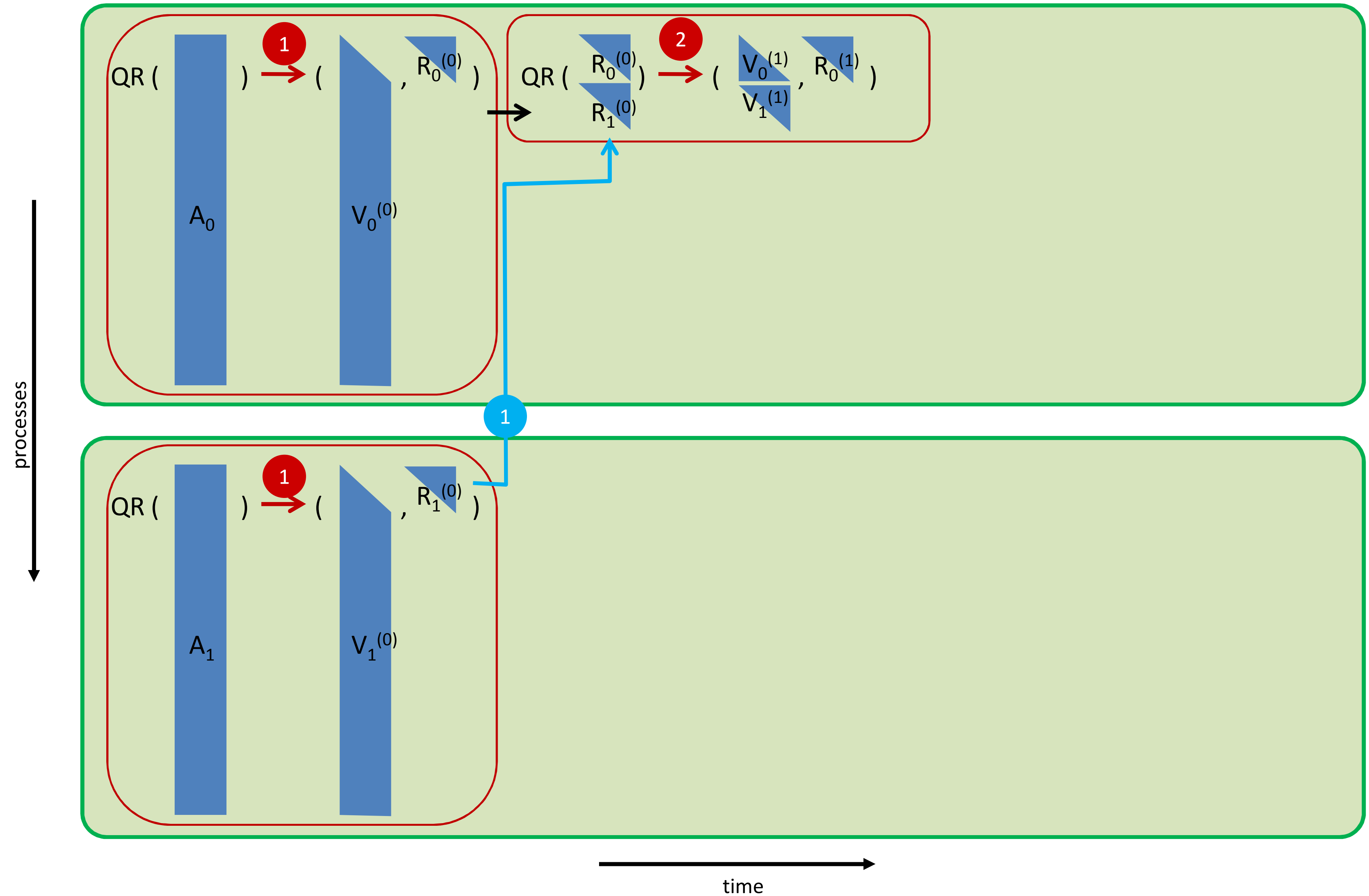
On two processes



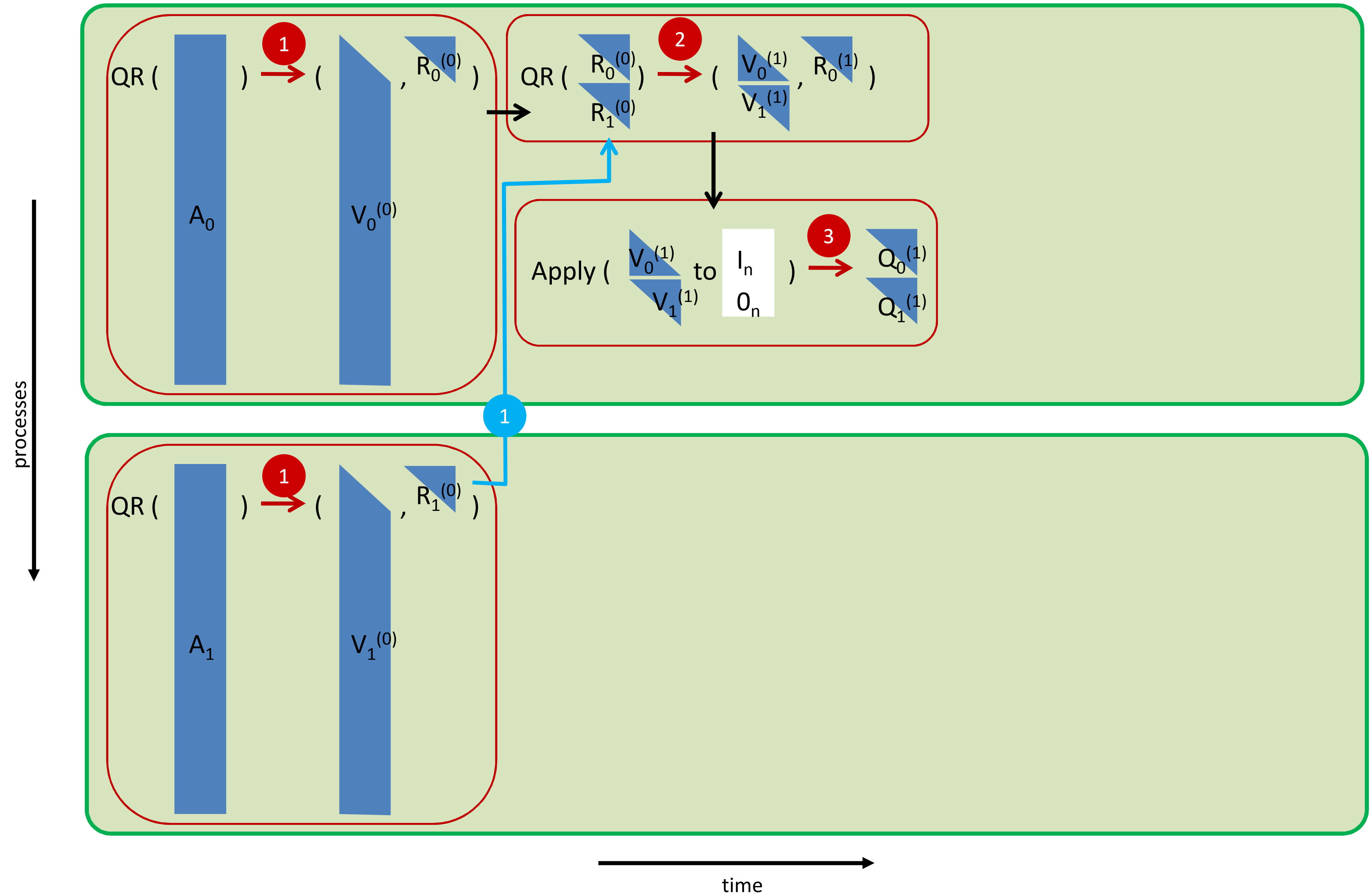
On two processes



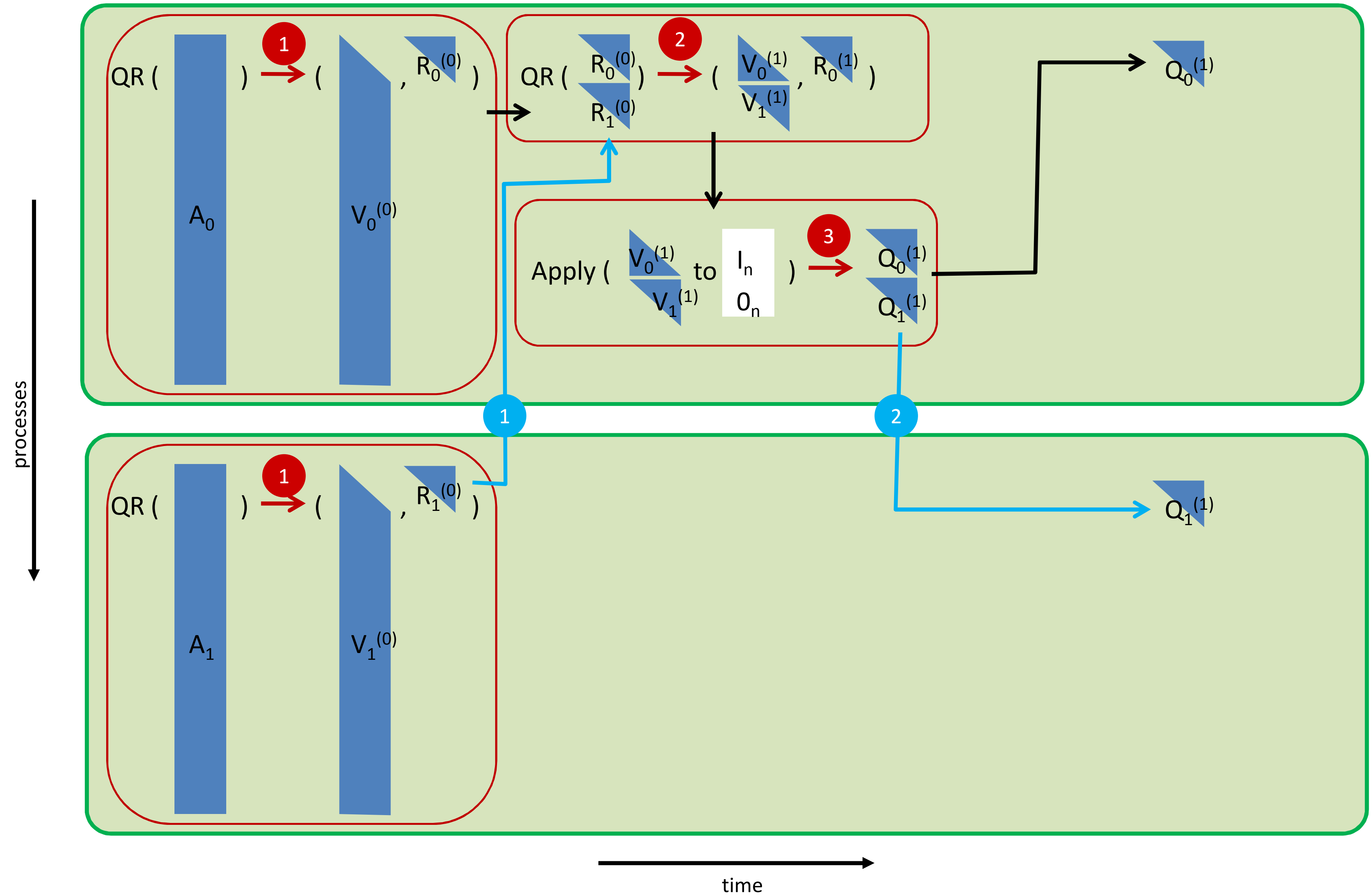
On two processes



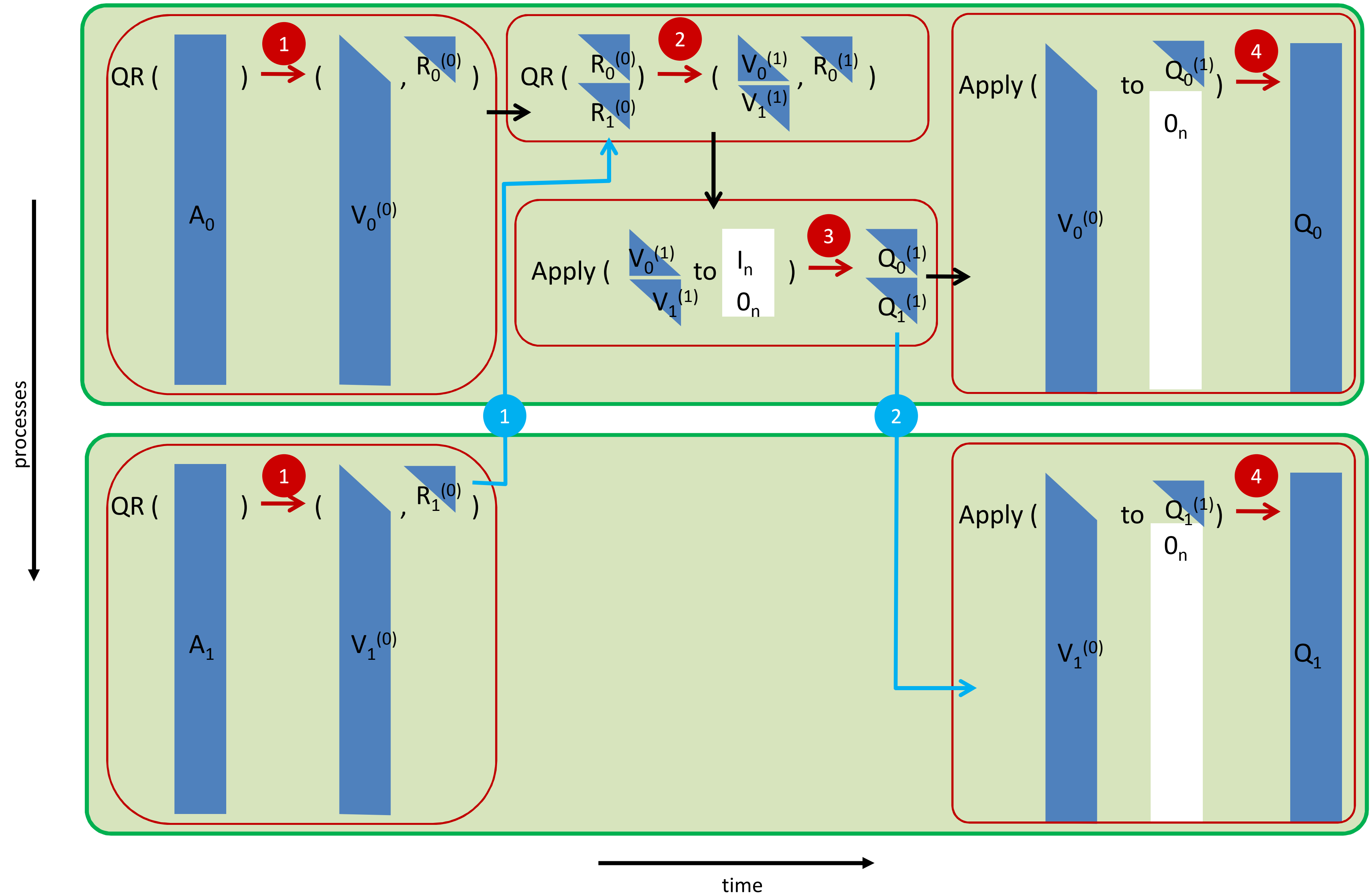
On two processes



On two processes

