VIII Международная молодежная научно-практическая школа
“Высокопроизводительные вычисления на GRID системах”

Будущее вычислений: от мобильных платформ до экзафлопсных суперкомпьютерных систем

Вл.В.Воеводин

чл.-корр. РАН, профессор,

Заместитель директора НИВЦ МГУ,

Заведующий кафедрой Суперкомпьютеров и квантовой информатики ВМК МГУ

voevodin@parallel.ru

6-е февраля, 2017, САФУ, Архангельск

VIII International Youth Scientific School
"High-Performance Computing Using GRID Systems"

Future Computing: from Mobile Platforms to Exascale Supercomputing Systems

Prof. Vladimir Voevodin
Deputy Director, Research Computing Center, MSU
Head of Department on Supercomputers and Quantum Informatics, CMC, MSU

voevodin@parallel.ru

February 6th, 2017, NARFU, Arkhangelsk

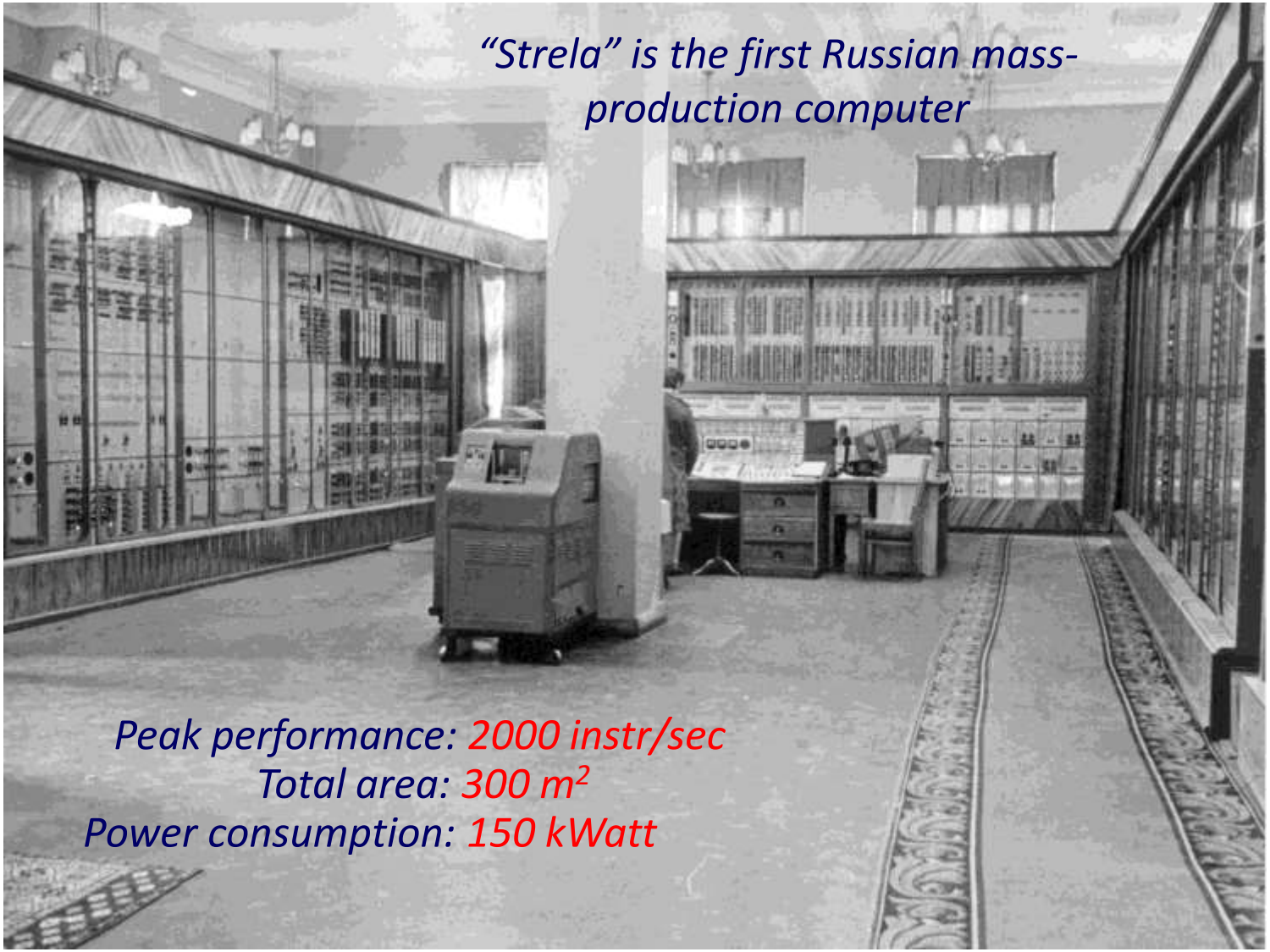
Lomonosov Moscow State University

(established in 1755)

41 faculties
350+ departments
5 major research institutes

45 000+ students,
2500 full doctors (Dr.Sci.), 6000 PhDs,
1000+ full professors,
5000 researchers.

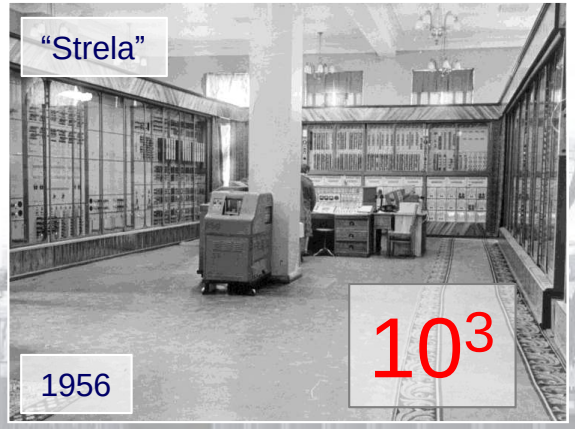
MSU Computing Center, 1956

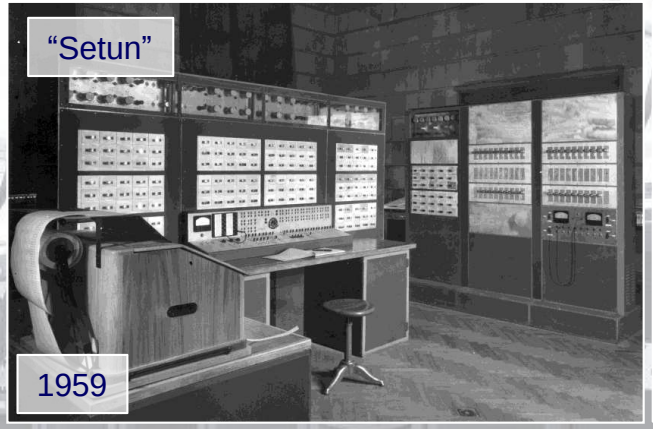


MSU Computing Center, 1959

Computing History of Moscow State University

(from 1956 up to now)





Computing History of Moscow State University

(from 1956 up to now)

Today the key statement in "computing area" is "Building Up Parallel Computing World". Why ?

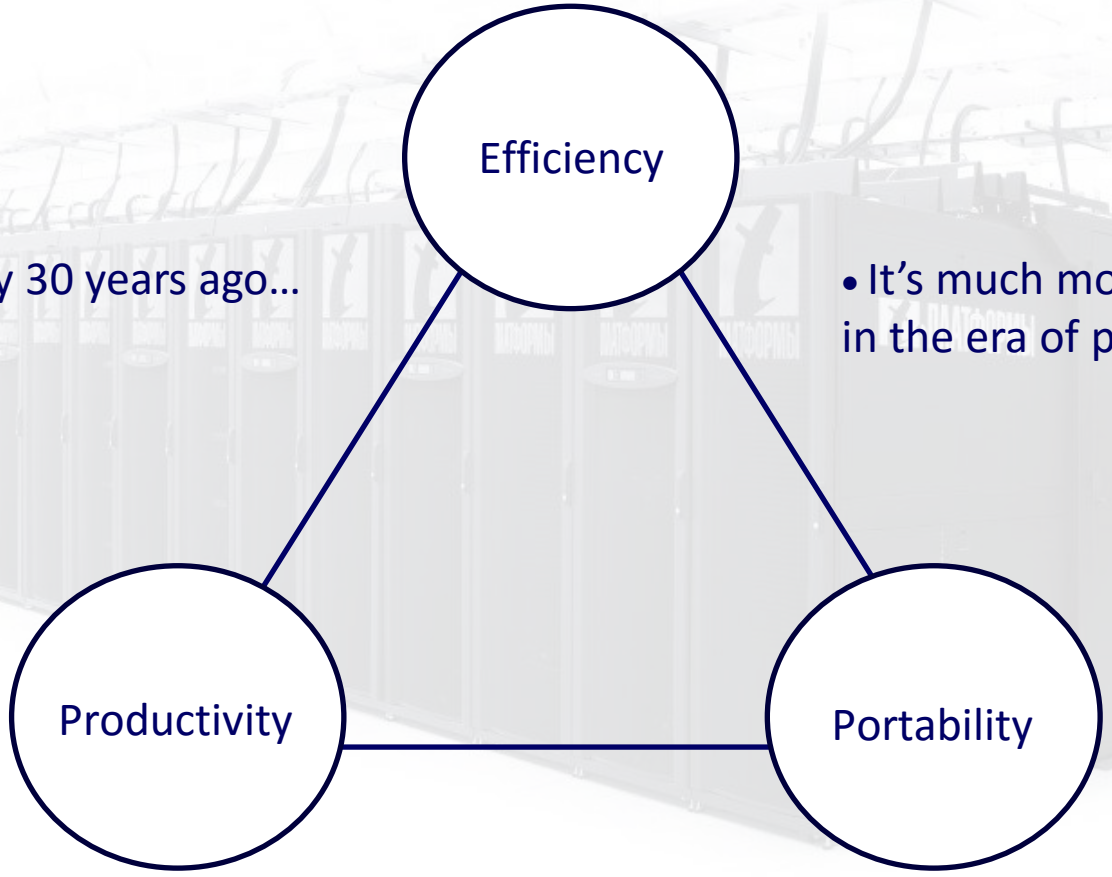
There are a number of reasons…


Computer World
Supercomputers
Servers…
PCs, Laptops…
Tablets, Smartphones…


Degree of parallelism (+/- 8 years)
2009
2017
2025
10^5
10^7
10^9
Supercomputers
2-8
12-64
10^4
Servers…
1
4-8
10^3
PCs, Laptops…
1
1-4
10^2
Tablets, Smartphones…

Software Development Process

(We always make a compromise…)



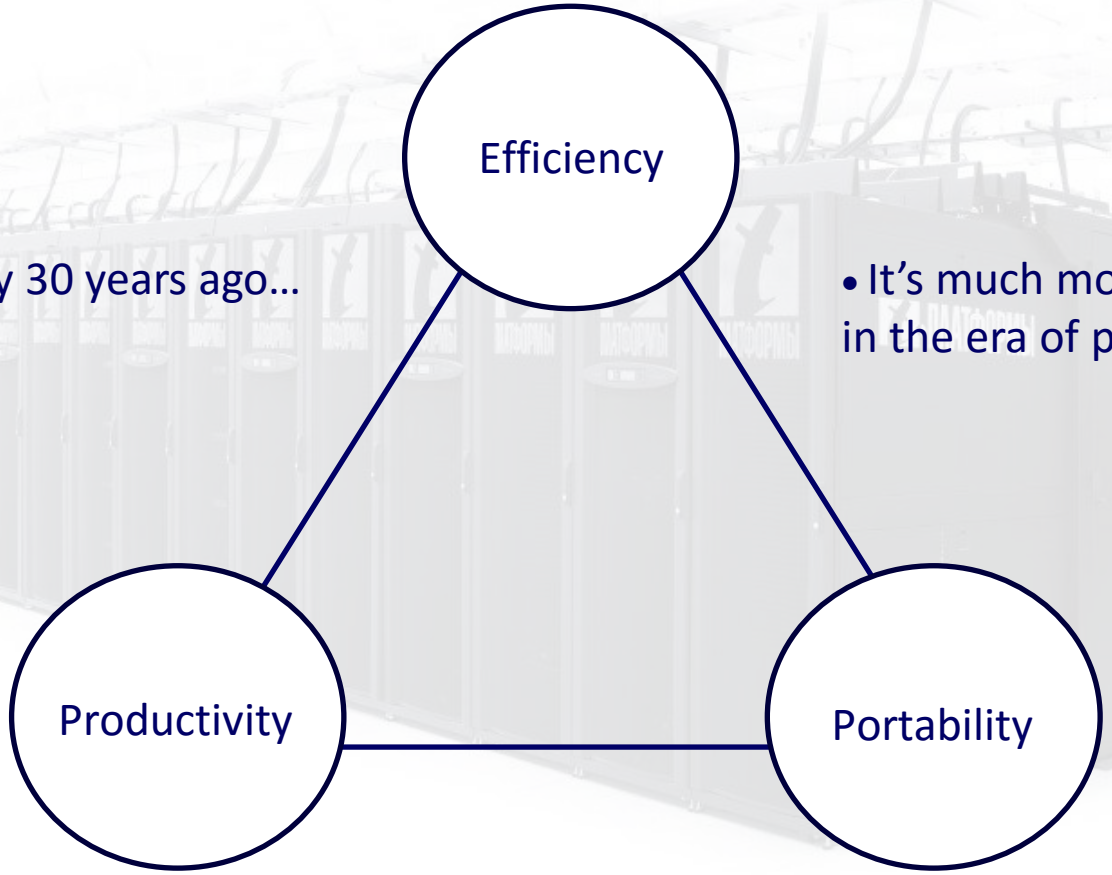
- It was not easy 30 years ago…

- It's much more difficult now in the era of parallel computing…

Another reason – diversity of architectures

Software Development Process

(We always make a compromise…)



- It was not easy 30 years ago…

- It's much more difficult now in the era of parallel computing…

- Diversity of architectures …

Code portability, performance portability – Is it Difficult?

Portability… Is It Always Difficult?

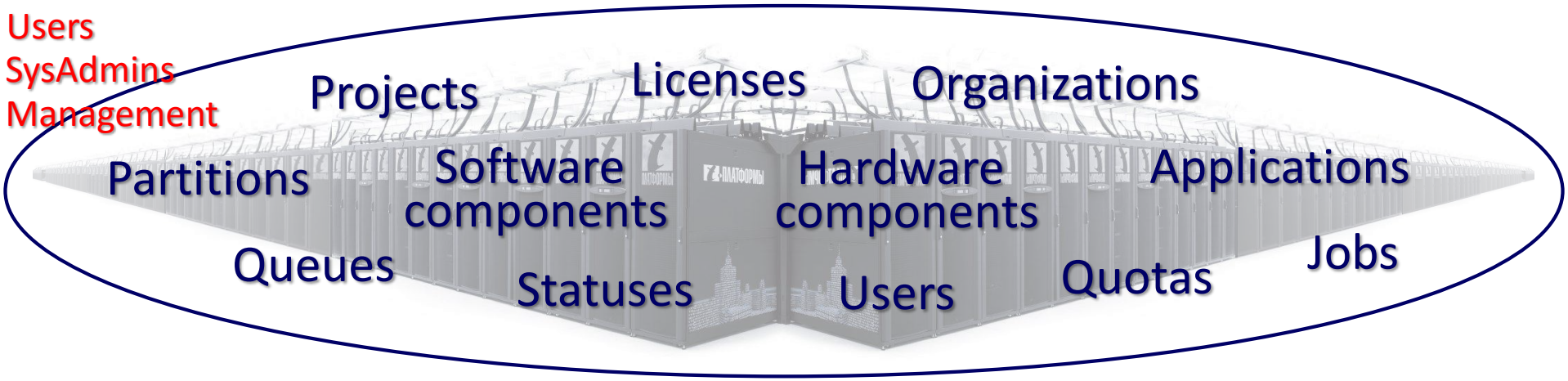
Portability… Is It Always Difficult?

Complexity of Supercomputing Centers
(reasons to talk about "Building Up Parallel Computing World")

Where are sources of efficiency losses?