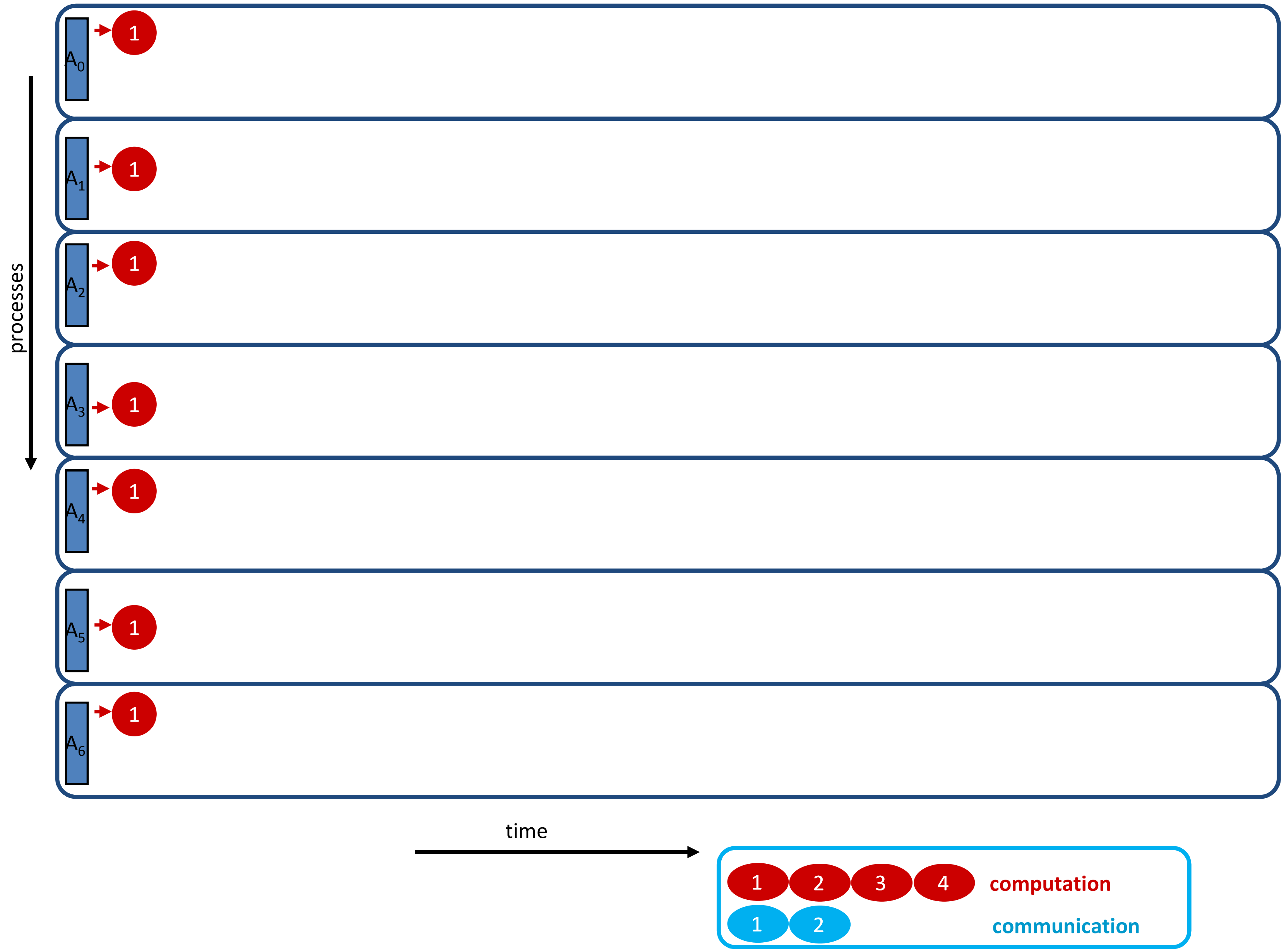


On two processes

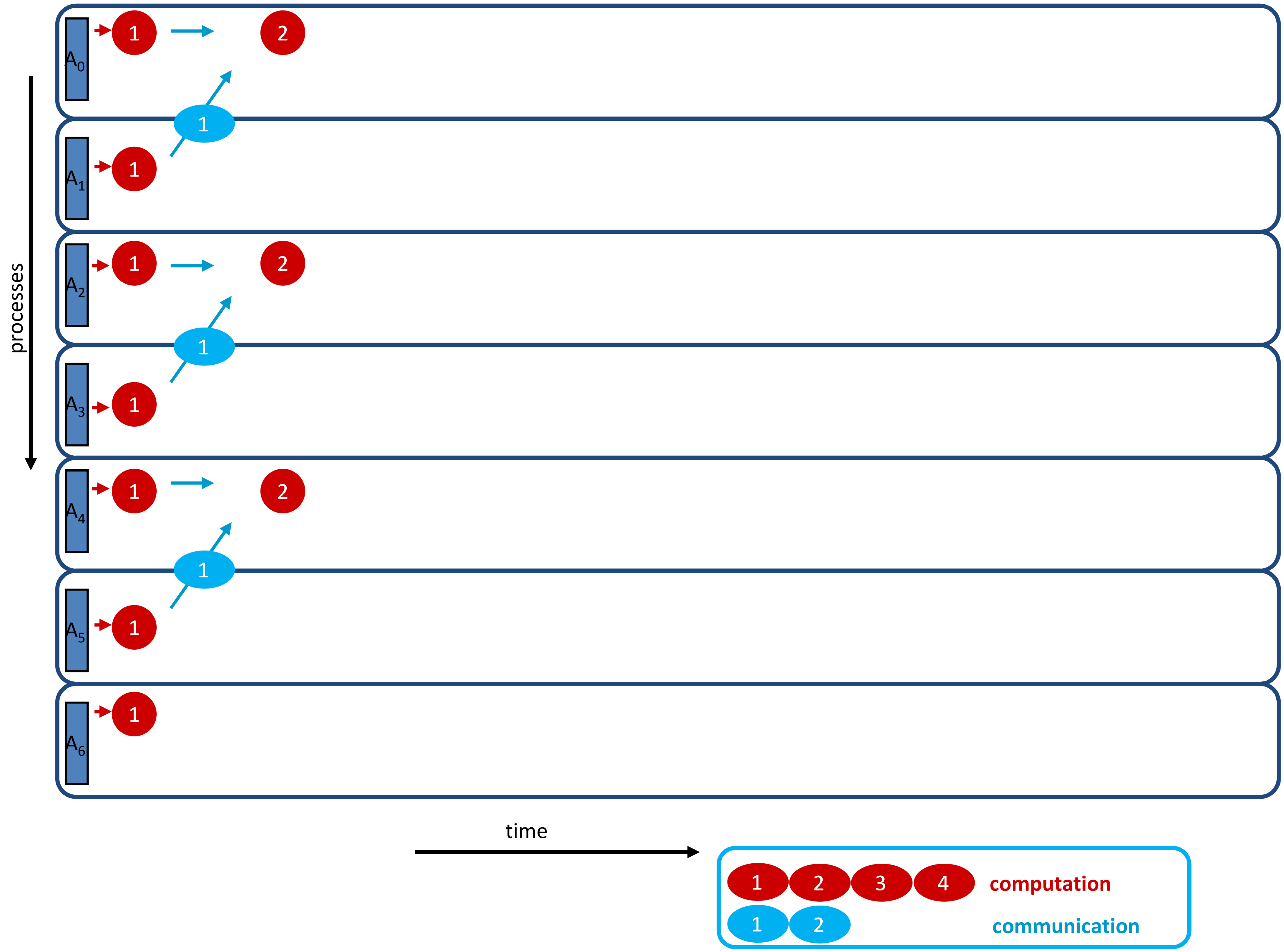


The big picture ….

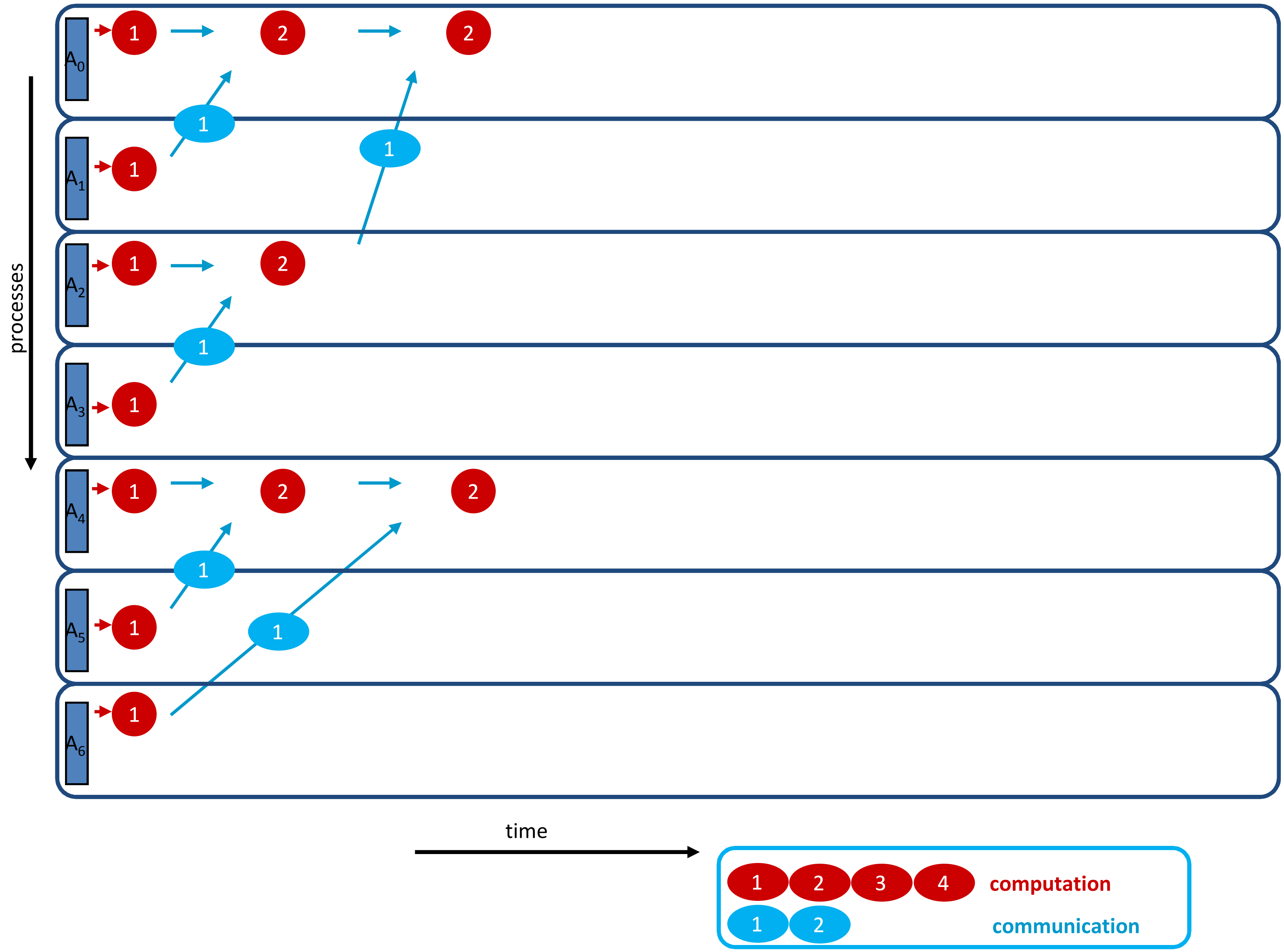
The big picture ….



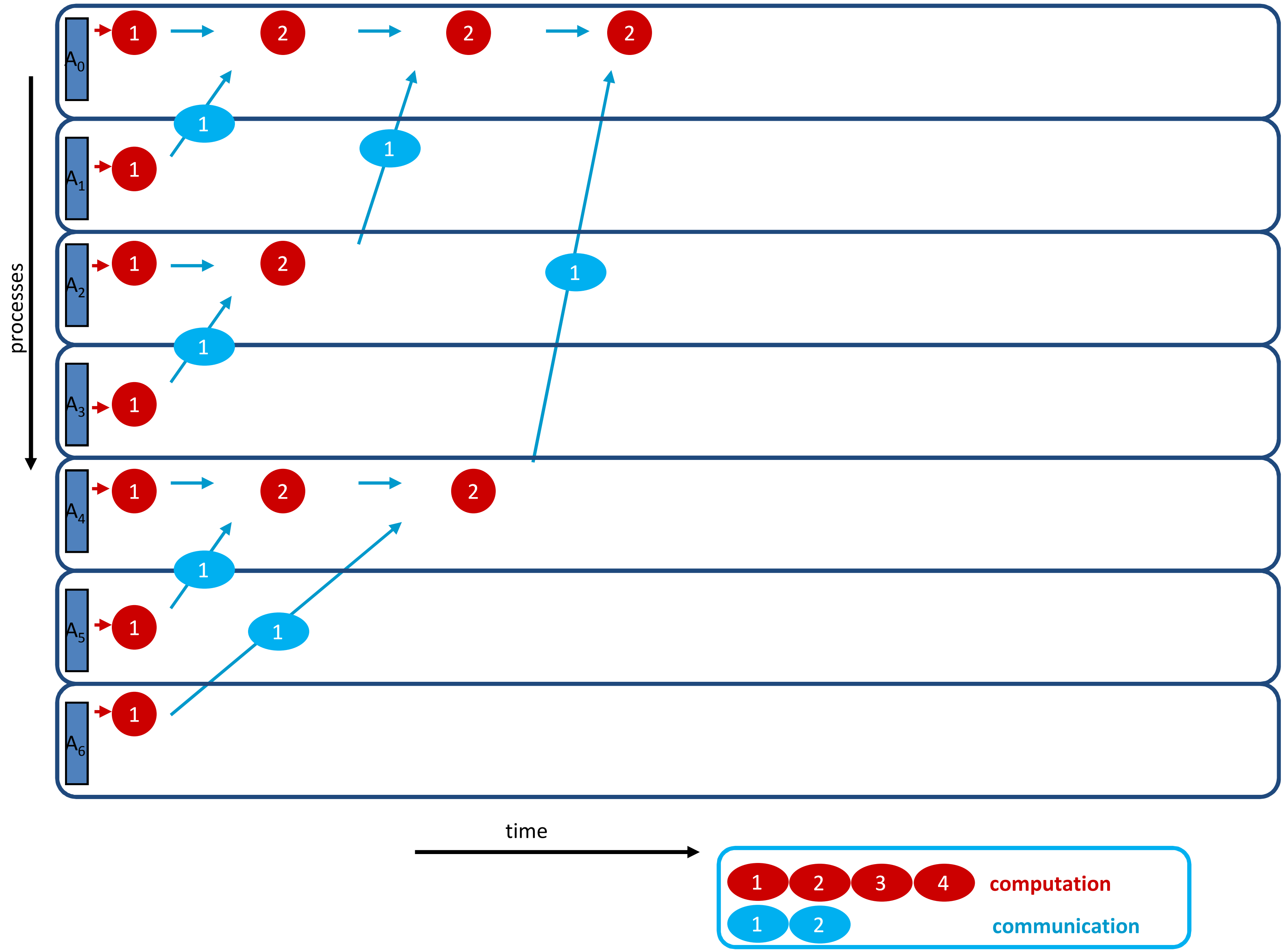
The big picture ….



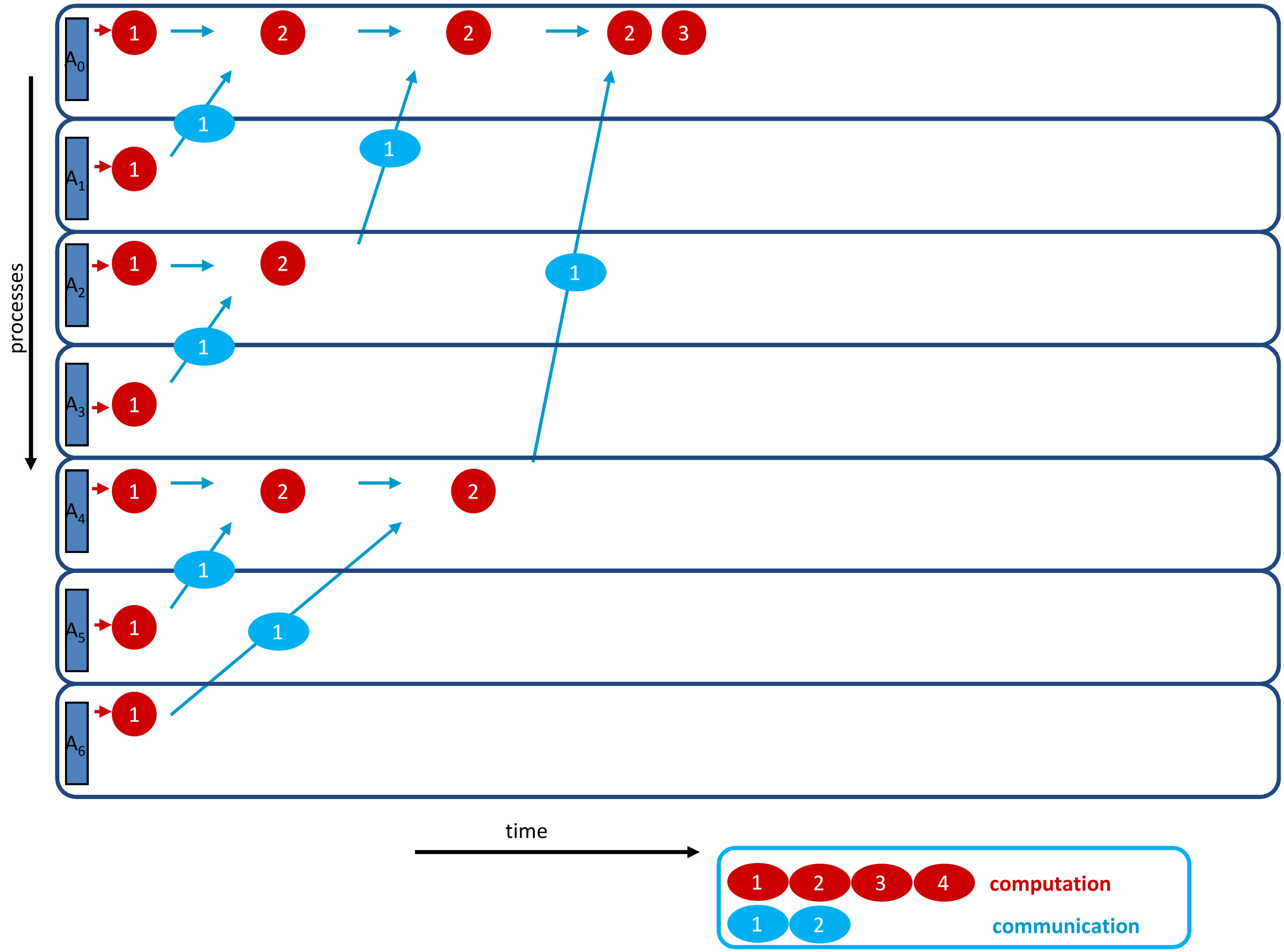
The big picture ….



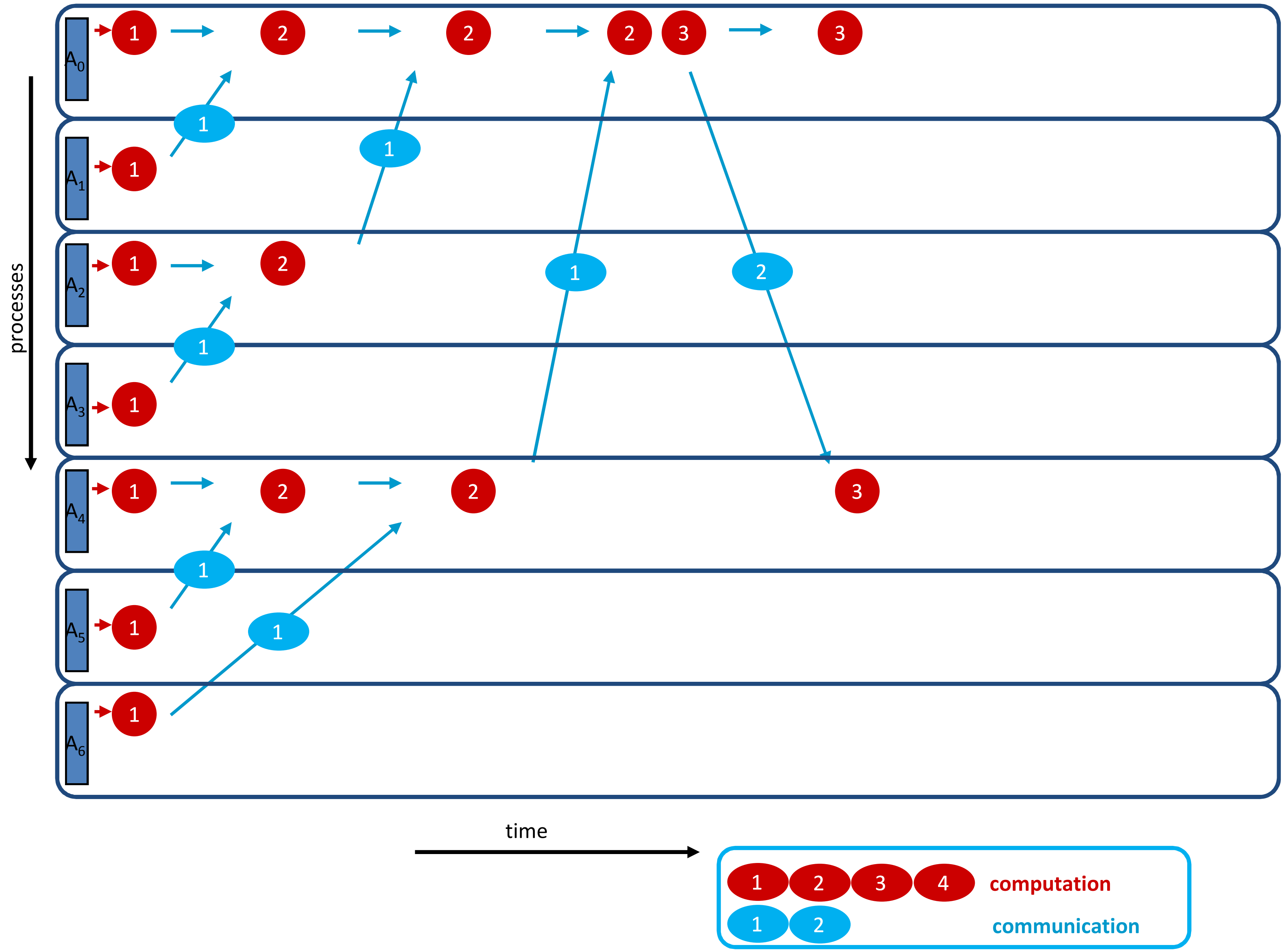
The big picture ....



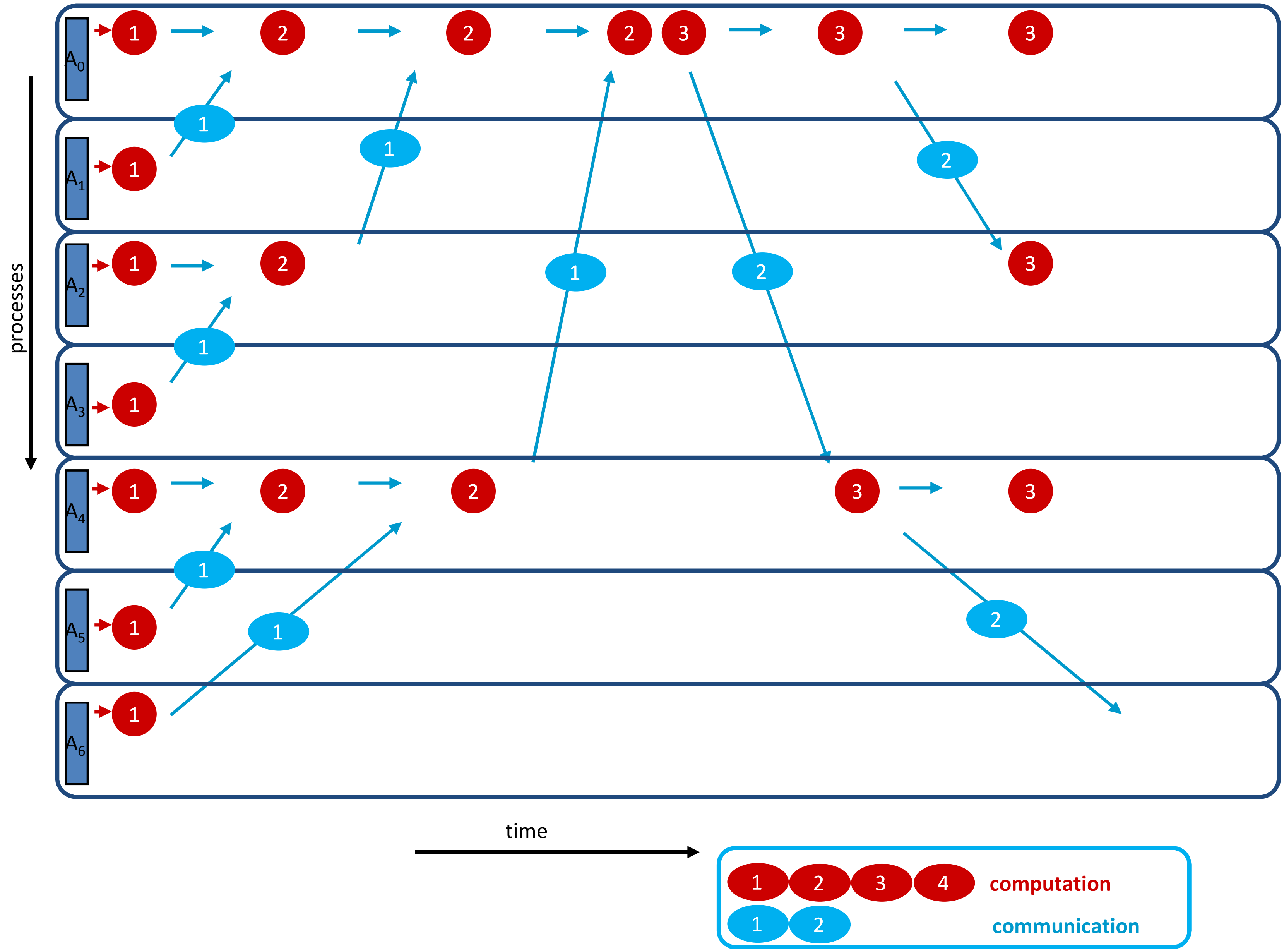
The big picture ….



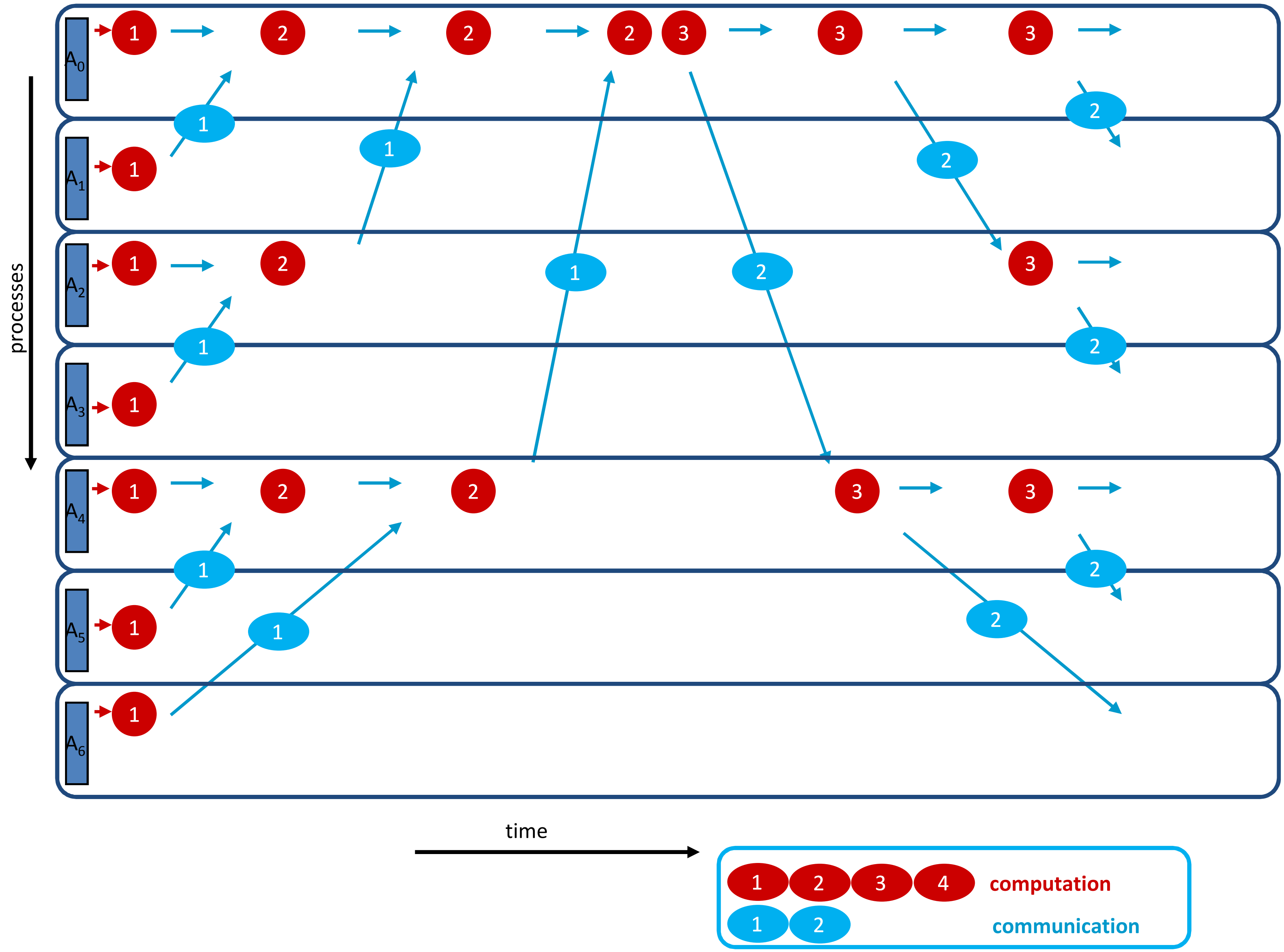
The big picture ….