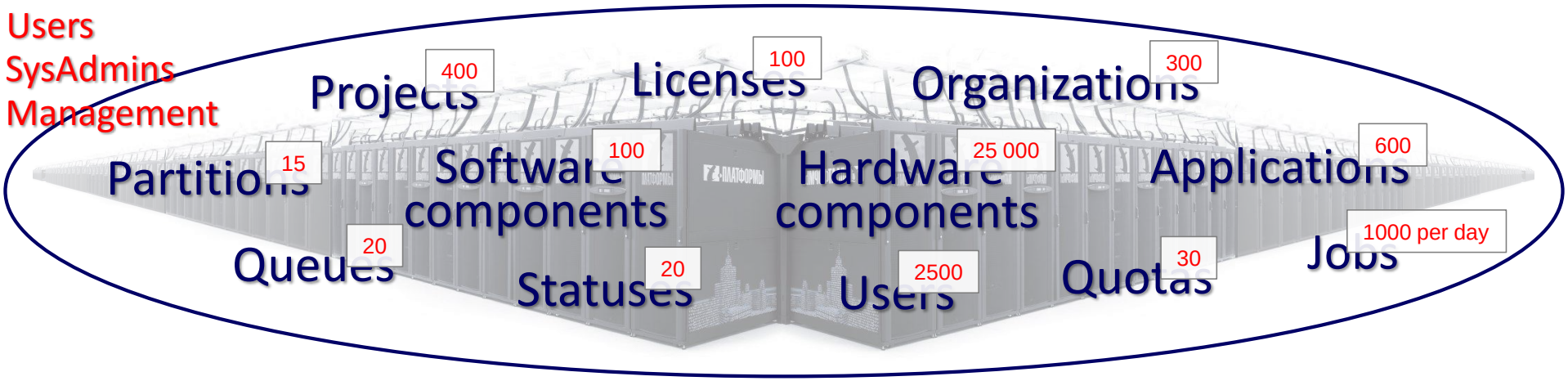
Complexity of Supercomputing Centers

Is it difficult to control few components ? *A few ?..*



A few? Info on MSU "Lomonosov" Supercomputer :

(1.7 Pflops, 6000 computing nodes, 12K CPUs, 2K GPUs…)



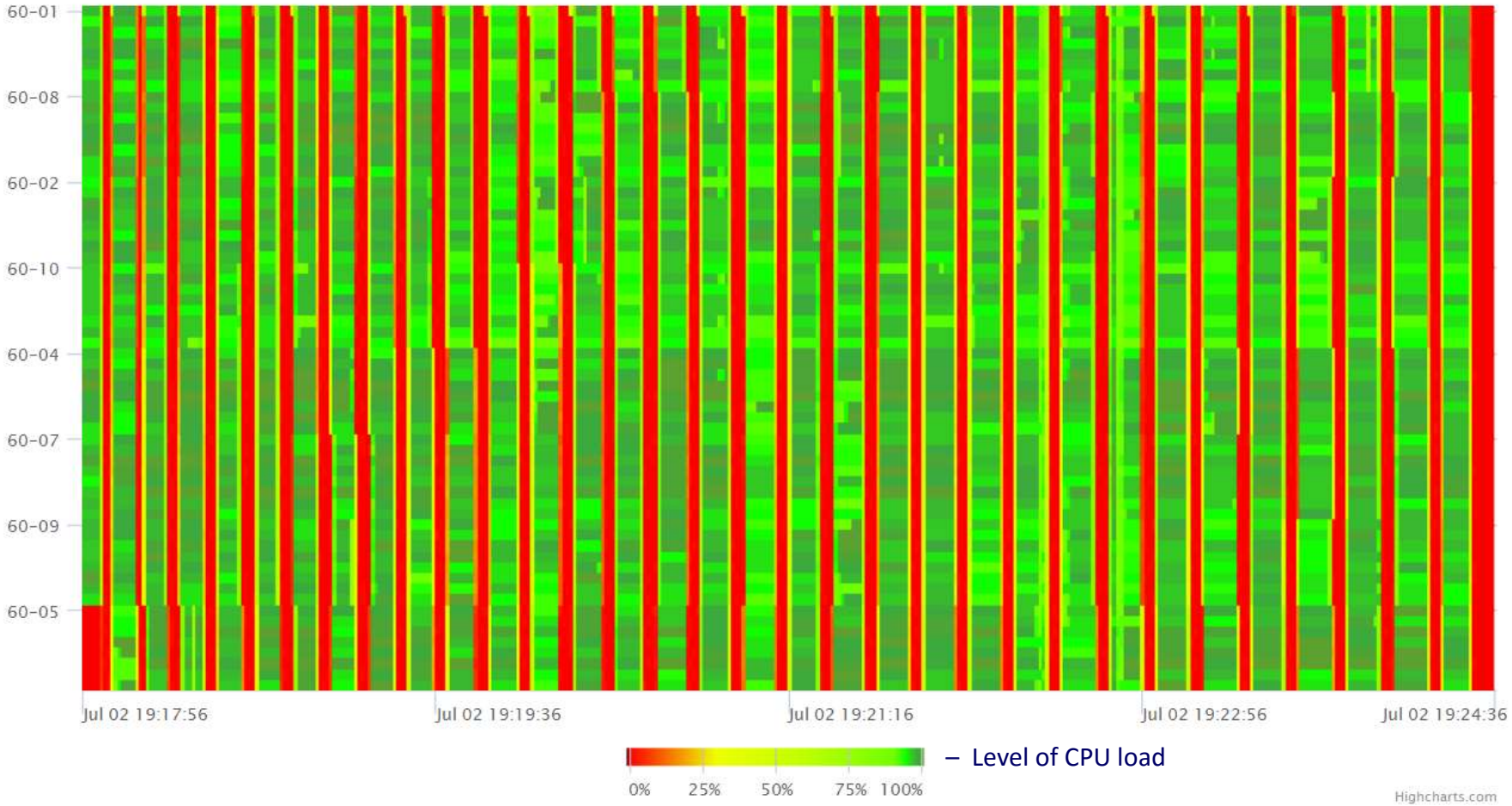
Current trend: all these numbers grow extremely fast !

⇩

It's impossible to predict/describe a state of supercomputers…

We have almost lost control over supercomputers…

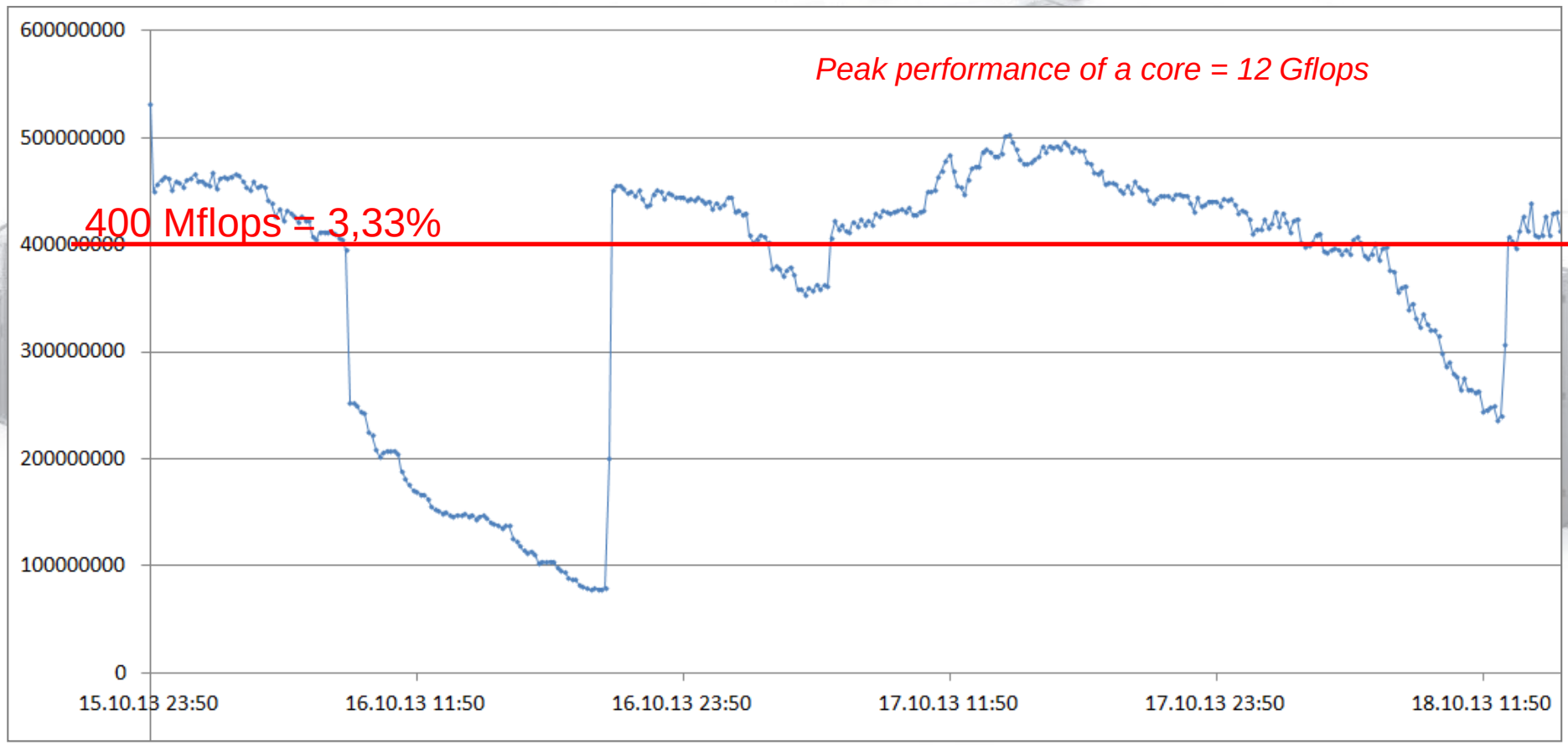
Structure of algorithms and applications
(one more reason to talk about "Building Up Parallel Computing World")



– Level of CPU load

Efficiency of Supercomputing Centers

(another reason to talk about "Building Up Parallel Computing World")



Average performance (one core) of "Chebyshev" supercomputer for 3 days

Today the key statement in "computing area" is "Building Up Parallel Computing World". Why ?

There are a number of reasons…

There are a lot of questions…

How well do we know properties, features of Parallel Algorithms ?

Should we think about Parallel Algorithms?

Yes… Unfortunately, Yes…

How well do we know static and dynamic characteristics of Parallel Programs ?

Should we think about properties of Parallel Programs?

Yes… Unfortunately, Yes…

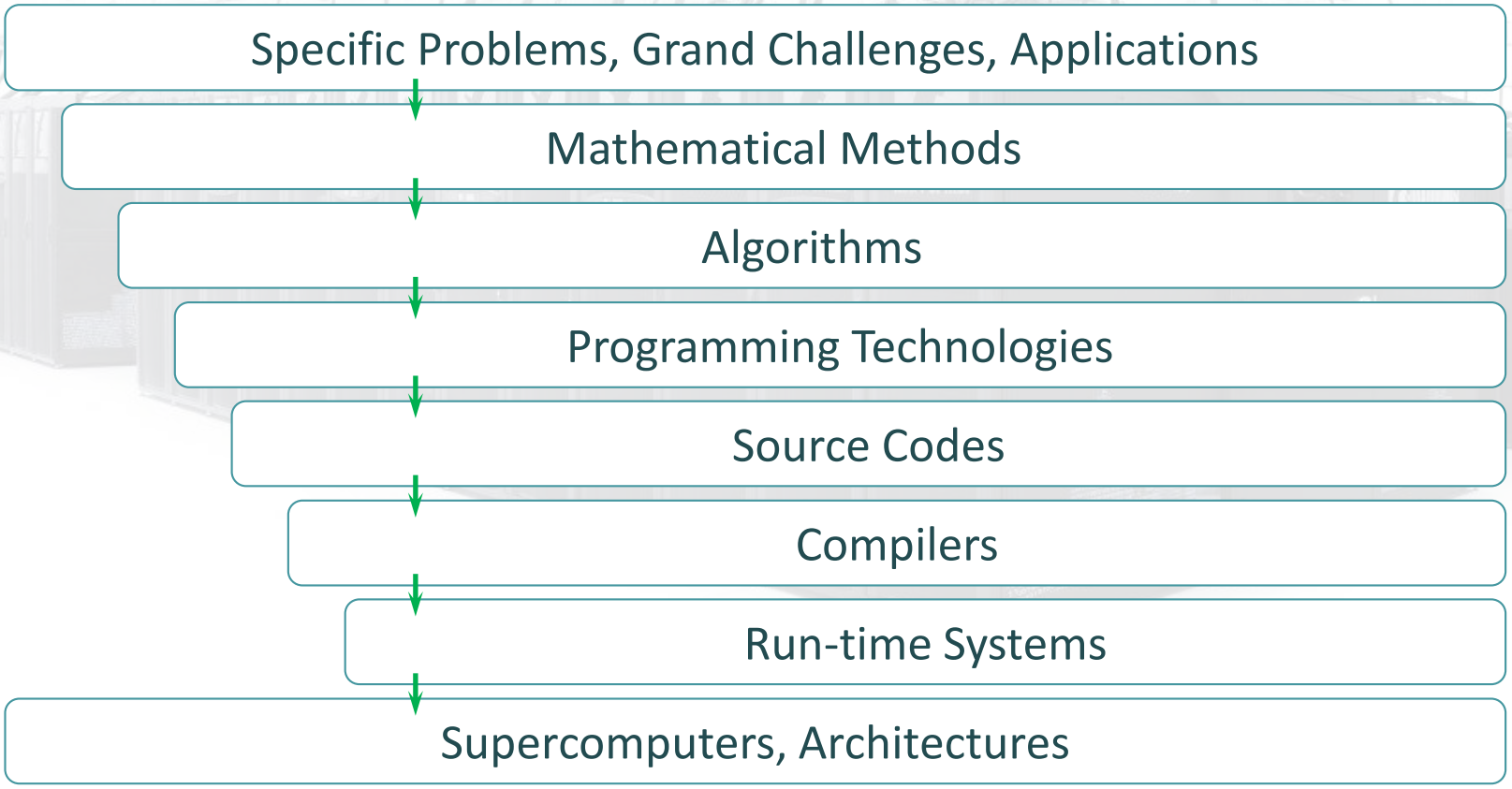
How well do we know architectures of parallel computers ?

Should we think about architectures?

Yes… Unfortunately, Yes…

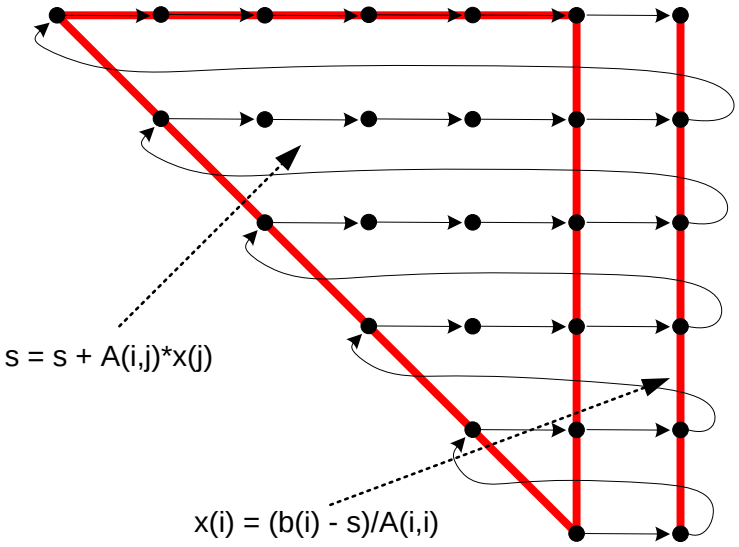
Have we ever met these questions before ? Yes, often…

Main stages to solve problems
(Supercomputing co-design chain)

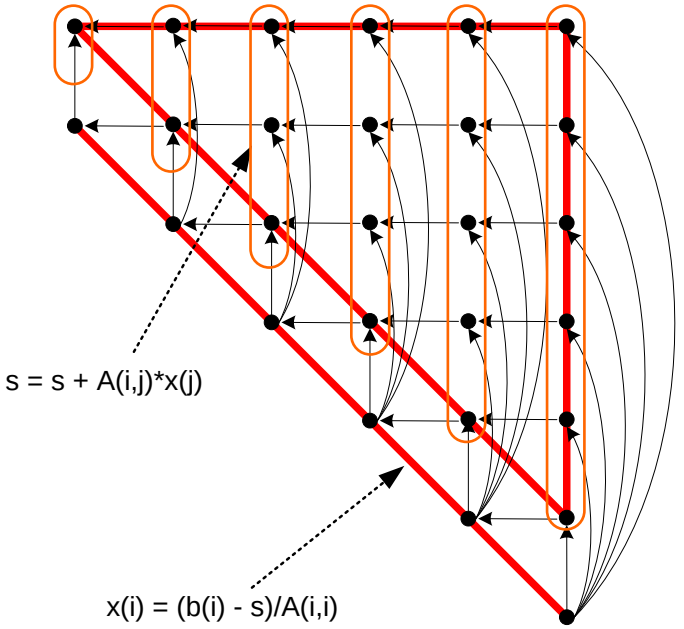


Main stages to solve problems: Methods and Algorithms

(GAUSS elimination method)