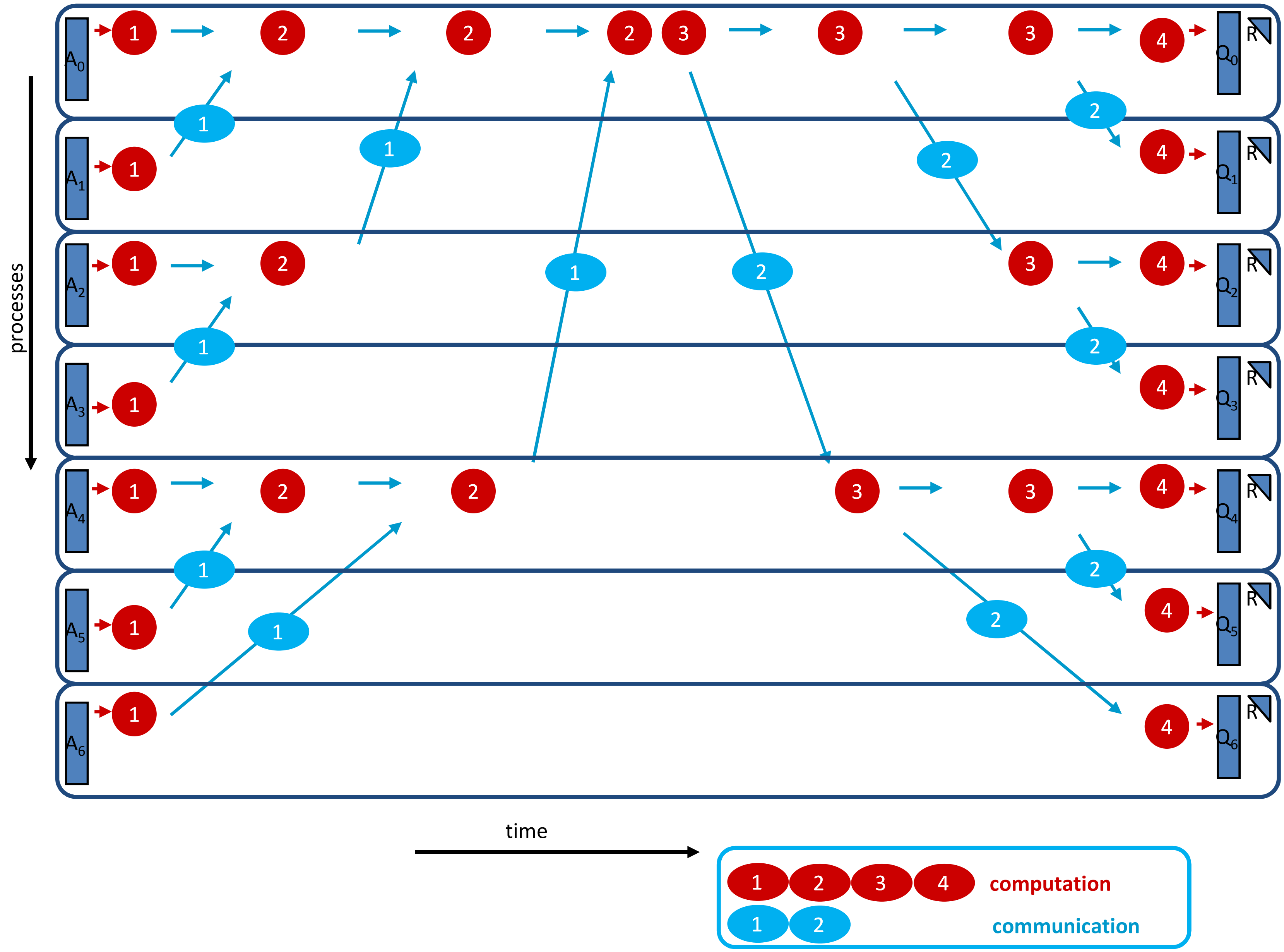
The big picture ….



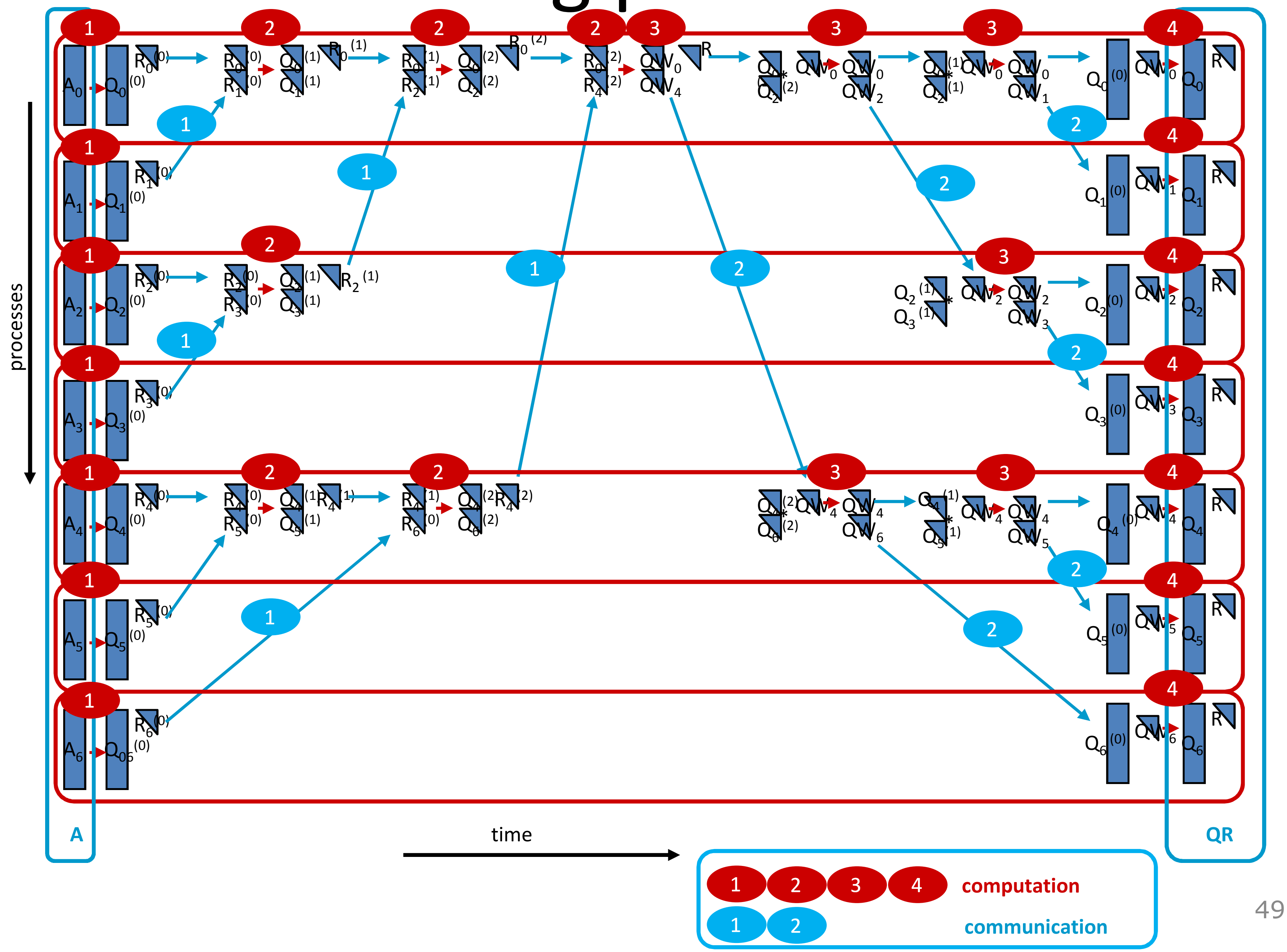
The big picture ….



The big picture ….



The big picture ….



More details

1 The **communication** of type 1 represents a send-receive of one upper triangular matrix ($R_i^{(j)}$).

2 The **communication** of type 2 represents a send-receive of two upper triangular matrices (QW_i and R).

1 The **computation** 1 is done by your favorite QR sequential algorithm, the cost is $(2mn^2 - 4/3\ n^3)/p$ per processors.

2 The **computation** of 2 using a standard QR algorithm would be $10/3\ n^3$; using the fact that $R_j^{(i)}$ $R_k^{(i)}$ are both upper triangular, we can reduce the cost to $2/3\ n^3$.

3 The **computation** of 3 using a standard QR algorithm would be $6\ n^3$; using the fact that $Q_j^{(i)}$, $Q_k^{(i)}$, QW_j, and QW_k are all upper triangular, we can reduce the cost to $2/3\ n^3$.

4 The **computation** 4 is made using a tuned DORMQR the cost is roughly $2mn^2$.

```c
int LILA qr uppers (int n, double *R1, double *R2, double *tau,
        double *work ){
/*
 * The cost of this operation is 2/3 n^3 to compare with
 * 10/3n^3 (=2mn^2-2/3*n^3, with m=2n) using a standard
  * Householder code
 *
 * We exploit the fact that:
 * - the two matrices R1 and R2 are triangular
 * - the matrix H is lower triangular
 * The cost comes mainly from step (j.2): 2*(n-j)*j and (j.4): 2*(n-j)*j
 * that you integrate from j=1:n.
 *
 * Purpose
 * =======
 * Consider the (2N)-by-N matrix:
 * W = [ R1 ]
 * [ R2 ]
 *
 * LILA qr uppers performs the QR factorization of W.
 *
 * The output are stored in
 * TAU, the scalars to apply the Householder transformation
 * for further use
 * R2, the upper triangular matrix that holds the Householder
 * vectors. They are represented as:
 * [ I ]
 * [ R2 ]
 * R1, the upper triangular matrix that holds the R factor
 */
```

```c
int j;
for (j=1;j<n;j++){

        lapack_dlarfg( j+2, &(R1[j*n+j]),
                &(R2[j*n]), 1, &(tau[j]));

        if ((j<n-1)&&(tau[j] != 0.0e+00)){
/*
 *              w := R2(1:j,j+1:n)' * v(1:j) + R1(j,j+1:n)
 */
                cblas_dgemv( CblasColMajor,  CblasTrans,
                        j+1, n-j-1, 1.0e+00,
                        &(R2[(j+1)*n]), n,
                        &(R2[j*n]), 1,
                        0.0e+00, work, 1 );

                cblas_daxpy( n-j-1, 1.0e+00,
                        &(R1[(j+1)*n+j]), n, work, 1);
/*
 *              R1(j,j+1:n) = R1(j,j+1:n) - tau * w
 *              R2(1:j,j+1:n) = R2(1:j,j+1:n) - tau * v(1:j) * w
 */
                cblas_daxpy( n-j-1, tau[j], work, 1,
                        &(R1[(j+1)*n+j]), n);

                cblas_dger( CblasColMajor, j+1, n-j-1,
                        -tau[j], &(R2[j*n]), 1,
                        work, 1, &(R2[(j+1)*n]), n );
                }
        }
        return 0;
}
```

Operations/Latency/Bandwidth

When Only R is needed (R on all the processes)				
	FLOPs (total)	# msg	Vol data exchanged	FLOPs
CholeskyQR	$mn^2 + n^3/3$	$2\log_2(p)$	$2\log_2(p)\ (n^2/2)$	$(mn^2)/p + n^3/3$
Gram-Schmidt	$2mn^2$	$2n \log_2(p)$	$2\log_2(p)\ (n^2/2)$	$(2mn^2)/p$
Householder	$2mn^2 - 2/3n^3$	$2n \log_2(p)$	$2\log_2(p)\ (n^2/2)$	$(2mn^2 - 2/3n^3)/p$
Allreduce HH	$(2mn^2 - 2/3n^3)$ $+2/3\ n^3\ p$	$2\log_2(p)$	$2\log_2(p)\ (n^2/2)$	$(2mn^2 - 2/3n)/p$ $+2/3\ n^3 \log_2(p)$

Q and R are needed				
	FLOPs (total)	# msg	Vol data exchanged	FLOPs
CholeskyQR	$2mn^2 + n^3/3$	$2\ \log_2(p)$	$2\ \log_2(p)\ (n^2/2)$	$(2mn^2)/p + n^3/3$
Gram-Schmidt	$2mn^2$	$4\ n \log_2(p)$	$4\ \log_2(p)\ (n^2/2)$	$(2mn^2)/p$
Householder	$4mn^2 - 4/3n^3$	$4\ n \log_2(p)$	$4\ \log_2(p)\ (n^2/2)$	$(4mn^2 - 4/3n^3)/p$
Allreduce HH	$(4mn^2 - 4/3n^3)$ $+4/3\ n^3\ p$	$2\ \log_2(p)$	$2\ \log_2(p)\ (n^2/2)$	$(4mn^2 - 4/3n^3)/p$ $+4/3\ n^3 \log_2(p)$