Serial only



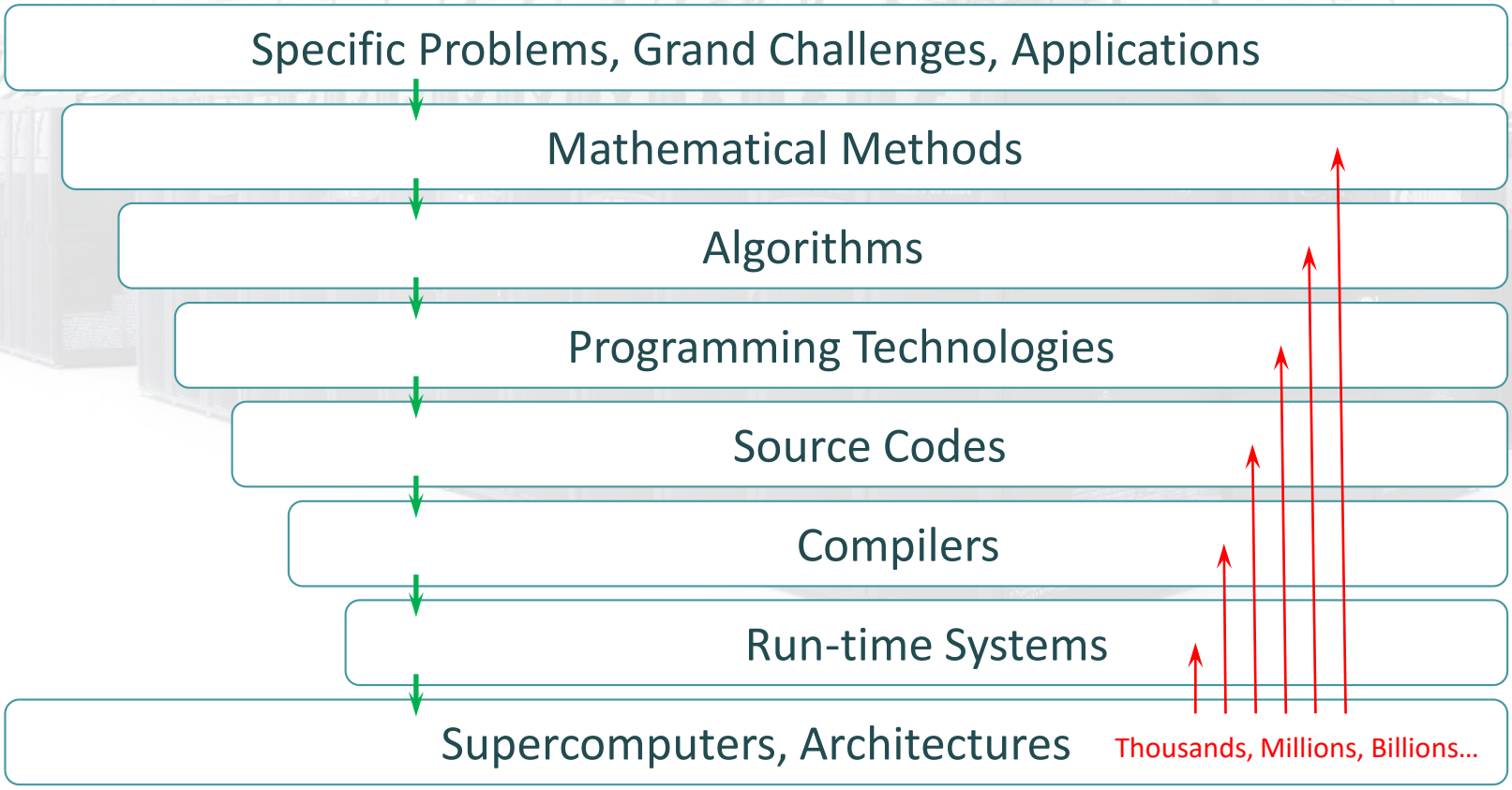
Parallel execution

```
do i = n, 1, -1
   s = 0
   do j = i+1, n
      s = s + A(i,j)*x(j)
   end do
   x(i) = (b(i) - s)/A(i,i)
end do
```

Order of iterations is the only difference !

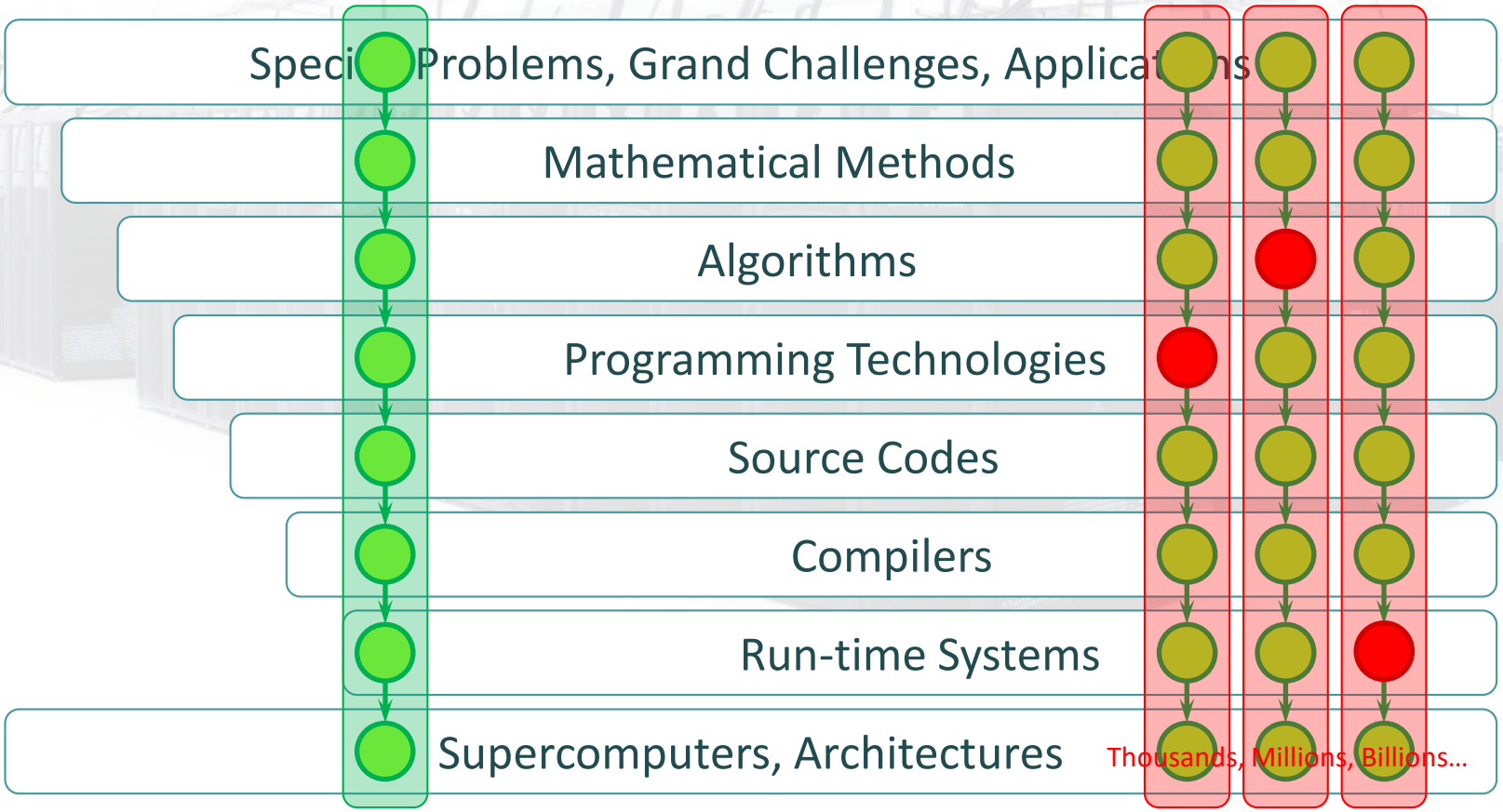
```
do i = n, 1, -1
   s = 0
   do j = n, i+1, -1
      s = s + A(i,j)*x(j)
   end do
   x(i) = (b(i) - s)/A(i,i)
end do
```

Main stages to solve problems
(Ubiquitous parallelism)

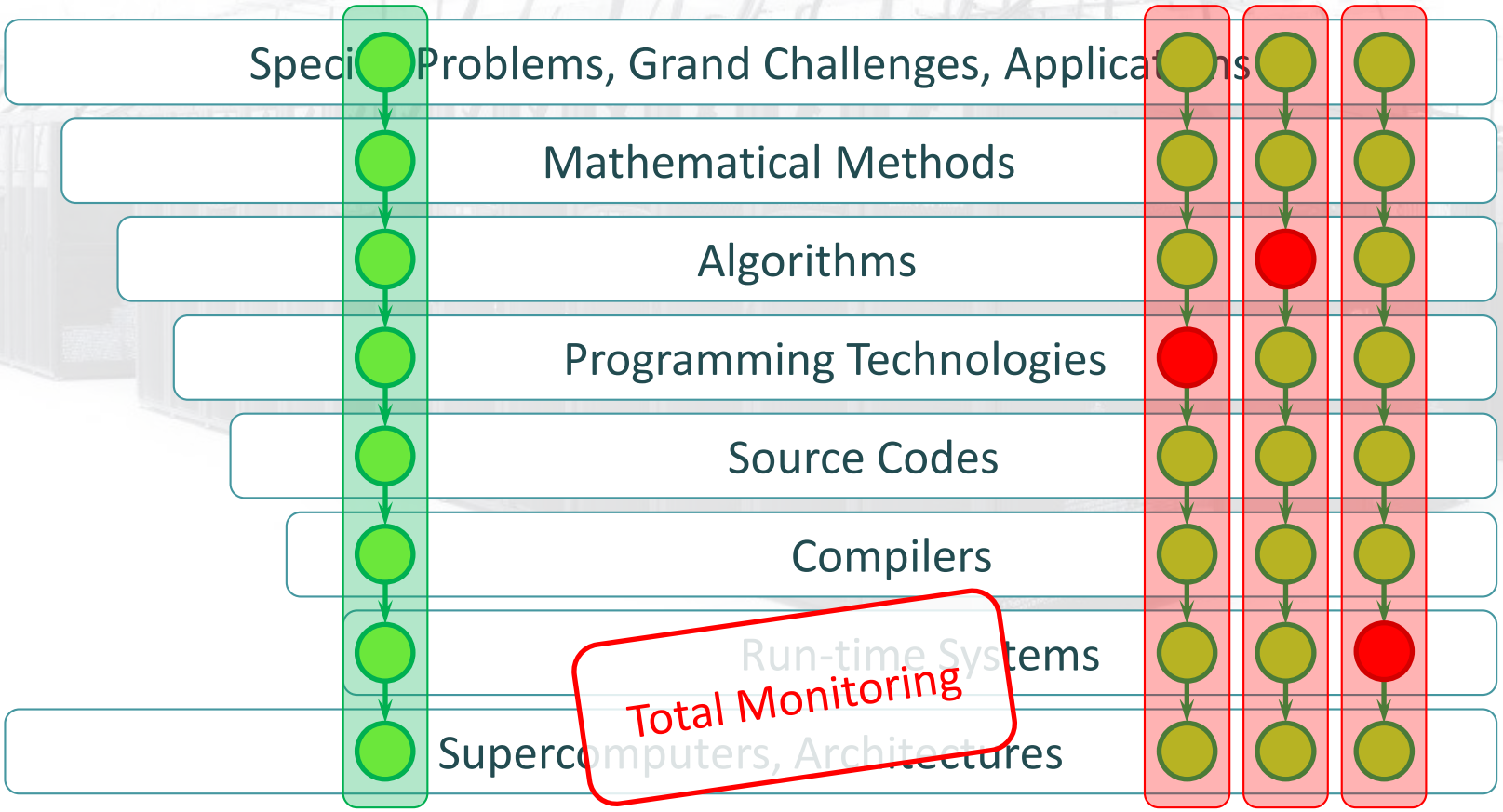


Main stages to solve problems
(Where supercomputing co-design begins)

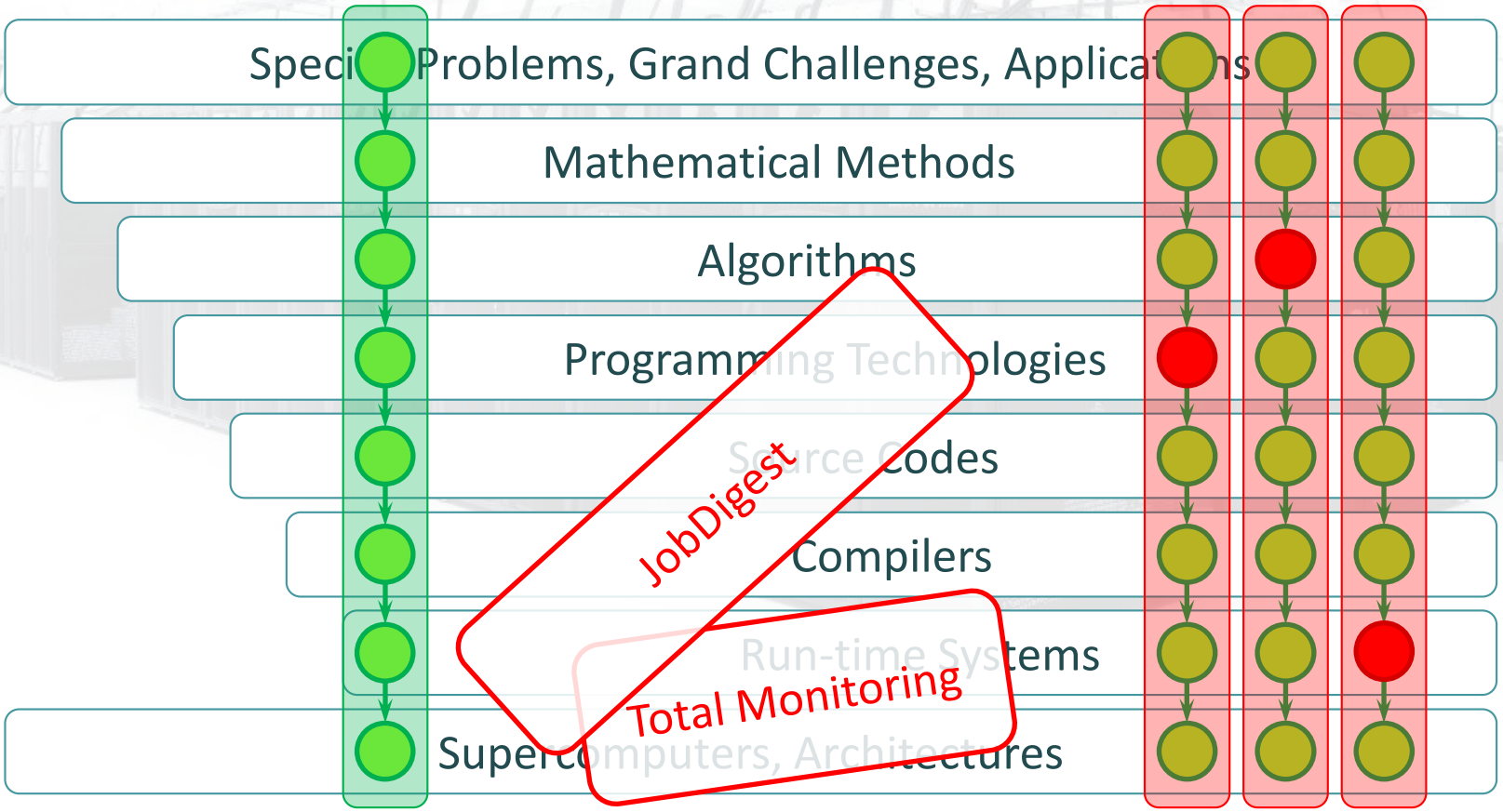
How to ensure efficiency of this supercomputing co-design chain ?



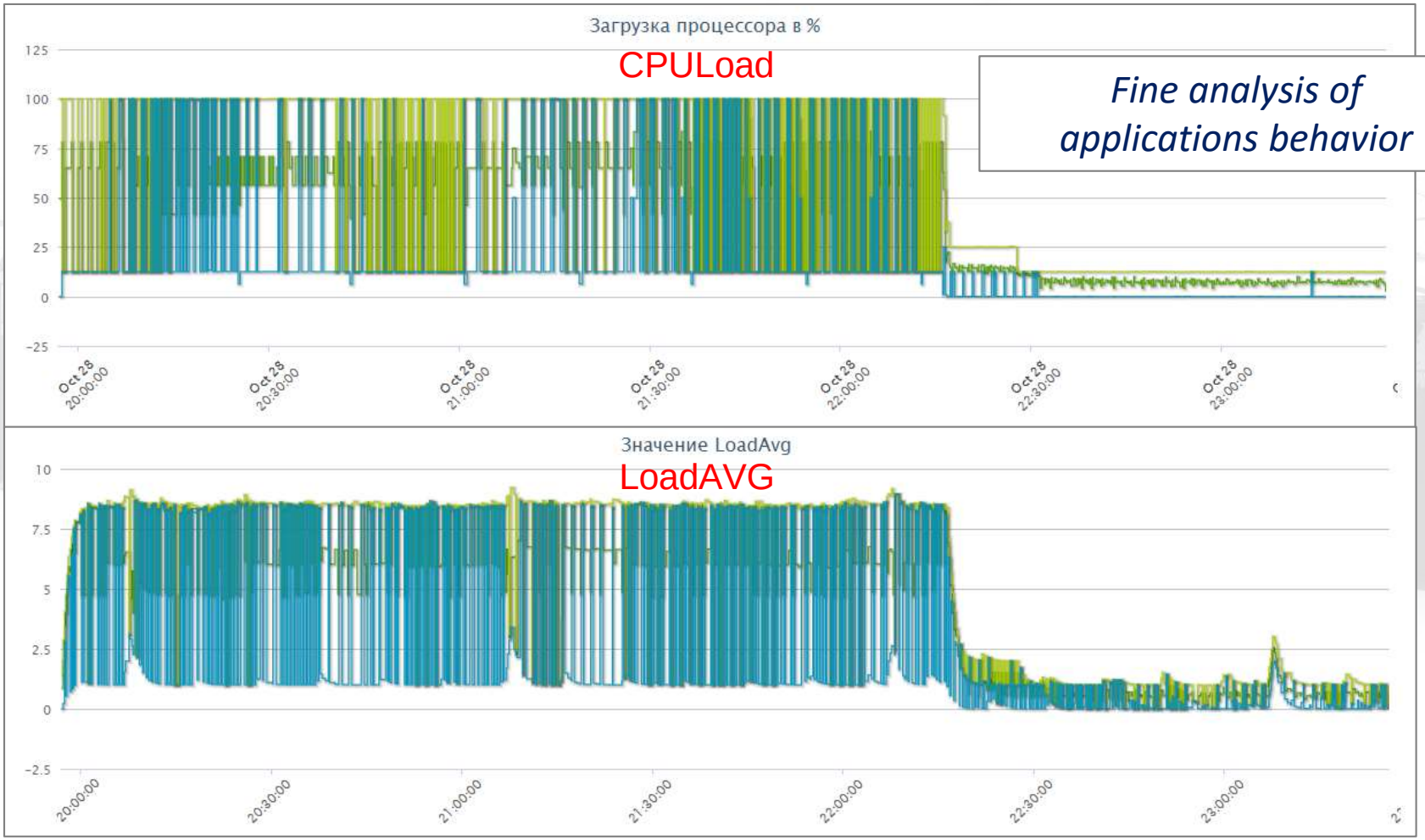
Supercomputing Co-Design Technologies and Tools
(A practical approach)



Supercomputing Co-Design Technologies and Tools
(A practical approach)



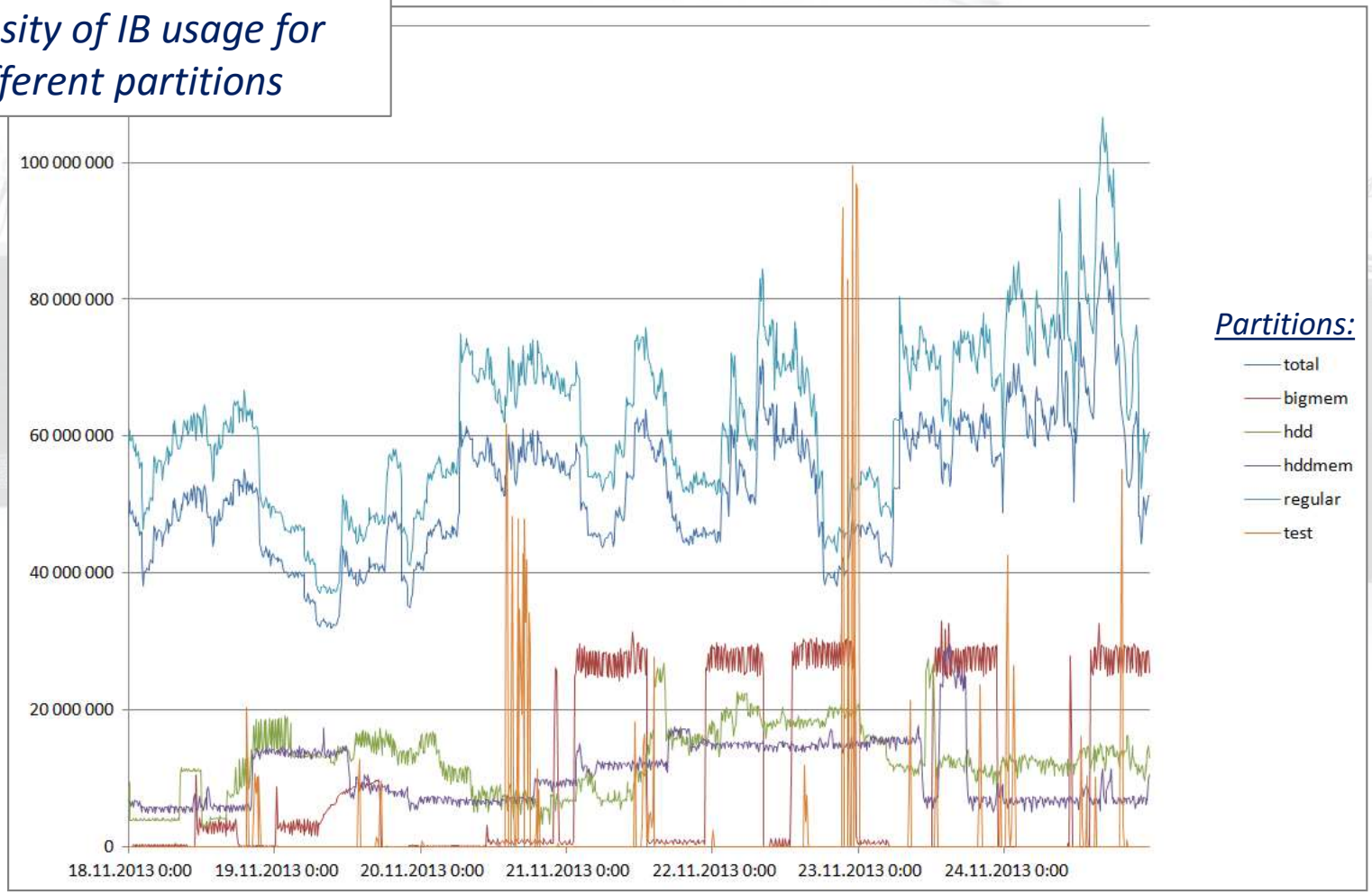
Supercomputing Co-Design Technologies and Tools
(JobDigest of an application)



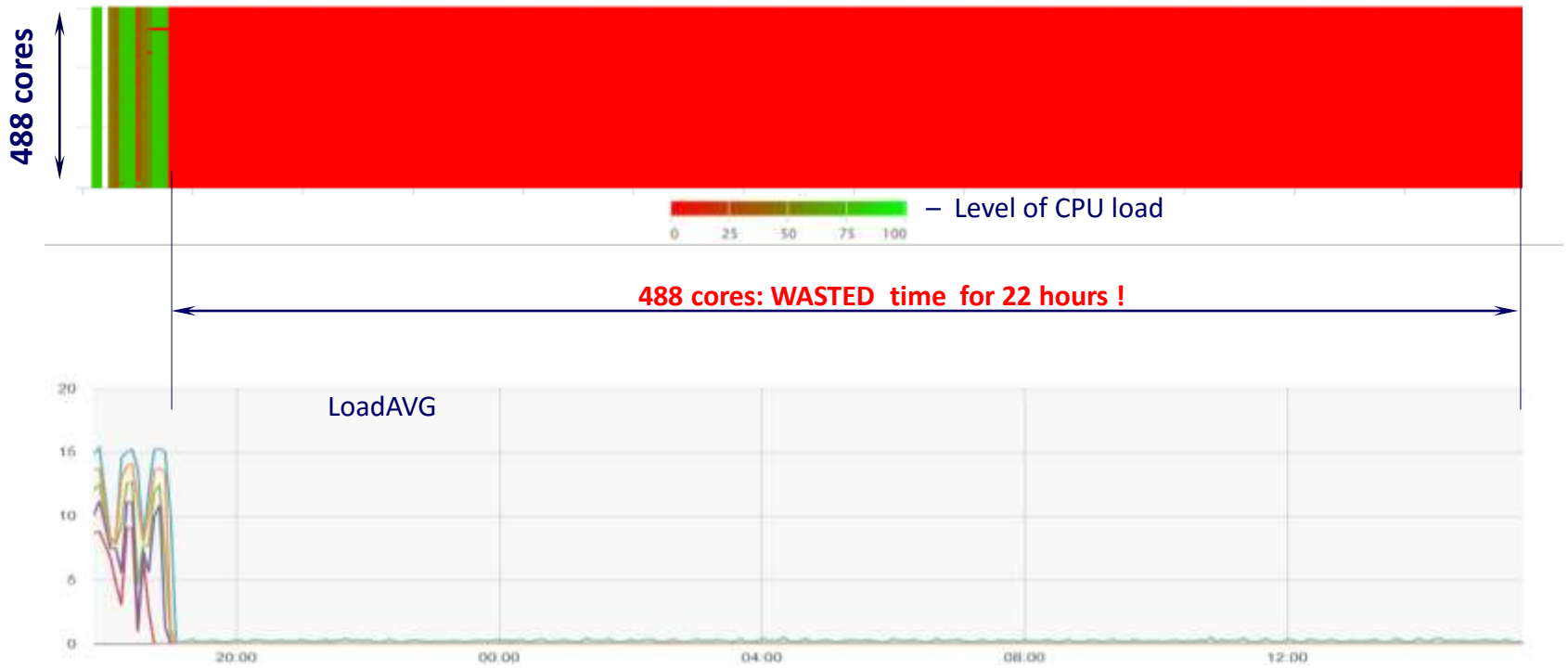
Fine analysis of applications behavior

Supercomputing Co-Design Technologies and Tools
(analysis at different levels)

"Chebyshev" supercomputer: intensity of IB usage for different partitions



Supercomputing Co-Design Technologies and Tools
(How well do we control a state of supercomputers?)



What could be a reason of this situation?

- Hardware failure? Yes, it could be …
- Software failure? Yes …
- Error in the code? Yes …
- Algorithmic problem? Yes …

Supercomputing Co-Design Technologies and Tools

(OctoTron: predictability and autonomous life of supercomputers)

***Large numbers in supercomputers**: cores, processors, accelerators, nodes, HW&SW components, files, indexes, users, projects, processes, threads, running and queued jobs…*

We don't know for sure the current state of supercomputer's components …

Supercomputing Co-Design Technologies and Tools

(OctoTron: predictability and autonomous life of supercomputers)

What is now? *We hope that a component works until we get an evidence that it has failed.*

What do we need?

Our expectations = Reality

We need a guarantee*:*
if something goes wrong inside a supercomputer we shall be notified immediately.

We want a system behaves in a way we expect it should behave.

Supercomputing Co-Design Technologies and Tools

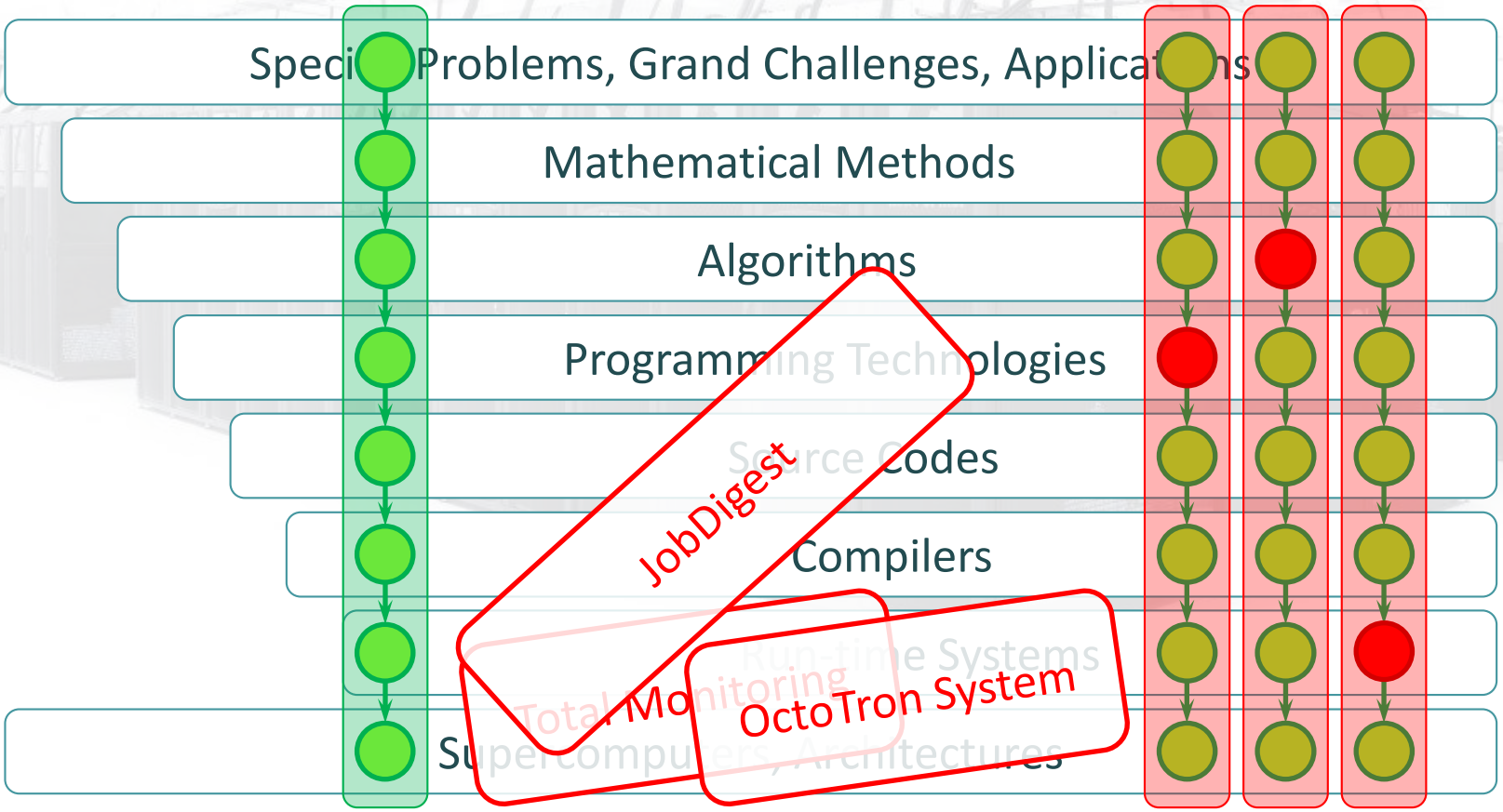
(OctoTron: predictability and autonomous life of supercomputers)

If discrepancy occurs between our expectations and supercomputer behavior (i.e. reality) we need to know immediately about it.

How can we do that? Supercomputer is huge, we can't control it to a full extent any more.

But… supercomputer can do it itself (instead of us), if we explain what "our expectations" are.

Supercomputing Co-Design Technologies and Tools
(A practical approach)



Supercomputing Co-Design Technologies and Tools
(OctoTron: predictability and autonomous life of supercomputers)

Supercomputers should be autonomous in self-control.
(They become more dynamic, more sophisticated, more and more parallel)

…The larger supercomputers, the more autonomous they should be...

Supercomputing Co-Design Technologies and Tools
(OctoTron: predictability and autonomous life of supercomputers)

A guarantee of "our expectations = reality", how this can be done?