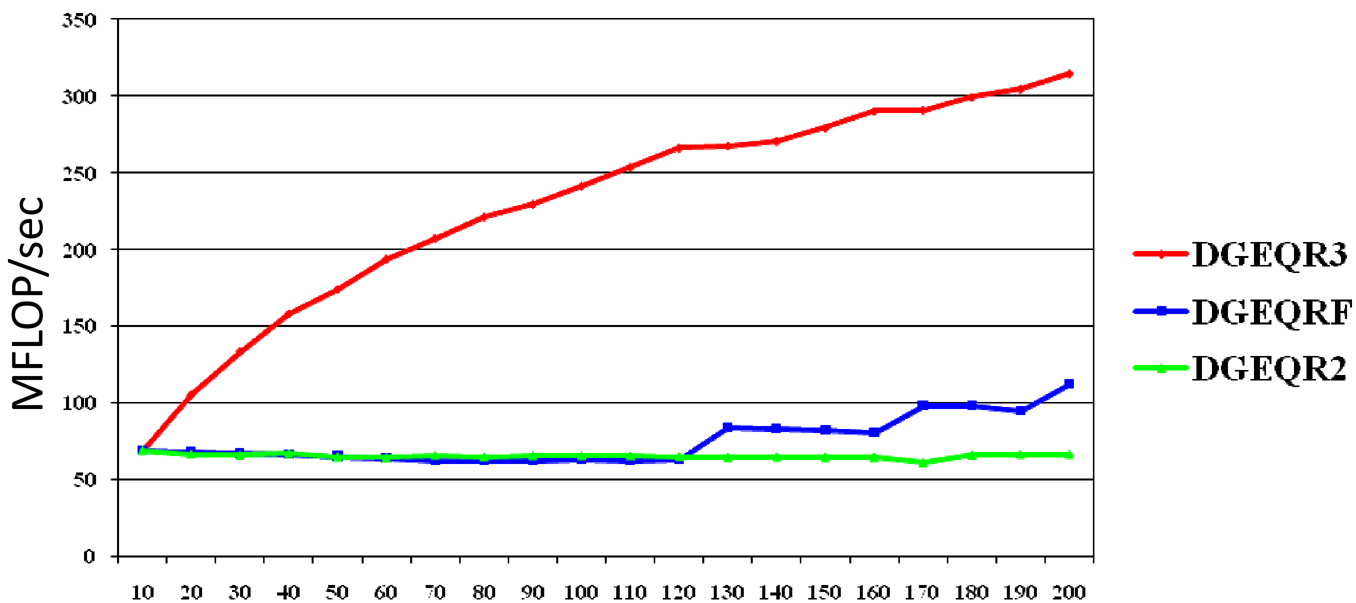
The total time is

α * (# msg) + β * (vol data exchanged) + γ * (FLOPs)

Latency but also possibility of fast panel factorization.

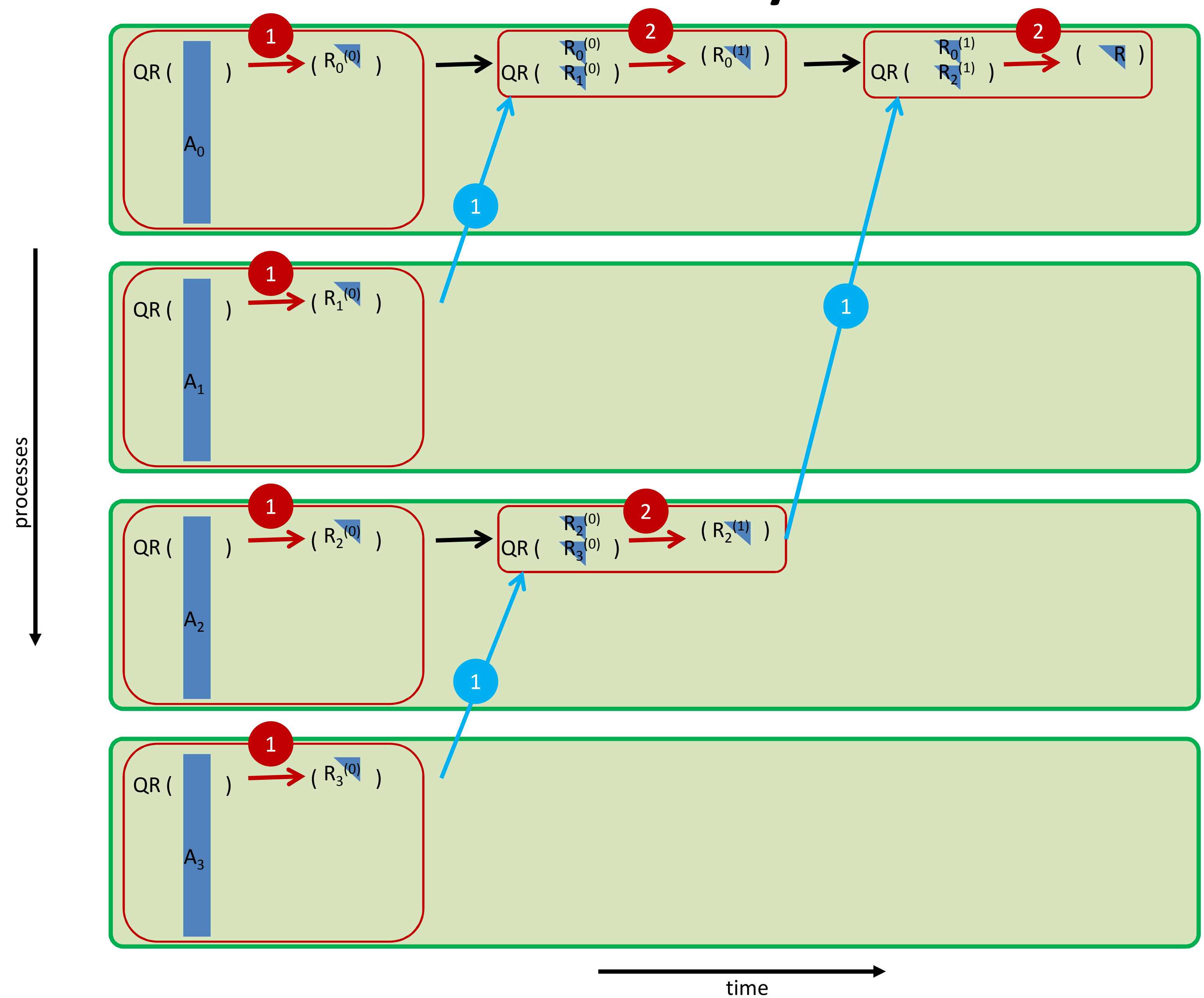
- **DGEQR3** is the recursive algorithm (see Elmroth and Gustavson, 2000), **DGEQRF** and **DGEQR2** are the LAPACK routines.
- Times include QR and DLARFT.
- Run on Pentium III.

QR factorization and construction of T m = 100,000 Perf in MFLOP/sec (Times in sec)						
n	DGEQR3		DGEQRF		DGEQR2	
50	173.6	(0.29)	65.0	(0.77)	64.6	(0.77)
100	240.5	(0.83)	62.6	(3.17)	65.3	(3.04)
150	277.9	(1.60)	81.6	(5.46)	64.2	(6.94)
200	312.5	(2.53)	111.3	(7.09)	65.9	(11.98)



m=100,000, the x axis is n

When only R is wanted



When only R is wanted: The MPI_Allreduce

In the case where only R is wanted, instead of constructing our own tree, one can simply use MPI_Allreduce with a user defined operation. The operation we give to MPI is basically the Algorithm 2. It performs the operation:

$$QR\begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \longrightarrow R$$

This **binary** operation is **associative** and this is all MPI needs to use a user-defined operation on a user-defined datatype. Moreover, if we change the signs of the elements of R so that the diagonal of R holds positive elements then the binary operation **`Rfactor`** becomes **commutative**.

The code becomes two lines:

```
lapack_dgeqrf( mloc, n, A, lda, tau, &dlwork, lwork, &info );
MPI_Allreduce( MPI_IN_PLACE, A, 1, MPI_UPPER,
        LILA_MPIOP_QR_UPPER, mpi_comm);
```

Does it work?

Does it work?

- The experiments are performed on the beowulf cluster at the University of Colorado at Denver. The cluster is made of 35 bi-pro Pentium III (900MHz) connected with Dolphin interconnect.
- Number of operations is taken as $2mn^2$ for all the methods
- The block size used in ScaLAPACK is 32.
- The code is written in C, use MPI (mpich-2.1), LAPACK (3.1.1), BLAS (goto-1.10), the LAPACK Cwrappers (http://icl.cs.utk.edu/~delmas/lapwrapmw.htm) and the BLAS C wrappers (http://www.netlib.org/blas/blast-forum/cblas.tgz)
- The codes has been tested in various configuration and have never failed to produce a correct answer, releasing those codes is in the agenda

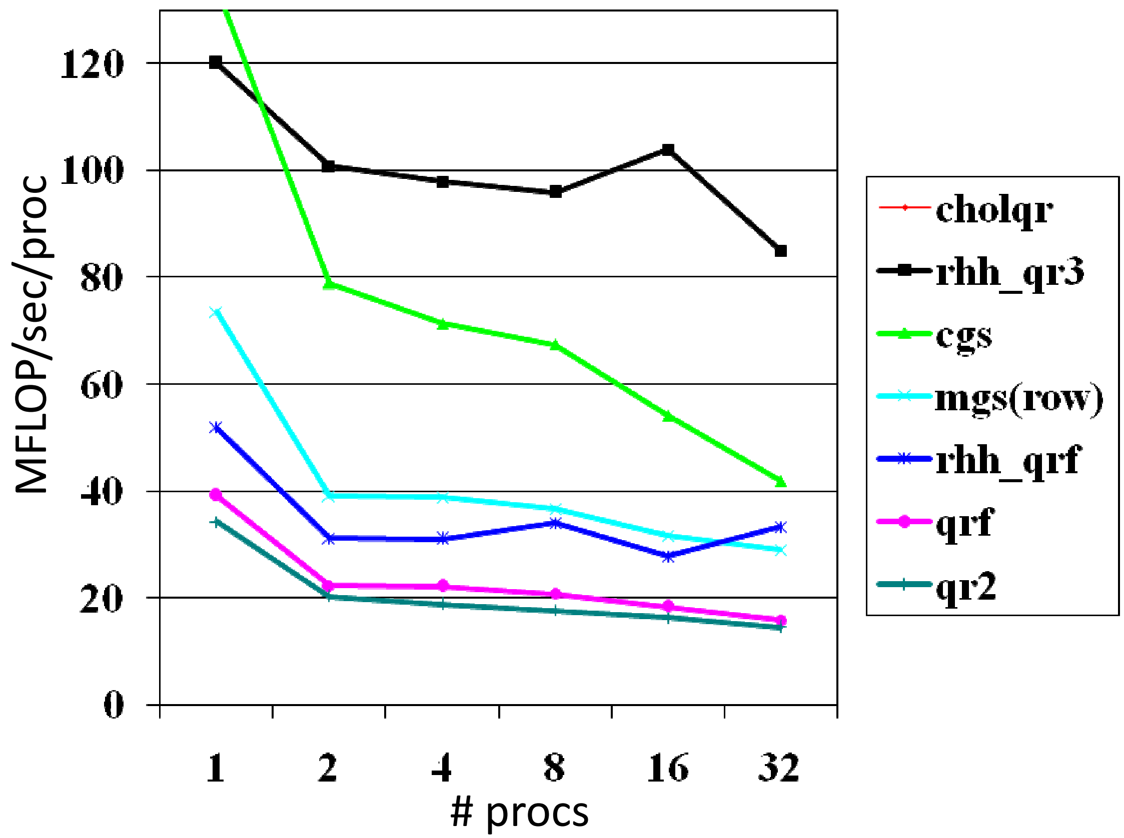
CAUTION! **Number of operations is taken as $2mn^2$ for all the methods** CAUTION!

	FLOPs (total) for R only	FLOPs (total) for Q and R
CholeskyQR	$mn^2 + n^3/3$	$2mn^2 + n^3/3$
Gram-Schmidt	$2mn^2$	$2mn^2$
Householder	$2mn^2 - 2/3n^3$	$4mn^2 - 4/3n^3$
Allreduce HH	$(2mn^2 - 2/3n^3) + 2/3\ n^3\ p$	$(4mn^2 - 4/3n^3) + 4/3\ n^3\ p$

Q and R: Strong scalability

- In this experiment, we fix the problem: **m=100,000** and **n=50**. Then we increase the number of processors.

- Once more the algorithm **rhh_qr3** is the second behind **CholeskyQR**. Note that **rhh_qr3** is incondionnally stable while the stability of **CholeskyQR** depends on the square of the condition number of the initial matrix.



MFLOP/sec/proc

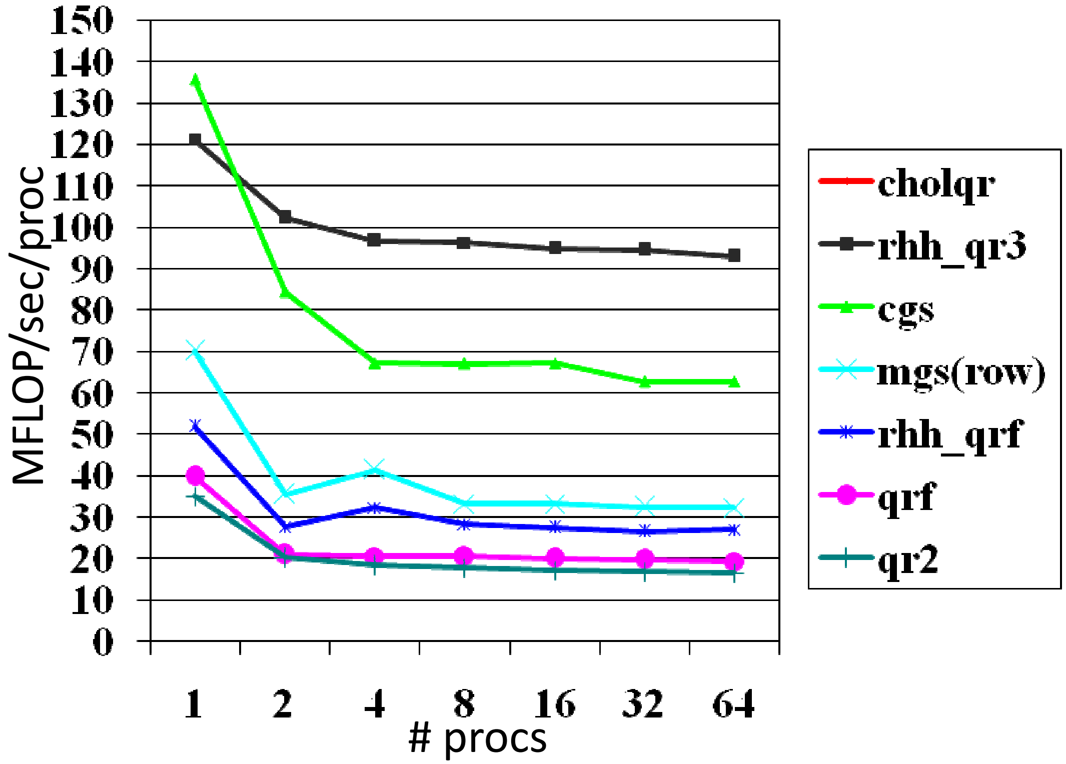
Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	489.2	(1.02)	120.0	(4.17)	134.1	(3.73)	73.5	(6.81)	51.9	(9.64)	39.1	(12.78)	34.3	(14.60)
2	467.3	(0.54)	100.8	(2.48)	78.9	(3.17)	39.0	(6.41)	31.2	(8.02)	22.3	(11.21)	20.2	(12.53)
4	466.4	(0.27)	97.9	(1.28)	71.3	(1.75)	38.7	(3.23)	31.0	(4.03)	22.2	(5.63)	18.8	(6.66)
8	434.0	(0.14)	95.9	(0.65)	67.4	(0.93)	36.7	(1.70)	34.0	(1.84)	20.8	(3.01)	17.7	(3.54)
16	359.2	(0.09)	103.8	(0.30)	54.2	(0.58)	31.6	(0.99)	27.8	(1.12)	18.3	(1.71)	16.3	(1.91)
32	197.8	(0.08)	84.9	(0.18)	41.9	(0.37)	29.0	(0.54)	33.3	(0.47)	15.8	(0.99)	14.5	(1.08)

Q and R: Weak scalability with respect to m

- We fix the local size to be **mloc=100,000** and **n=50**. When we increase the number of processors, the global m grows proportionally.