- *a formal model of supercomputers (**model is a graph**),*
- *a set of formal rules,*
- *a set of reactions,*

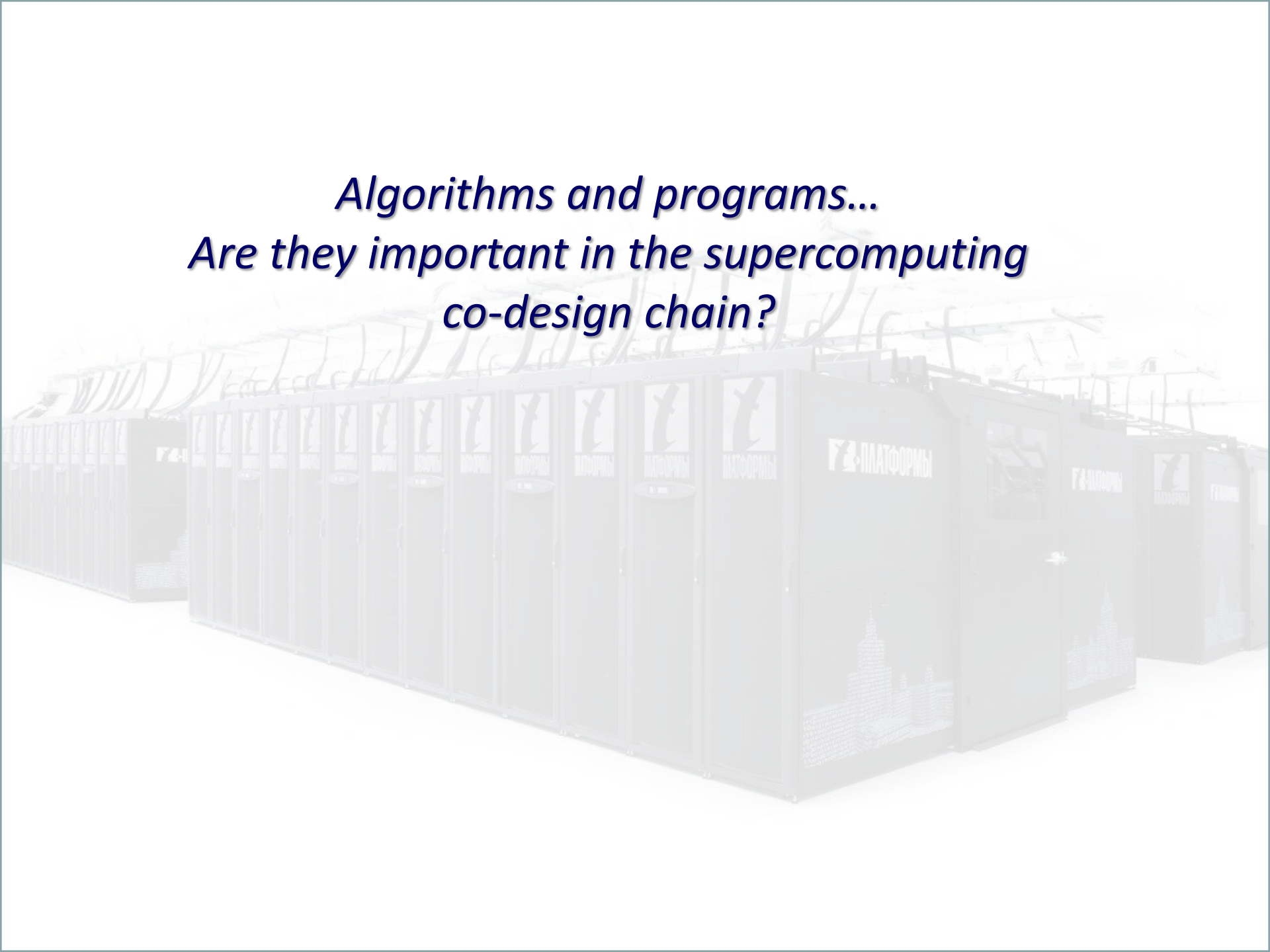
Autonomous life and control of MSU supercomputers:

*-**"Chebyshev"** supercomputer, 60 Tflops, 625 CPUs:*
10 228 nodes, 24 698 edges, 205 044 attributes, 160 rules, 100 reactions;
*- **"Lomonosov"** supercomputer, 1.7 Pflops, 12 000 CPUs, 2 000 GPU:*
116 000 nodes, 332 000 edges, 2 400 000 attributes,...

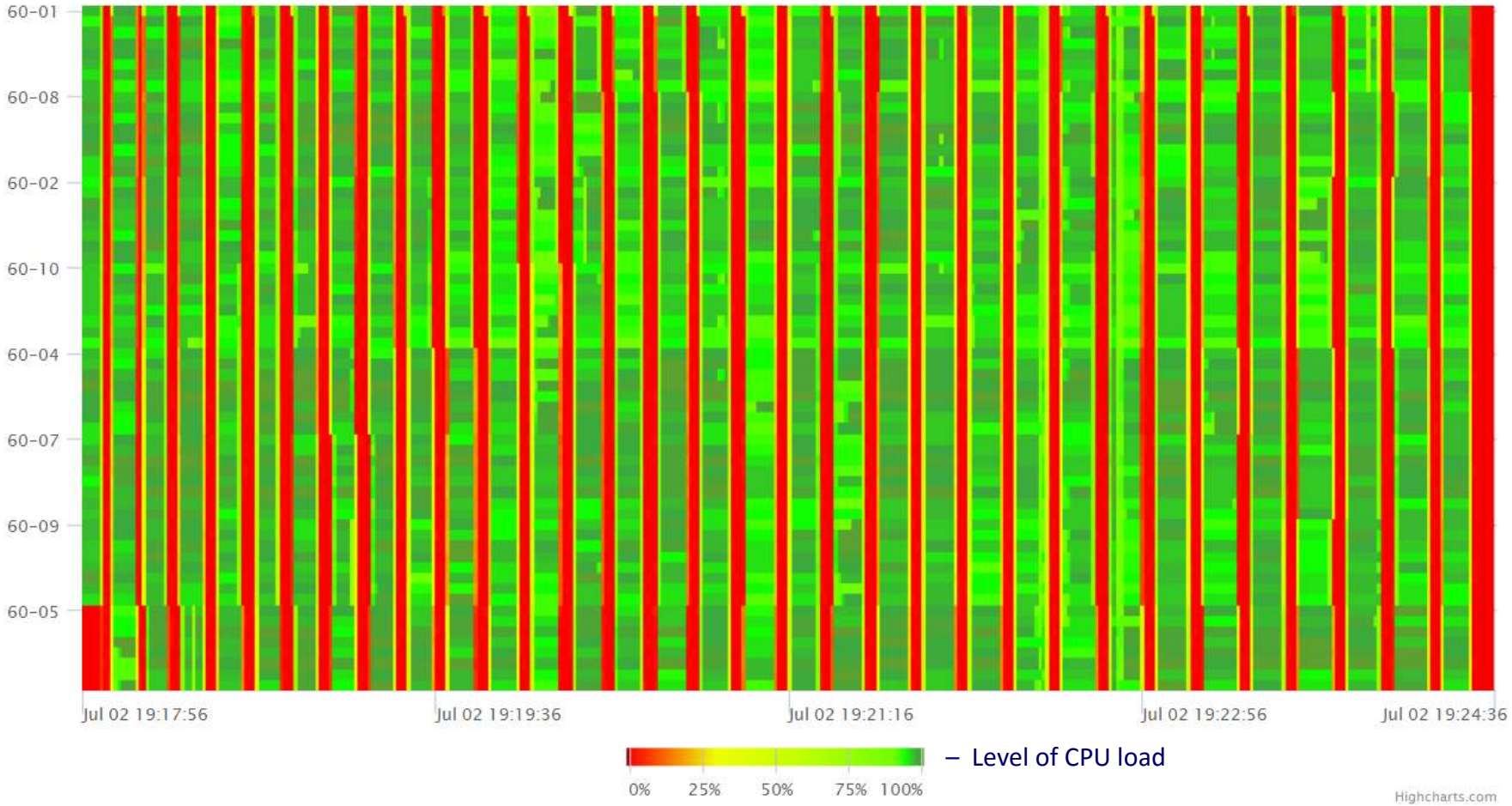
Initial deployment, Detection of faults, critical and emergency situations, Turning off minimum amount of hardware, Self diagnostics, Previous accidents, etc. are done according to a model and rules.

Current trend: many decisions about control over HW&SW of supercomputers must be taken automatically.

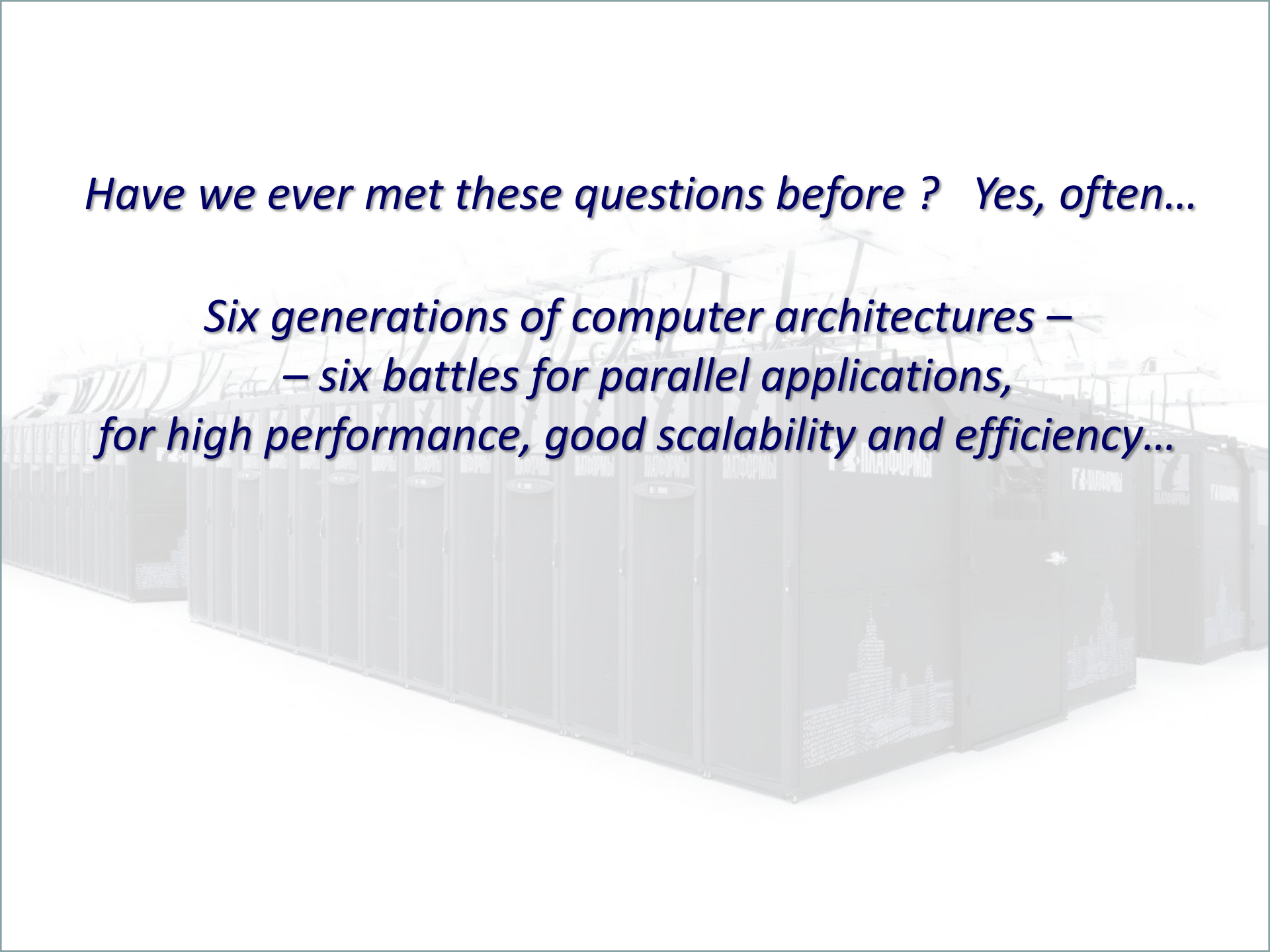
Algorithms and programs…
Are they important in the supercomputing co-design chain?



What is a good parallel program / algorithm?



– Level of CPU load



Have we ever met these questions before ? Yes, often…

Six generations of computer architectures –
– six battles for parallel applications,
for high performance, good scalability and efficiency…

6 Generations of computers and programming paradigms
(endless rewriting of applications)

Cray-1 supercomputer

Vector computers

Mid 70-s.

Features: pipelined functional units, vector instructions, vector registers.

Programming: vectorization of the innermost loops.

6 Generations of computers and programming paradigms
(endless rewriting of applications)

Cray X-MP supercomputer

Cray Y-MP supercomputer

Vector-parallel computers

80-s.

Features: pipelined functional units, vector instructions, vector registers.
2-32 processors, shared memory.

Programming: vectorization of the innermost loops, parallelization of outer loops.

6 Generations of computers and programming paradigms
(endless rewriting of applications)

Cray T3D supercomputer

Intel Paragon XPS140 supercomputer

Massive-parallel computers

90-s.

Features: thousands of processors, distributed memory.

Programming: explicit message passing and data distribution, MPI.

6 Generations of computers and programming paradigms
(endless rewriting of applications)

DEC AlphaServer

Sun StarFire supercomputer

Shared-memory computers

Mid 90-s.

Features: tens/hundreds of processors, shared memory.

Programming: single address space, local and shared variables, OpenMP.

6 Generations of computers and programming paradigms
(endless rewriting of applications)

MSU supercomputer “Chebyshev”

“K” supercomputer

Clusters of SPM-nodes

2000-s.

Features: tens/hundreds thousands of nodes, distributed memory.
Tens of processors/cores, shared memory.

Programming: hybrid parallel programming technology MPI + OpenMP.

6 Generations of computers and programming paradigms
(endless rewriting of applications)

MSU supercomputer “Lomonosov”

Tianhe-2 supercomputer

Clusters of SMP-nodes with accelerators

Mid 2000-s.

Features: tens/hundreds thousands of nodes, distributed memory.
Tens of processors/cores, shared memory. Several accelerators (GPUs, Phi’s).

Programming: MPI+OpenMP+OpenCL/CUDA;

Generations of Parallel Computer Architectures
(or How often have we have to rewrite applications completely?)

Parallel programming paradigms (from the 70s up to now):

70s - Loop Vectorization (innermost)
80s - Loop Parallelization (outer) + Vectorization (innermost)
90s - MPI
mid 90s - OpenMP
mid 2000s - MPI+OpenMP
2010s - CUDA, OpenCL, MPI+OpenMP+accelerators (GPU, Xeon Phi)
…

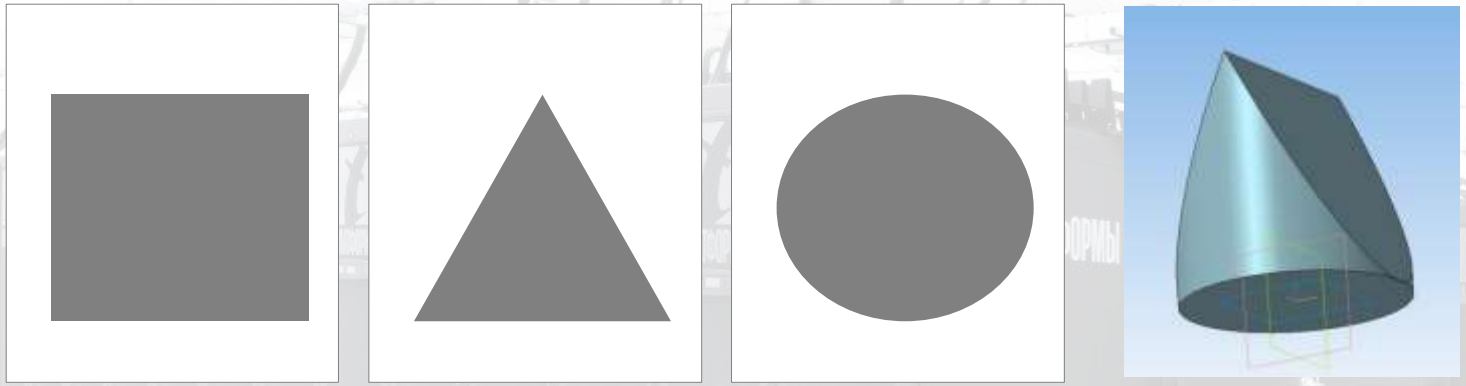
Changes in computer architectures do not change algorithms! But…

For each generation of a new computing platform we have to:
- Analyze algorithms to find a way to match better features and properties of the platform ;
- Express the properties of algorithms we found to obtain efficient Implementation for the platform.

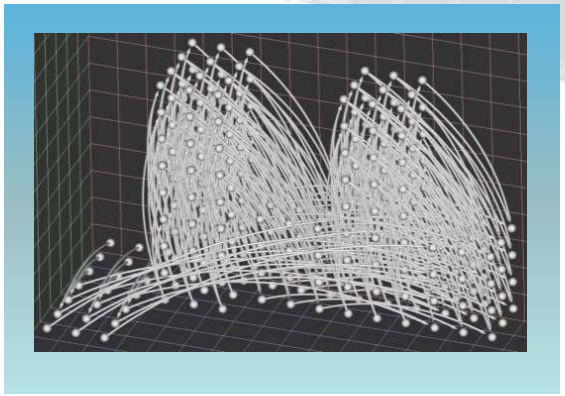
Changes in computer architectures do not change algorithms!

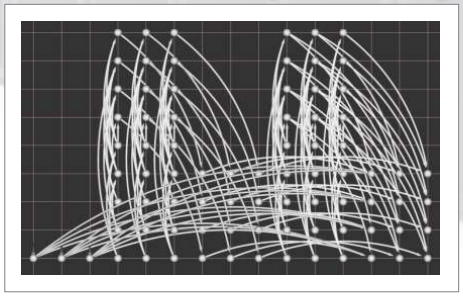
(Algorithms remain the same)

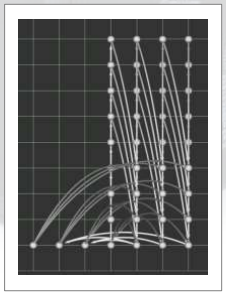
Are these figures different?

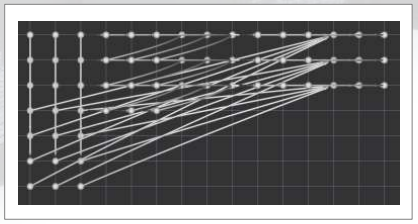


What are possible representations of this algorithm?









…

Generations of Parallel Computer Architectures
(or How often have we have to rewrite applications completely?)

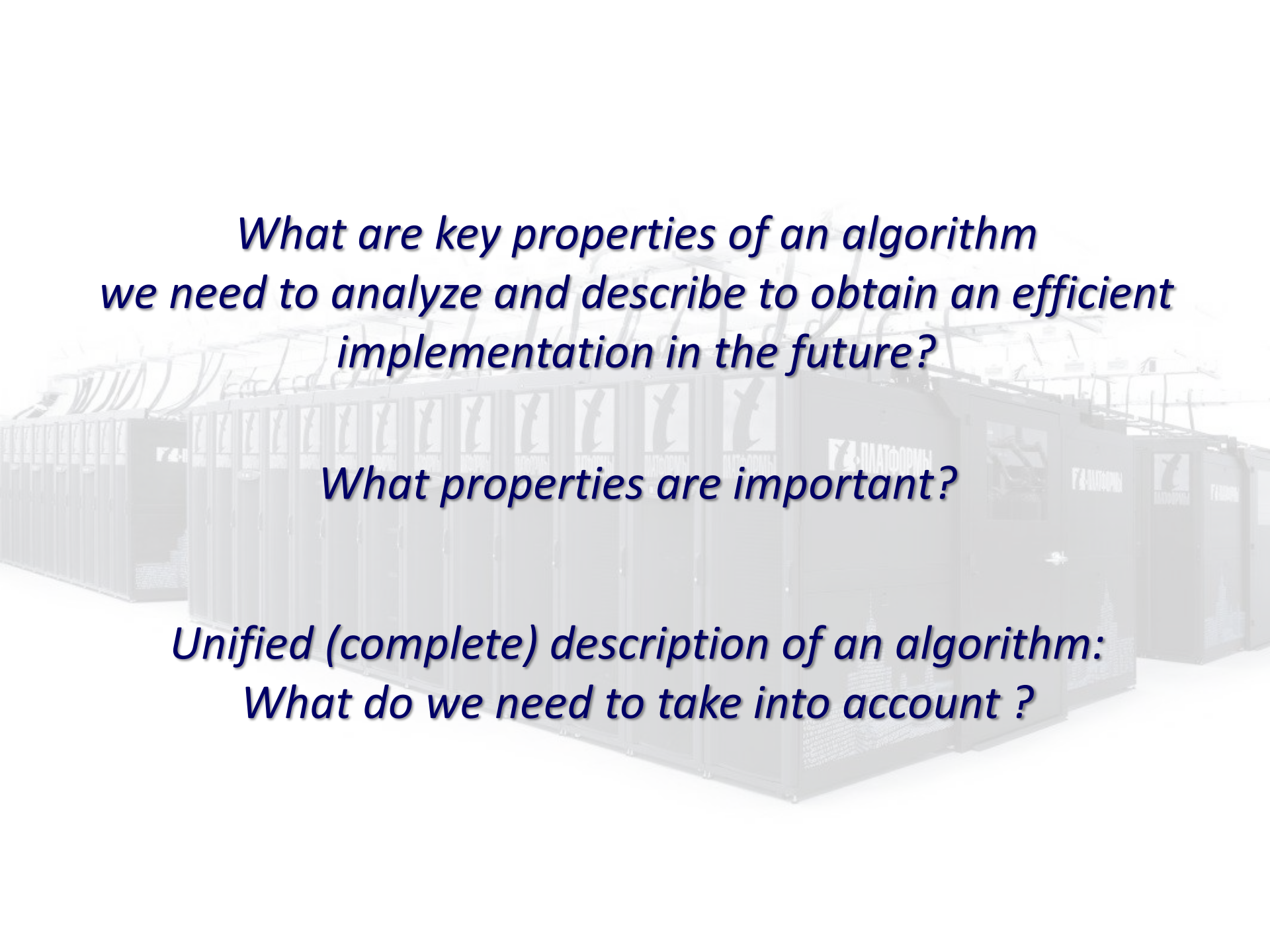
Parallel programming paradigms (from the 70s up to now):

70s - Loop Vectorization (innermost)
80s - Loop Parallelization (outer) + Vectorization (innermost)
90s - MPI
mid 90s - OpenMP
mid 2000s - MPI+OpenMP
2010s - CUDA, OpenCL, MPI+OpenMP+accelerators (GPU, Xeon Phi)
…

Can we analyze algorithms once and for all ?

Changes in computer architectures do not change algorithms! But…

For each generation of a new computing platform we have to:
- Analyze algorithms to find a way to match better features and properties of the platform ;
- Express the properties of algorithms we found to obtain efficient Implementation for the platform.



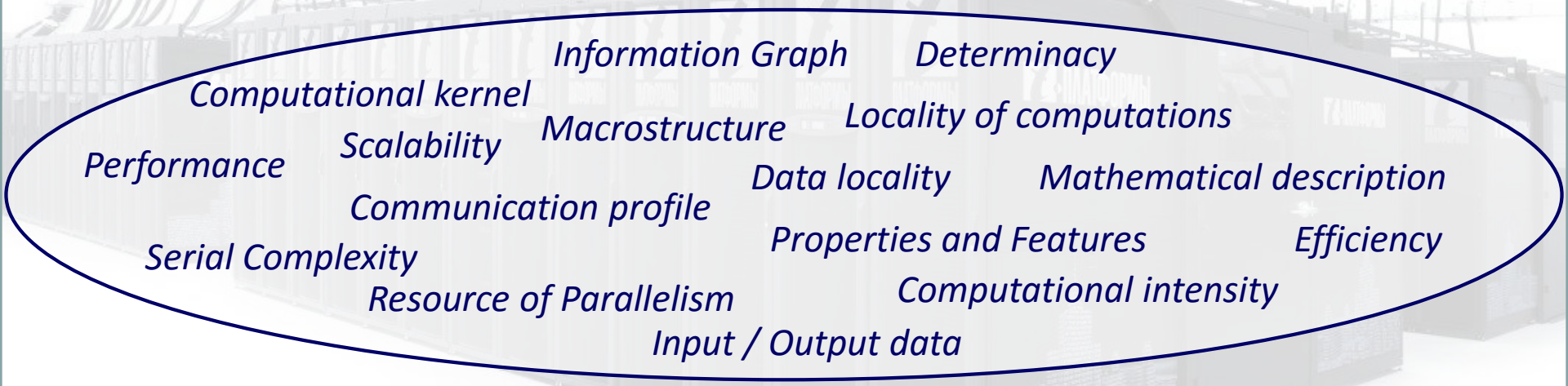
What are key properties of an algorithm we need to analyze and describe to obtain an efficient implementation in the future?

What properties are important?

Unified (complete) description of an algorithm: What do we need to take into account ?

Description of Algorithms

(What should be included in this description?)



Description of Algorithms
(at the starting point: Cholesky decomposition)

Description

For positive definite Hermitian matrices (*symmetric matrices in the real case*), we use the decomposition $A = LL^*$, where L is the lower triangular matrix, or the decomposition $A = U^*U$, where U is the upper triangular matrix. These forms of the Cholesky decomposition are equivalent in the sense of the amount of arithmetic operations and are different in the sense of data represntation. The essence of this decomposition consists in the implementation of formulas obtained uniquely for the elements of the matrix L from the above equality. The Cholesky decomposition is widely used due to the following features.

Mathematical Description

Input data: a symmetric positive definite matrix A whose elements are denoted by a_{ij}).

Output data: the lower triangular matrix L whose elements are denoted by l_{ij}).

The Cholesky algorithm can be represented in the form

$$l_{11} = \sqrt{a_{11}},$$

$$l_{j1} = \frac{a_{j1}}{l_{11}}, \quad j \in [2, n],$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{p=1}^{i-1} l_{ip}^2}, \quad i \in [2, n],$$

$$l_{ji} = \left(a_{ji} - \sum_{p=1}^{i-1} l_{ip} l_{jp} \right) / l_{ii}, \quad i \in [2, n-1], j \in [i+1, n].$$

Remarks on the Algorithm

The Cholesky decomposition allows one to use the so-called *accumulation mode* due to the fact that the significant part of computation involves *dot product operations*. Hence, these dot products can be accumulated in double precision for additional accuracy. In this mode, the Cholesky method has the least *equivalent perturbation*. During the process of decomposition, no growth of the matrix elements can occur, since the matrix is symmetric and positive definite. Thus, the Cholesky algorithm is unconditionally stable.

Description of Algorithms
(at the starting point: Cholesky decomposition)

Computational Kernel

A computational kernel of its serial version can be composed of $\frac{n(n-1)}{2}$ dot products of the matrix rows:

$$\sum_{p=1}^{i-1} l_{ip} l_{jp}.$$

Data movements are extremely important!

Serial Complexity

The following number of operations should be performed to decompose a matrix of order n using a serial version of the Cholesky algorithm:

- n square roots,
- $\frac{n(n-1)}{2}$ divisiona,
- $\frac{n^3-n}{6}$ multiplications and $\frac{n^3-n}{6}$ additions (subtractions): the main amount of computational work.

Data movements are extremely important!

Baseline Serial Implementation

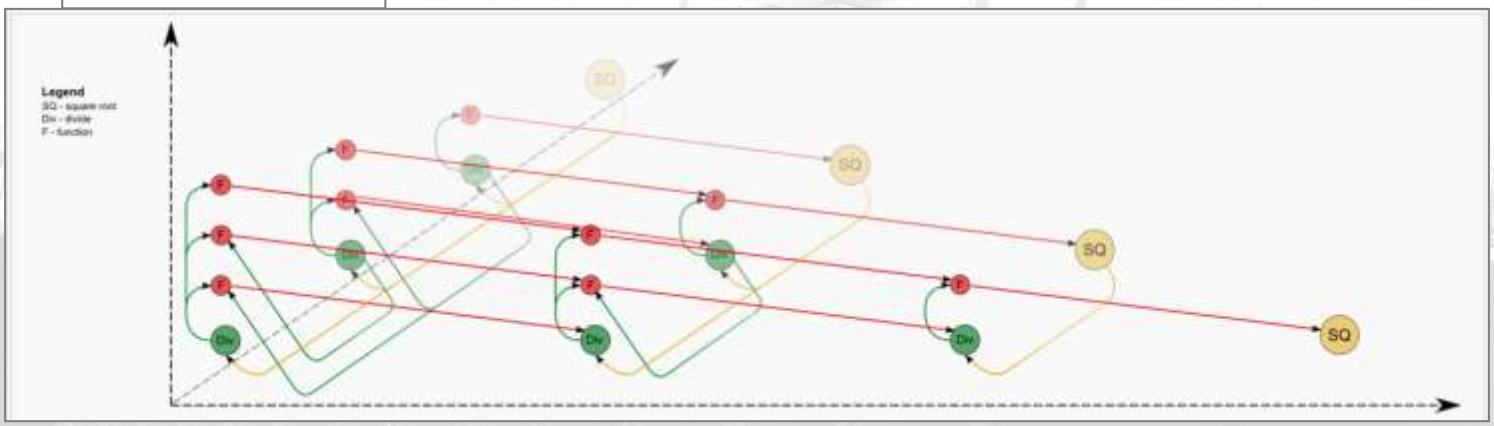
```
DO  I = 1, N
        S = A(I,I)
        DO  IP=1, I-1
                S = S - DPROD(A(I,IP), A(I,IP))
        END DO
        A(I,I) = SQRT (S)
        DO  J = I+1, N
                S = A(J,I)
                DO  IP=1, I-1
                        S = S - DPROD(A(I,IP), A(J,IP))
                END DO
                A(J,I) = S/A(I,I)
        END DO
END DO
```

Additional Info

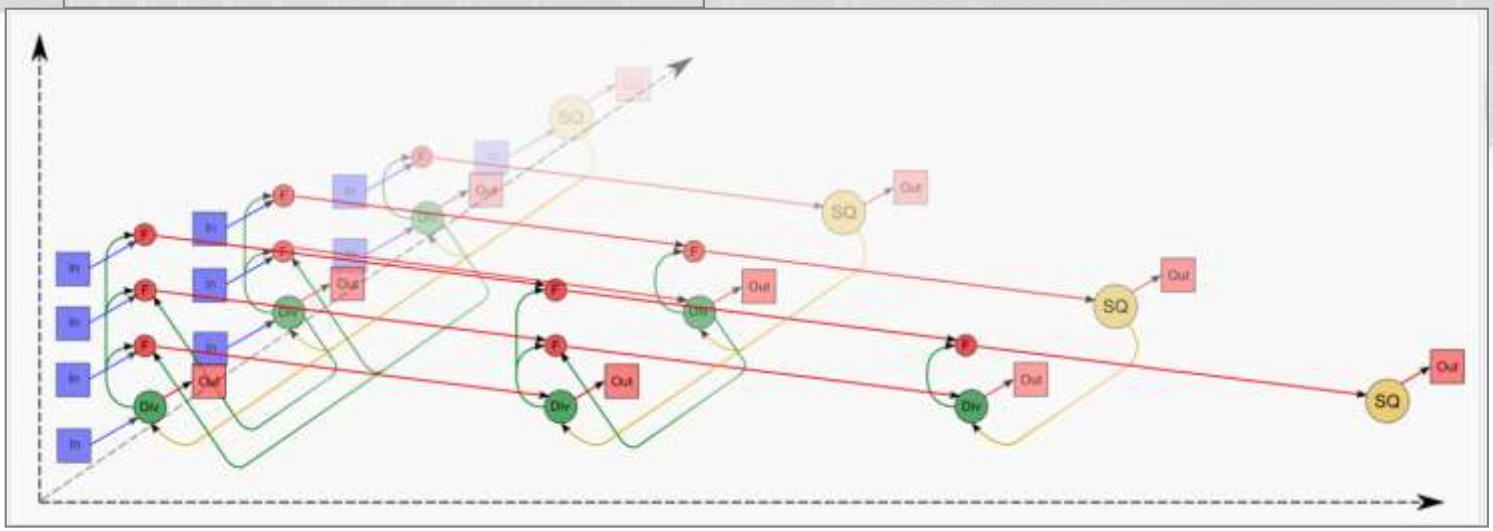
There exist block versions of this algorithm;

Description of Algorithms
(at the starting point: Cholesky decomposition)

Information Structure



Information Structure with Input/Output Data



Description of Algorithms
(at the starting point: Cholesky decomposition)

Data locality (memory usage profile)

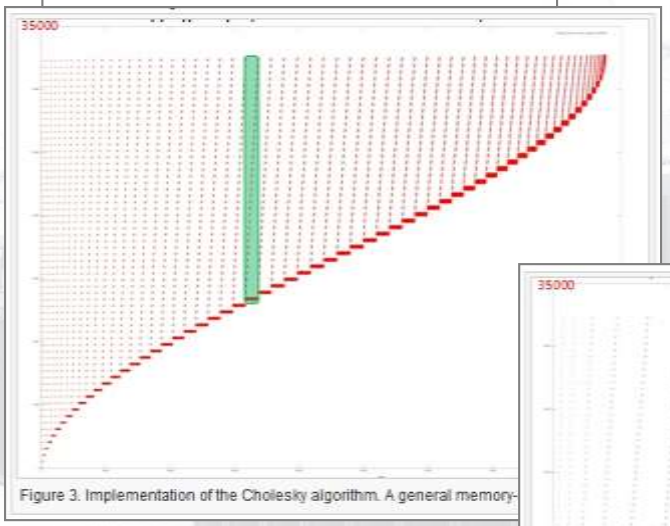


Figure 3. Implementation of the Cholesky algorithm. A general memory-

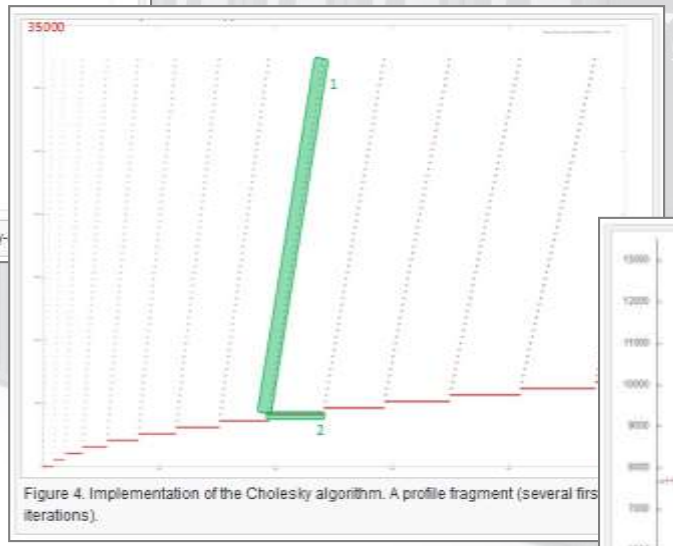


Figure 4. Implementation of the Cholesky algorithm. A profile fragment (several firs iterations).