- **rhh_qr3** is the Allreduce algorithm with recursive panel factorization, **rhh_qrf** is the same with LAPACK Householder QR. We see the obvious benefit of using recursion. See as well (6). **qr2** and **qrf** correspond to the ScaLAPACK Householder QR factorization routines.



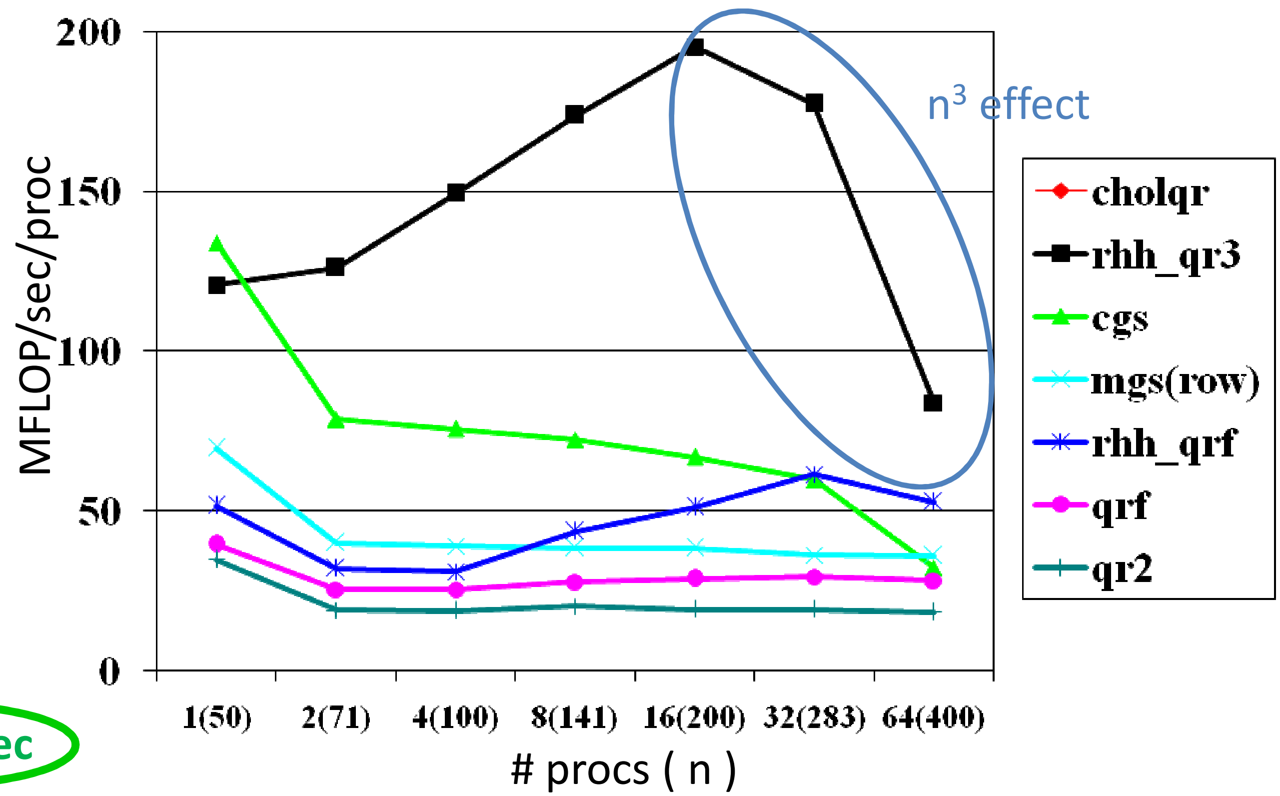
MFLOP/sec/proc Time in sec

# of procs	cholqr		rhh_qr3		Cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	489.2	(1.02)	121.2	(4.13)	135.7	(3.69)	70.2	(7.13)	51.9	(9.64)	39.8	(12.56)	35.1	(14.23)
2	466.9	(1.07)	102.3	(4.89)	84.4	(5.93)	35.6	(14.04)	27.7	(18.06)	20.9	(23.87)	20.2	(24.80)
4	454.1	(1.10)	96.7	(5.17)	67.2	(7.44)	41.4	(12.09)	32.3	(15.48)	20.6	(24.28)	18.3	(27.29)
8	458.7	(1.09)	96.2	(5.20)	67.1	(7.46)	33.2	(15.06)	28.3	(17.67)	20.5	(24.43)	17.8	(28.07)
16	451.3	(1.11)	94.8	(5.27)	67.2	(7.45)	33.3	(15.04)	27.4	(18.22)	20.0	(24.95)	17.2	(29.10)
32	442.1	(1.13)	94.6	(5.29)	62.8	(7.97)	32.5	(15.38)	26.5	(18.84)	19.8	(25.27)	16.9	(29.61)
64	414.9	(1.21)	93.0	(5.38)	62.8	(7.96)	32.3	(15.46)	27.0	(18.53)	19.4	(25.79)	16.6	(30.13)

Q and R: Weak scalability with respect to n

- We fix the global size **m=100,000** and then we increase n as sqrt(p) so that the workload mn^2 per processor remains constant.

- Due to better performance in the local factorization or SYRK, **CholeskyQR**, **rhh_q3** and **rhh_qrf** exhibit increasing performance at the beginning until the n^3 comes into play

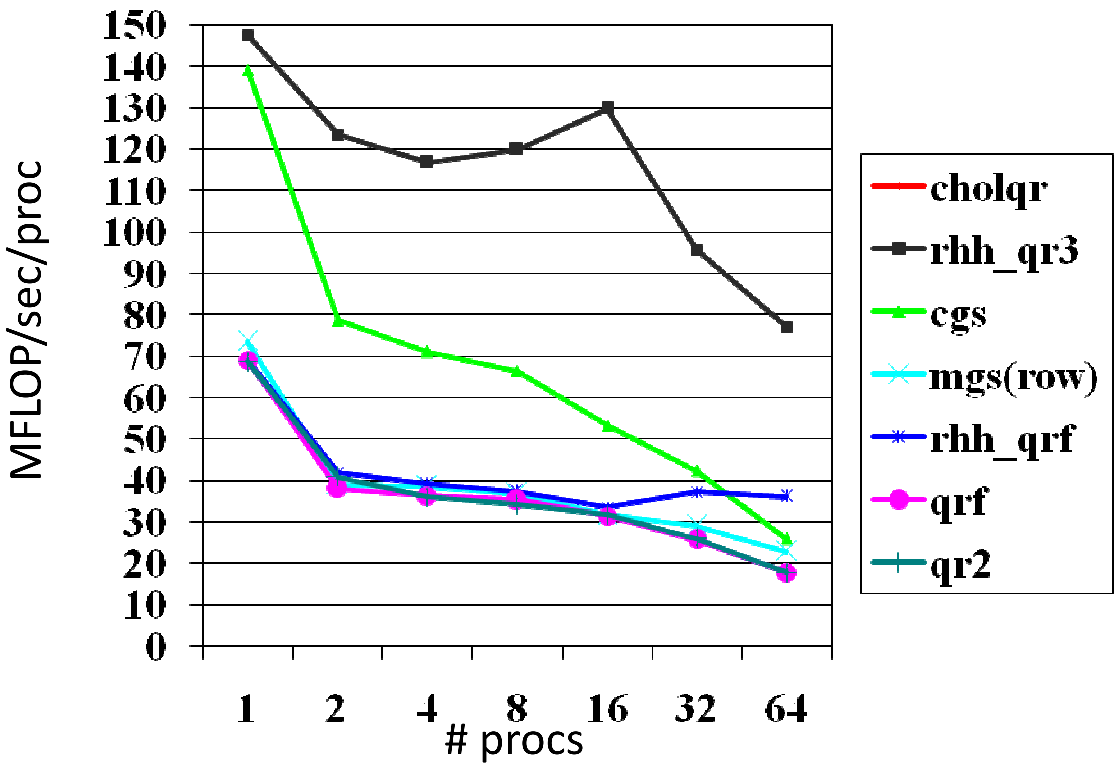


MFLOP/sec/proc Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	490.7	(1.02)	120.8	(4.14)	134.0	(3.73)	69.7	(7.17)	51.7	(9.68)	39.6	(12.63)	39.9	(14.31)
2	510.2	(0.99)	126.0	(4.00)	78.6	(6.41)	40.1	(12.56)	32.1	(15.71)	25.4	(19.88)	19.0	(26.56)
4	541.1	(0.92)	149.4	(3.35)	75.6	(6.62)	39.1	(12.78)	31.1	(16.07)	25.5	(19.59)	18.9	(26.48)
8	540.2	(0.92)	173.8	(2.86)	72.3	(6.87)	38.5	(12.89)	43.6	(11.41)	27.8	(17.85)	20.2	(24.58)
16	501.5	(1.00)	195.2	(2.56)	66.8	(7.48)	38.4	(13.02)	51.3	(9.75)	28.9	(17.29)	19.3	(25.87)
32	379.2	(1.32)	177.4	(2.82)	59.8	(8.37)	36.2	(13.84)	61.4	(8.15)	29.5	(16.95)	19.3	(25.92)
64	266.4	(1.88)	83.9	(5.96)	32.3	(15.46)	36.1	(13.84)	52.9	(9.46)	28.2	(17.74)	18.4	(27.13)

R only: Strong scalability

- In this experiment, we fix the problem: **m=100,000** and **n=50**. Then we increase the number of processors.



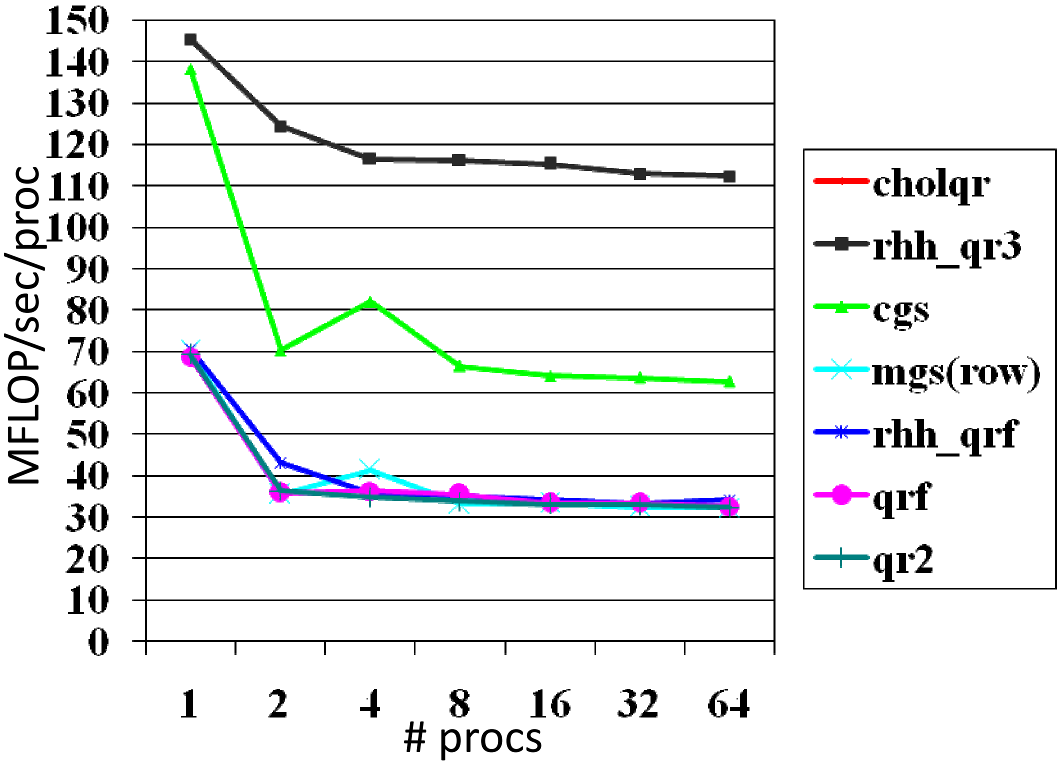
MFLOP/sec/proc

Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	1099.046	(0.45)	147.6	(3.38)	139.309	(3.58)	73.5	(6.81)	69.049	(7.24)	69.108	(7.23)	68.782	(7.27)
2	1067.856	(0.23)	123.424	(2.02)	78.649	(3.17)	39.0	(6.41)	41.837	(5.97)	38.008	(6.57)	40.782	(6.13)
4	1034.203	(0.12)	116.774	(1.07)	71.101	(1.76)	38.7	(3.23)	39.295	(3.18)	36.263	(3.44)	36.046	(3.47)
8	876.724	(0.07)	119.856	(0.52)	66.513	(0.94)	36.7	(1.70)	37.397	(1.67)	35.313	(1.77)	34.081	(1.83)
16	619.02	(0.05)	129.808	(0.24)	53.352	(0.59)	31.6	(0.99)	33.581	(0.93)	31.339	(0.99)	31.697	(0.98)
32	468.332	(0.03)	95.607	(0.16)	42.276	(0.37)	29.0	(0.54)	37.226	(0.42)	25.695	(0.60)	25.971	(0.60)
64	195.885	(0.04)	77.084	(0.10)	25.89	(0.30)	22.8	(0.34)	36.126	(0.22)	17.746	(0.44)	17.725	(0.44)

R only: Weak scalability with respect to m

- We fix the local size to be **mloc=100,000** and **n=50**. When we increase the number of processors, the global m grows proportionally.