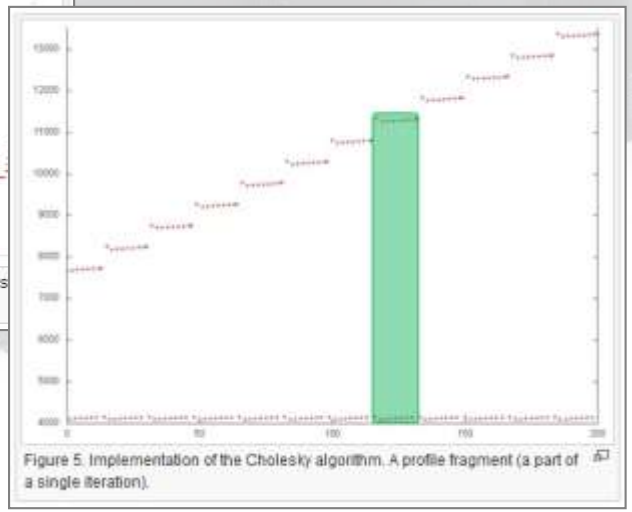


Figure 5. Implementation of the Cholesky algorithm. A profile fragment (a part of a single iteration).

Description of Algorithms
(at the starting point: Cholesky decomposition)

1 Properties and structure of the algorithm

1.1 General description

The Cholesky decomposition algorithm was first proposed by Andre-Louis Cholesky (October 15, 1875 - August 31, 1918) at the end of the First World War shortly before he was killed in battle. He was a French military officer and mathematician. The idea of this algorithm was published in 1924 by his fellow officer and, later, was used by Banachiewicz in 1938 [7]. In the Russian mathematical literature, the Cholesky decomposition is also known as the square-root method [1-3] due to the square root operations used in this decomposition and not used in Gaussian elimination.

Originally, the Cholesky decomposition was used only for dense real symmetric positive definite matrices. At present, the application of this decomposition is much wider. For example, it can also be employed for the case of Hermitian matrices. In order to increase the computing performance, its block versions are often applied.

In the case of sparse matrices, the Cholesky decomposition is also widely used as the main stage of a direct method for solving linear systems. In order to reduce the memory requirements and the profile of the matrix, special reordering strategies are applied to minimize the number of arithmetic operations. A number of reordering strategies are used to identify the independent matrix blocks for parallel computing systems.

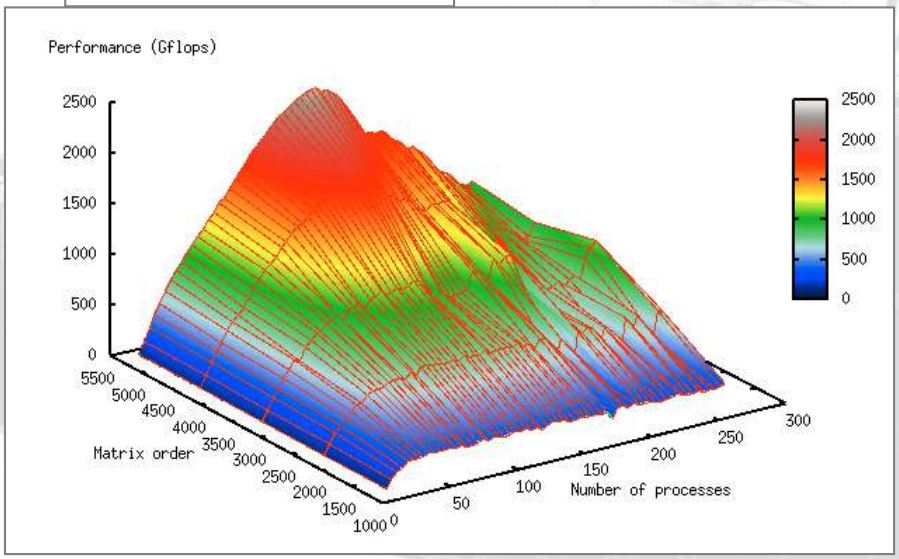
Summary

Properties of the algorithm:

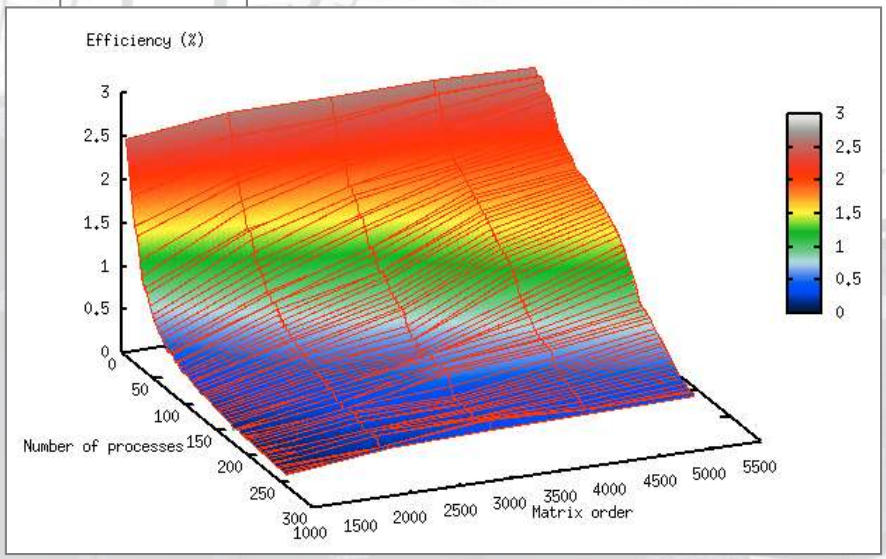
- Sequential complexity: $O(n^3)$
- Height of the parallel form: $O(n)$
- Width of the parallel form: $O(n^2)$
- Amount of input data: $\frac{n(n+1)}{2}$
- Amount of output data: $\frac{n(n+1)}{2}$

Description of Algorithms
(at the starting point: Cholesky decomposition)

Scalability (performance) *



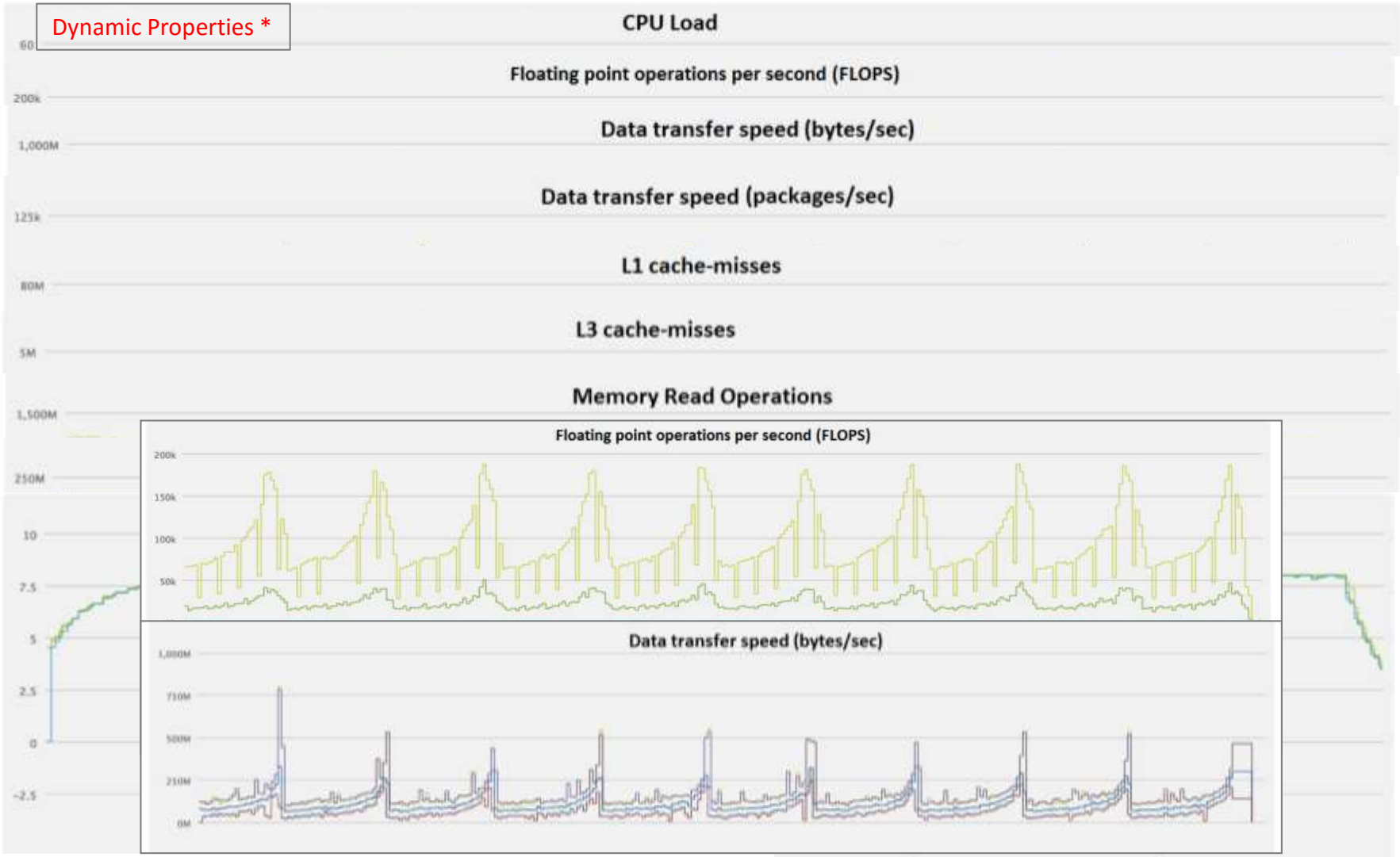
Efficiency *



* Scalability, performance, efficiency were measured on MSU “Lomonosov” Supercomputer

Description of Algorithms
(at the starting point: Cholesky decomposition)

Dynamic Properties *



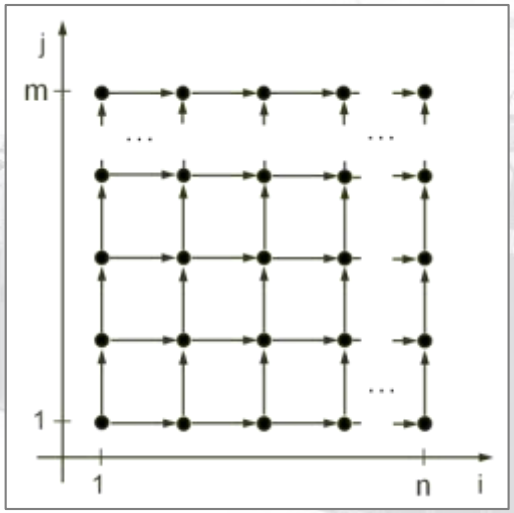
* Dynamic Properties were obtained on MSU “Lomonosov” Supercomputer

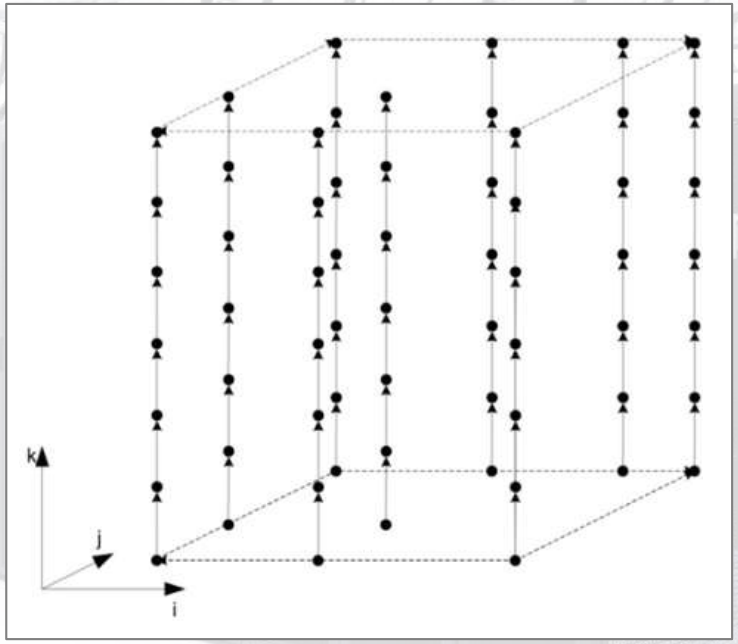
It is very useful information about the algorithm, we desperately need it.

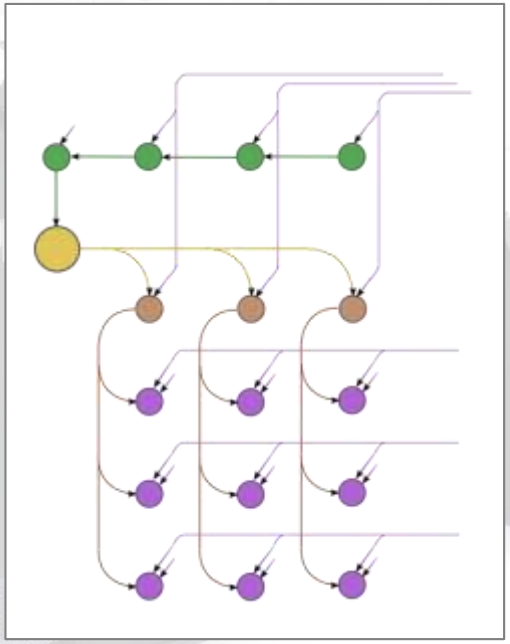
But… CHALLENGES are everywhere…

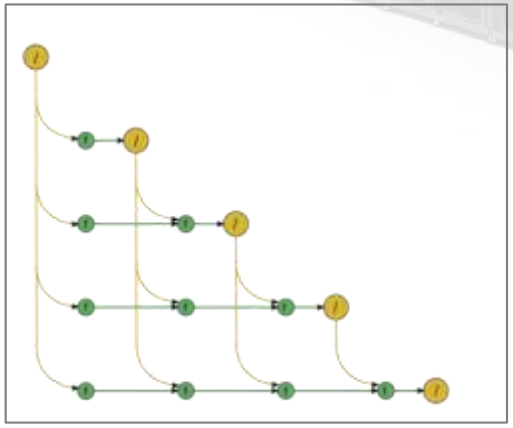
Information structure: how to extract, describe, show… ?