MFLOP/sec/proc Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	1098.7	(0.45)	145.4	(3.43)	138.2	(3.61)	70.2	(7.13)	70.6	(7.07)	68.7	(7.26)	69.1	(7.22)
2	1048.3	(0.47)	124.3	(4.02)	70.3	(7.11)	35.6	(14.04)	43.1	(11.59)	35.8	(13.95)	36.3	(13.76)
4	1044.0	(0.47)	116.5	(4.29)	82.0	(6.09)	41.4	(12.09)	35.8	(13.94)	36.3	(13.74)	34.7	(14.40)
8	993.9	(0.50)	116.2	(4.30)	66.3	(7.53)	33.2	(15.06)	35.1	(14.21)	35.5	(14.05)	33.8	(14.75)
16	918.7	(0.54)	115.2	(4.33)	64.1	(7.79)	33.3	(15.04)	34.0	(14.66)	33.4	(14.94)	33.0	(15.11)
32	950.7	(0.52)	112.9	(4.42)	63.6	(7.85)	32.5	(15.38)	33.4	(14.95)	33.3	(15.01)	32.9	(15.19)
64	764.6	(0.65)	112.3	(4.45)	62.7	(7.96)	32.3	(15.46)	34.0	(14.66)	32.6	(15.33)	32.3	(15.46)

References.

E. Chu and A. George.
QR factorization of a dense matrix on a shared memory multiprocessor,
Parallel Comput., 11:55-71, 1989.

A. Pothen and P. Raghavan.
Distributed orthogonal factorization: Givens and Householder algorithms,
SIAM J. Sci. Stat. Comput., 10(6):1113-1134, 1989.

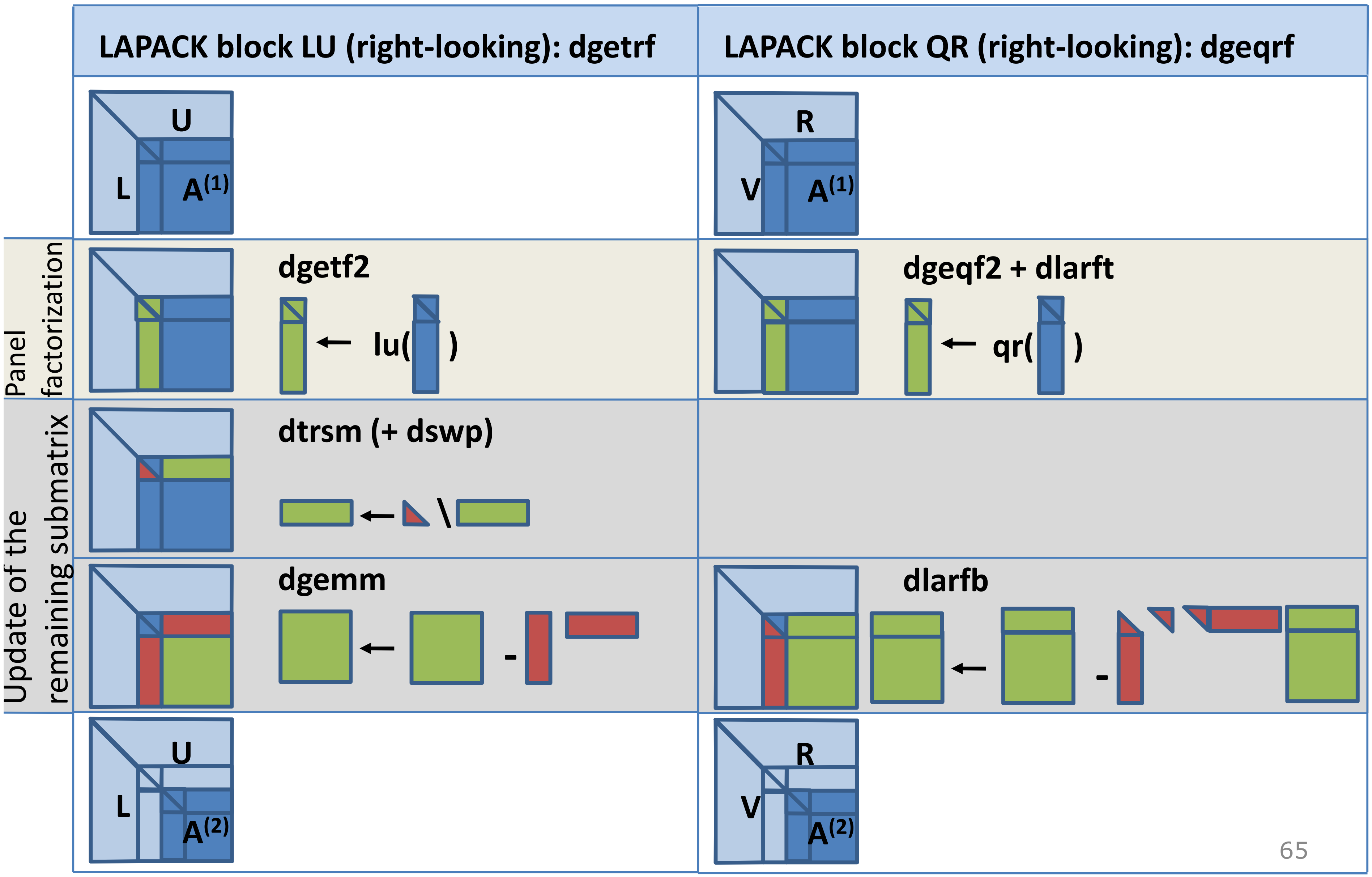
R. Dias da Cunha, D. Becker and James Carlton Patterson.
New parallel (rank-revealing) QR factorization algorithms
In the *Proceedings of Euro-Par 2002*.

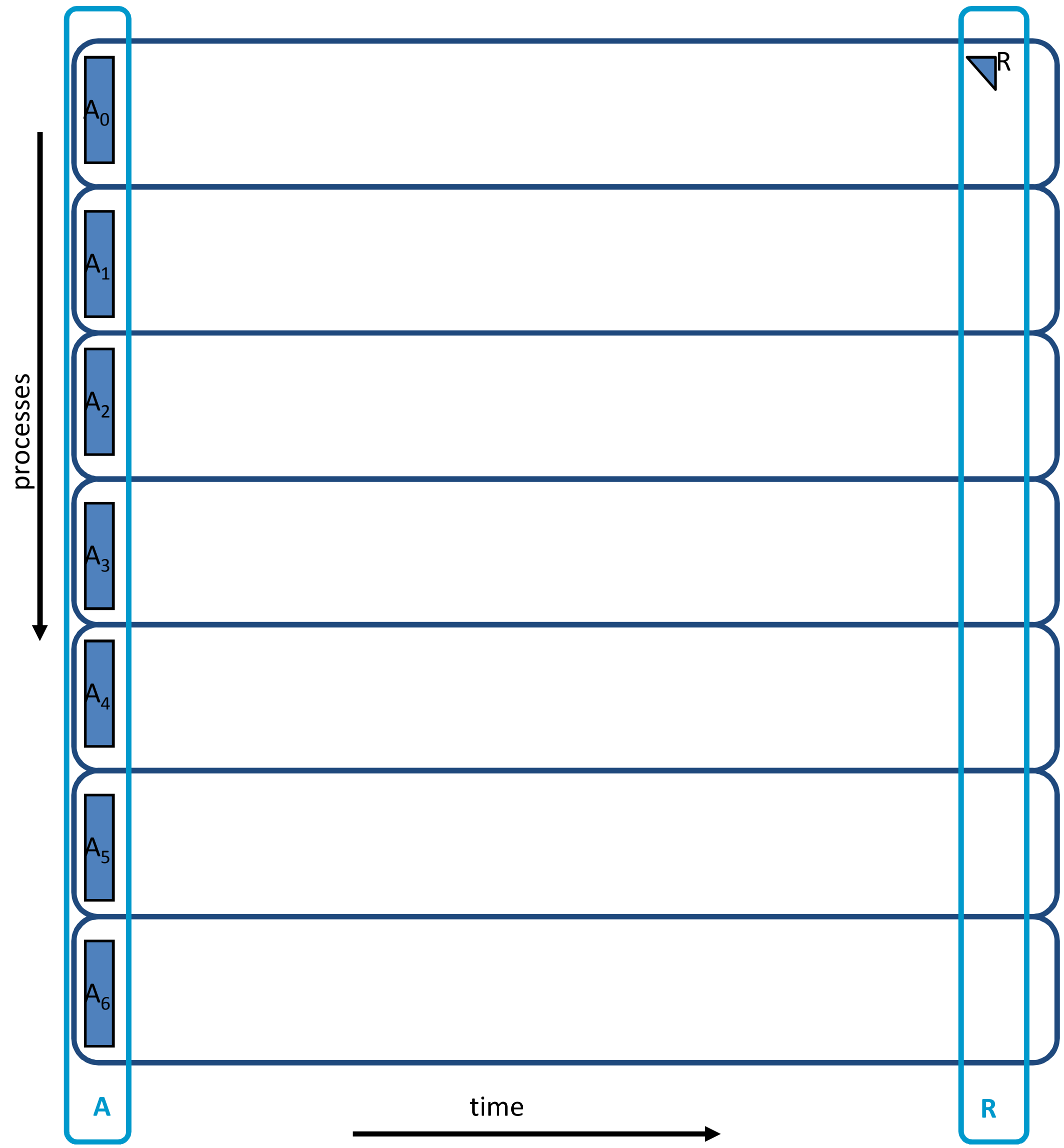
B. Gunter and R. van de Geijn.
Parallel Out-of-Core Computation and Updating of the QR Factorization,
ACM Transactions on Mathematical Software, 31(1):60-78, 2005.

AllReduce Algorithms

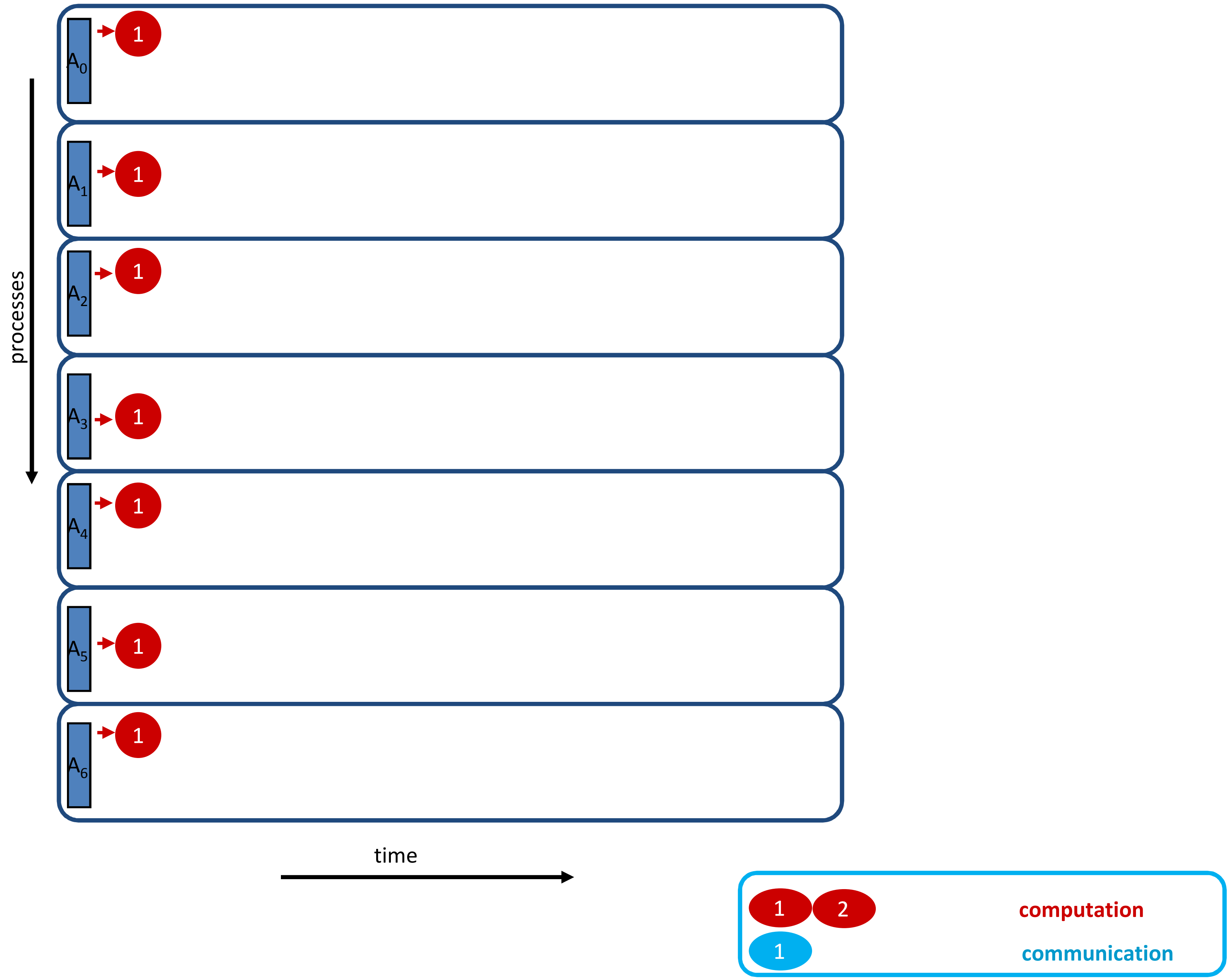
1) Tall Skinny matrices: Application
2) The CholeskyQR algorithm (see MATH6664)
3) AllReduce Householder factorization
4) **Application to dense LU and dense QR factorizations**

Example of applications: panel factorization of dense blocked factorization

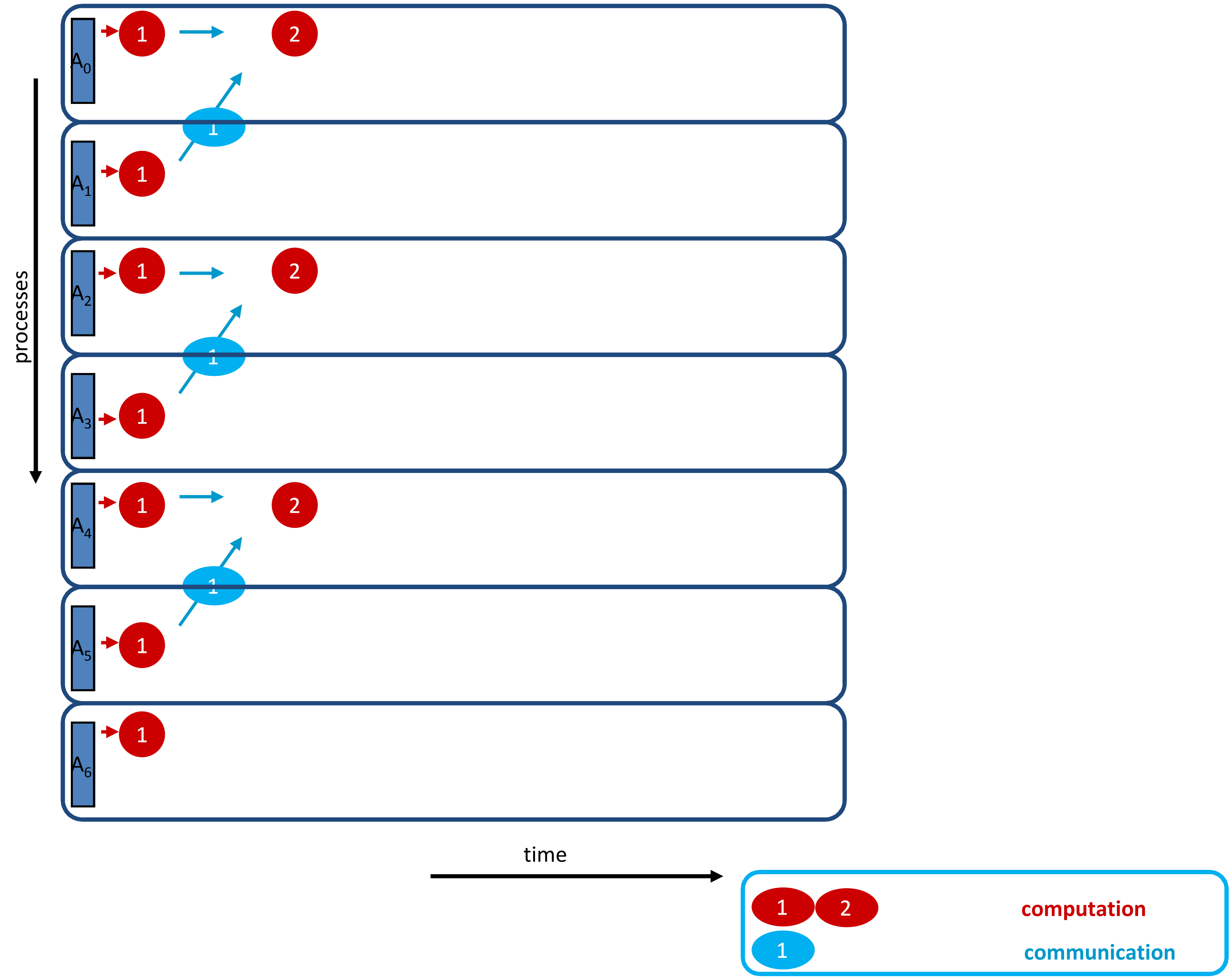




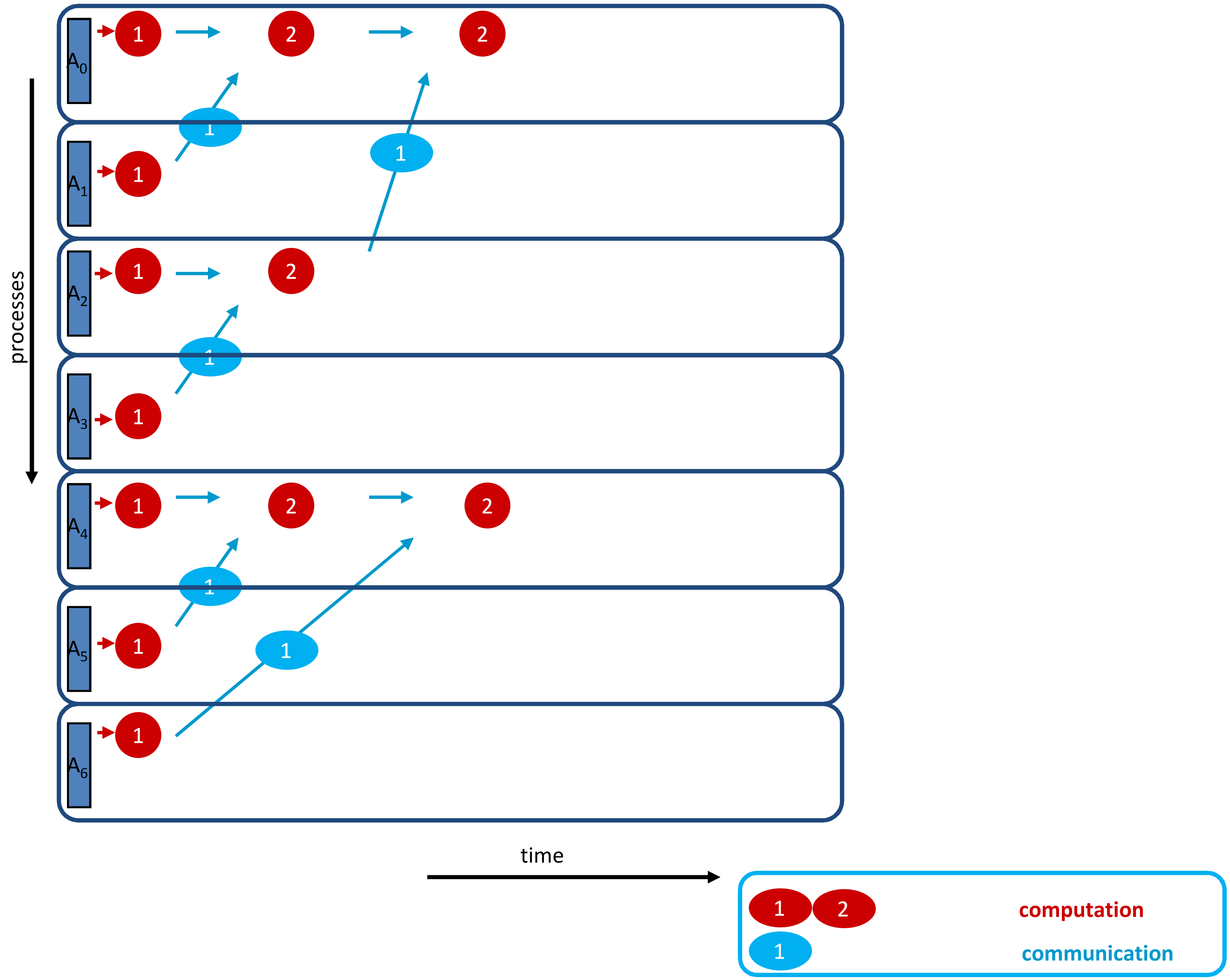
R
A_0
A_1
A_2
A_3
A_4
A_5
A_6
processes
A
time
R



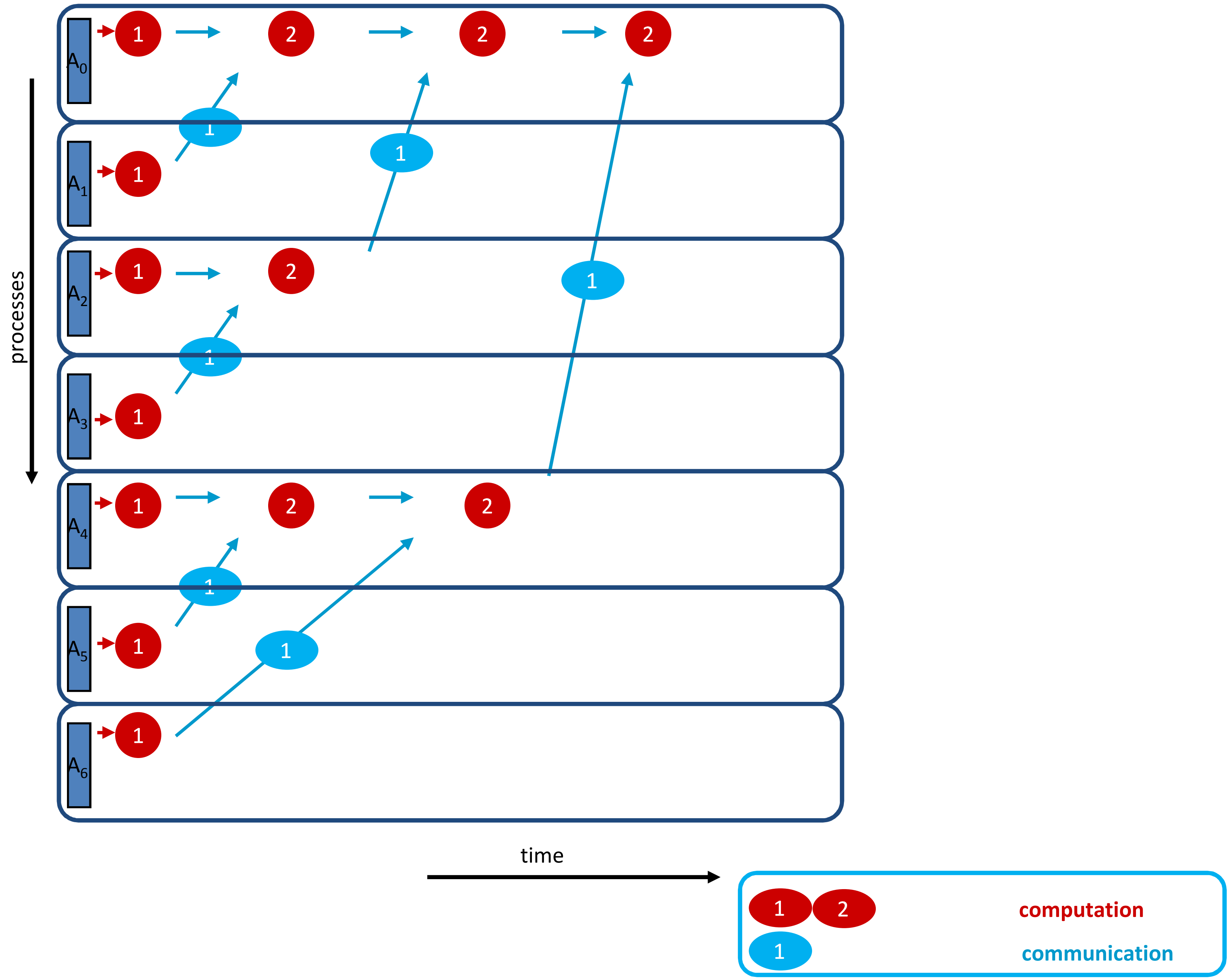
A0
A1
A2
A3
A4
A5
A6
1
processes
time
1 2 computation
1 communication



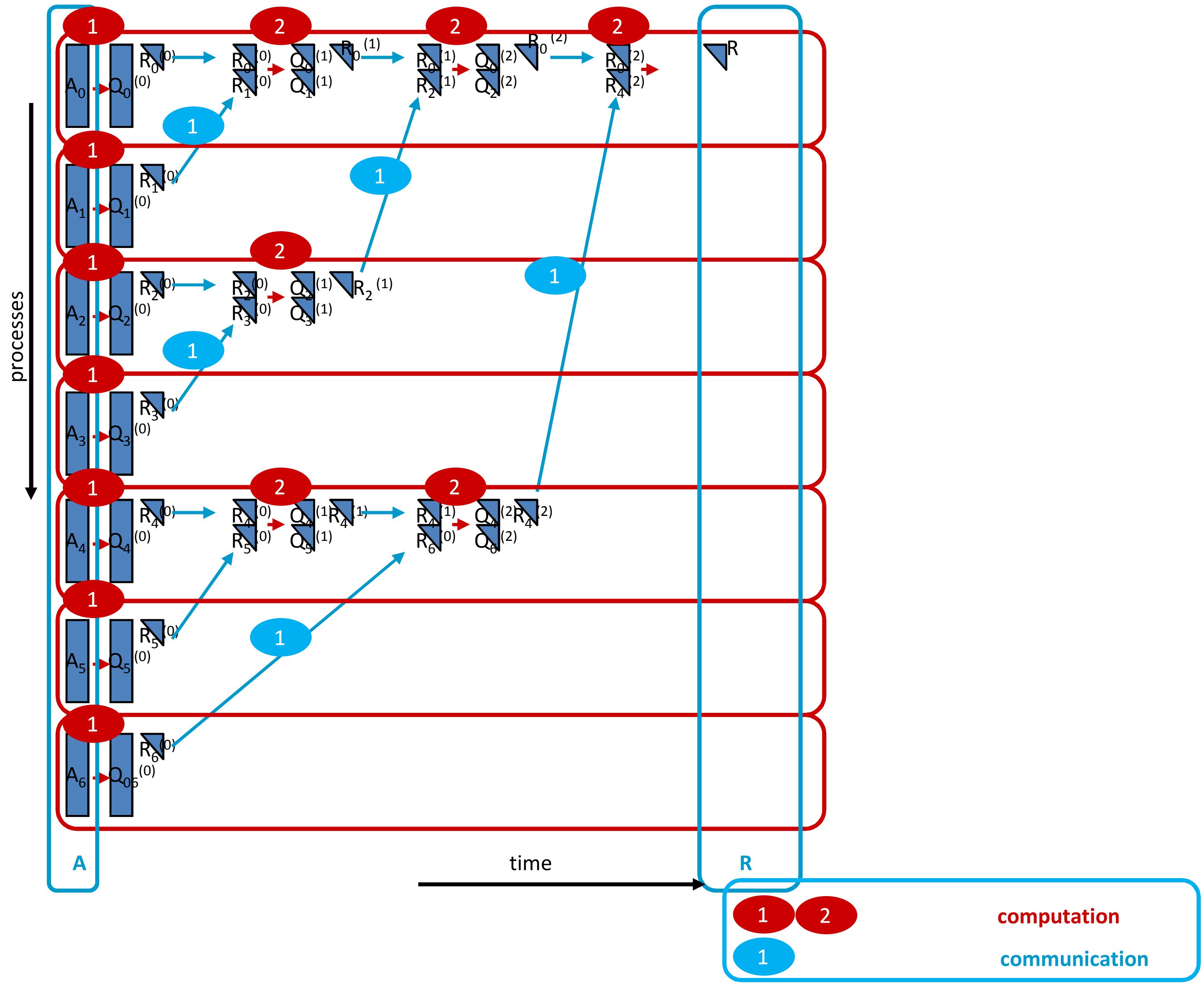
A0
A1
A2
A3
A4
A5
A6
processes
time
computation
communication



processes
A0
A1
A2
A3
A4
A5
A6
time
computation
communication



A0
1
2
2
2
A1
1
1
1
A2
1
2
1
1
A3
1
A4
1
2
2
1
A5
1
1
A6
1
processes
time
1
2
computation
1
communication


processes
time
A
R
computation
communication

Conclusions

We have described a new method for the Householder QR factorization of skinny matrices. The method is named **Allreduce Householder** and has four advantages:

1. there is **only one synchronization point** in the algorithm,
2. the method **harvests most of efficiency of the computing unit** by large local operations,
3. the method is **stable**,
4. and finally the method is **elegant** in particular in the case where only R is needed.

Allreduce algorithms have been depicted here with Householder QR factorization. However it can be applied to *anything* for example Gram-Schmidt or LU.

Current development is in writing a 2D block cyclic QR factorization and LU factorization based on those ideas.