(challenges of the algorithm description)





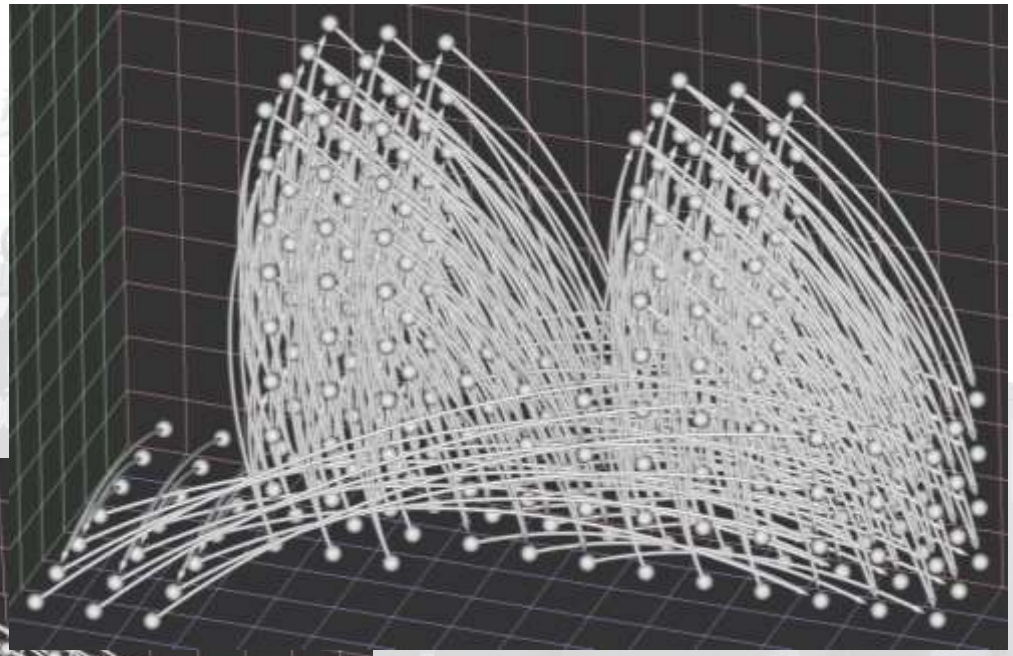


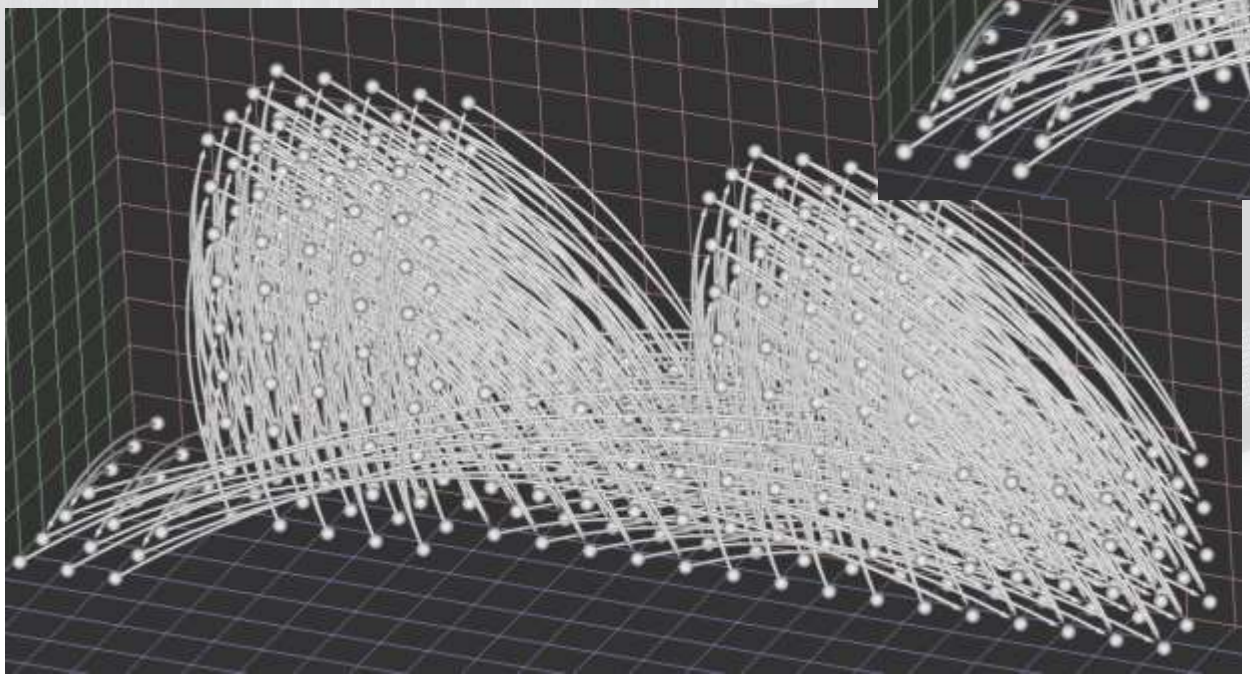


- *How to draw a potentially unlimited graph ?*
- *How to draw a potentially multidimensional graph ?*
- *How to show dependency of the graph on a problem size ?*

Information structure: how to extract, describe, show… ?

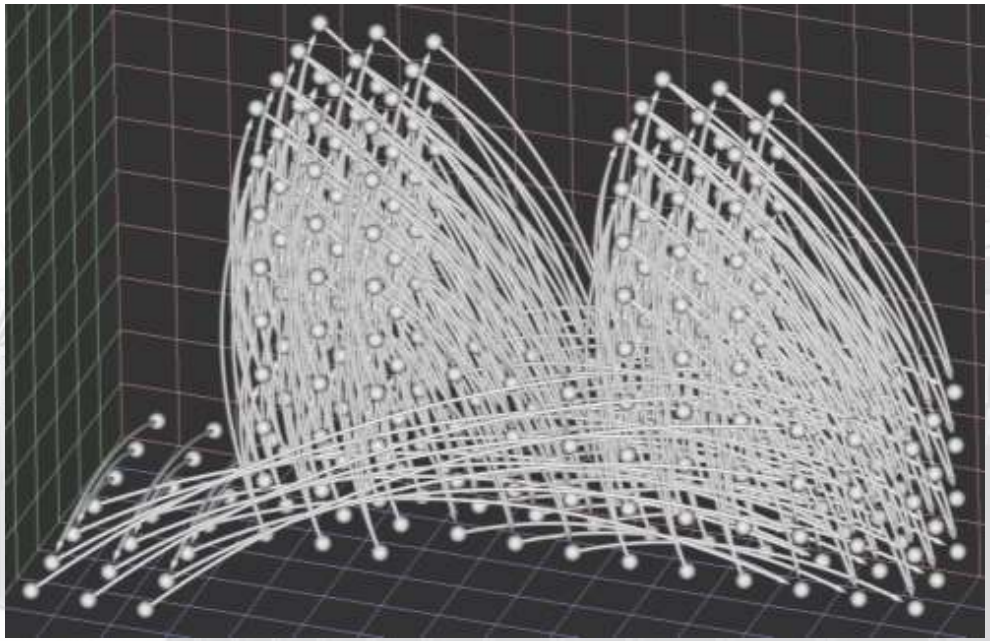
(challenges of the algorithm description)

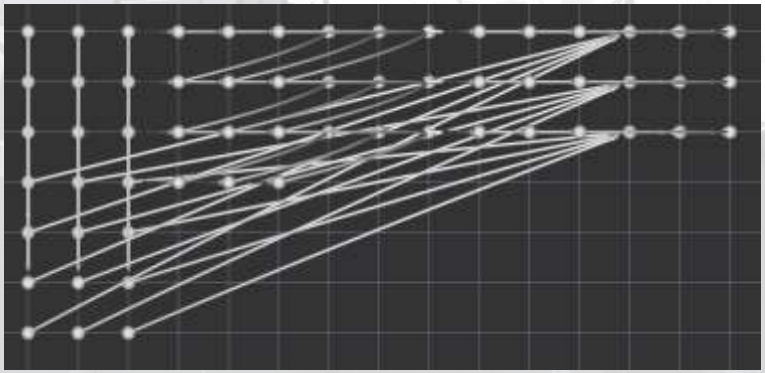


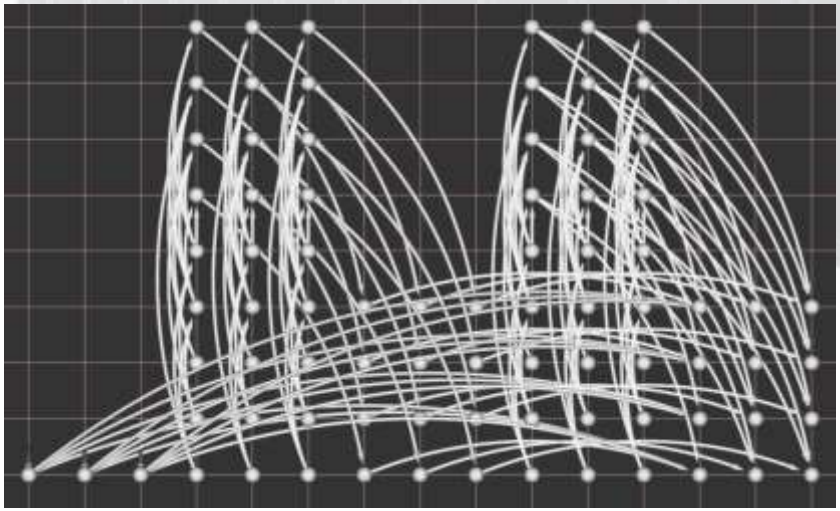


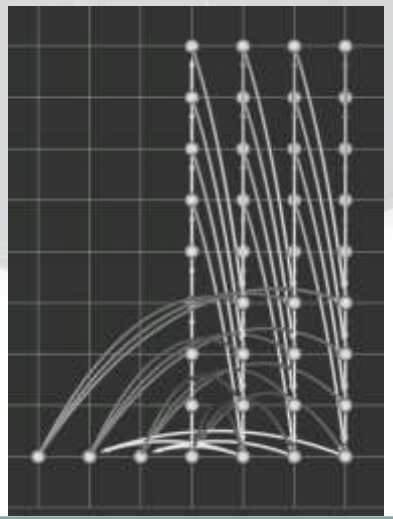
Information structure: how to extract, describe, show… ?

(challenges of the algorithm description)









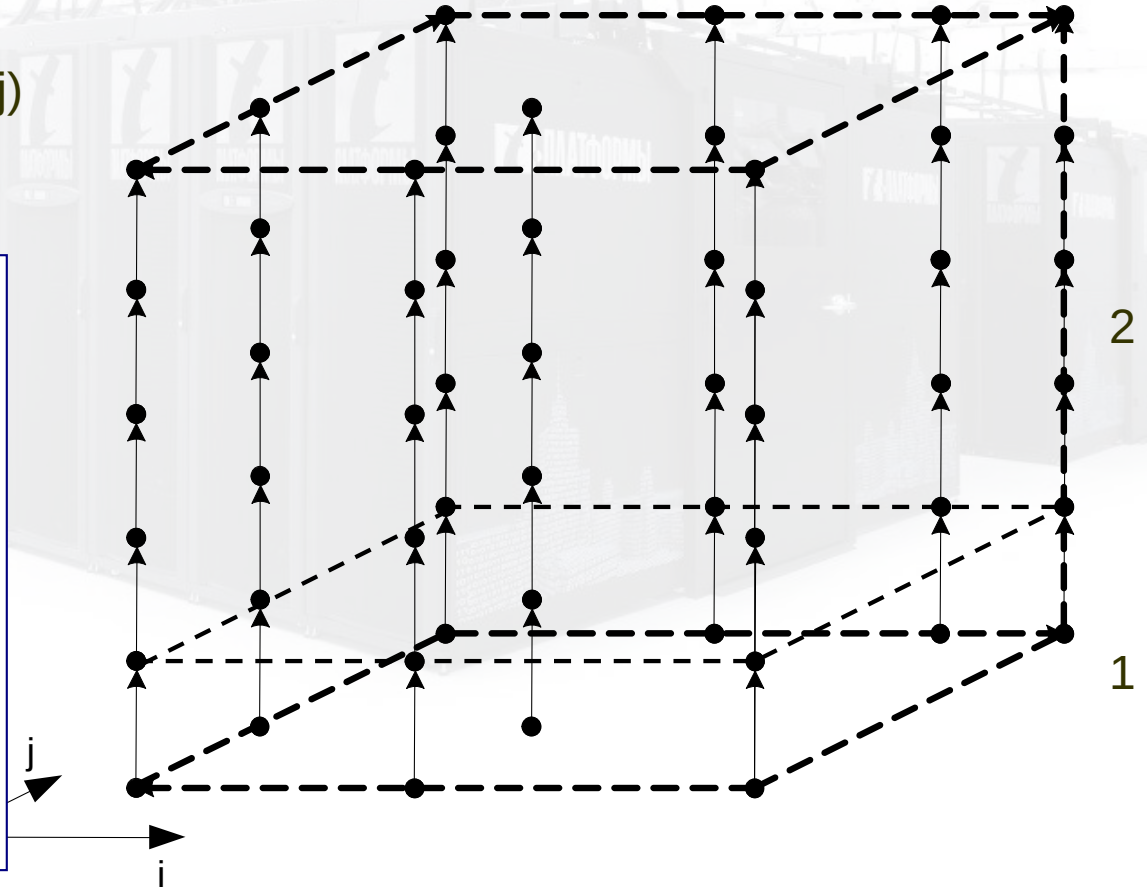
Information structure: how to extract, describe, show… ?

(challenges of the algorithm description)

```
    Do i = 1, n
       Do j = 1, n
1        A(i,j) = 0
          Do k = 1, n
2           A(i,j) = A(i,j) + B(i,k)*C(k,j)
```

How to formalize the information structure?
Polyhedra, inequalities, vector functions…

$$\begin{cases} 1 \le i \le n \\ 1 \le j \le n \\ k = 1 \\ \begin{cases} i_1 = i \\ j_1 = j \end{cases} \\ из\ (1) \end{cases} \qquad \begin{cases} 1 \le i \le n \\ 1 \le j \le n \\ 2 \le k \le n \\ \begin{cases} i_1 = i \\ j_1 = j \\ k_1 = k - 1 \end{cases} \\ из\ (2) \end{cases}$$

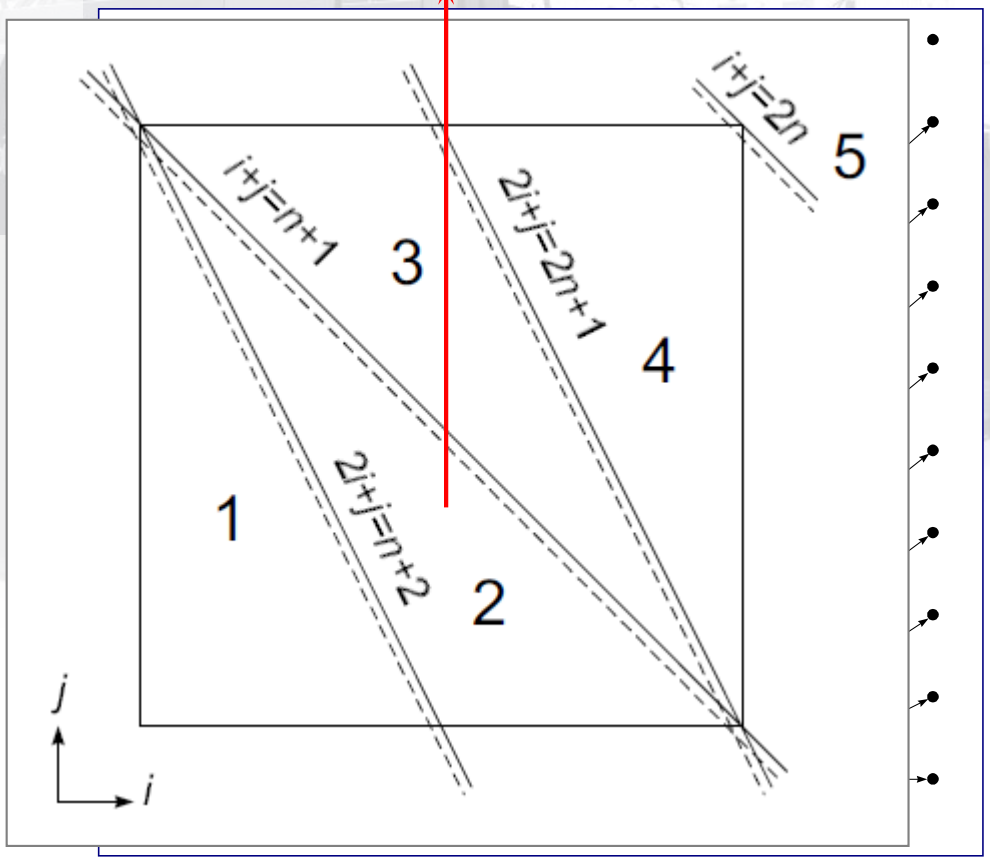


Information structure: how to extract, describe, show… ?

(challenges of the algorithm description)

```
DO  i = 1, n
      DO j = 1, n
            U( i + j ) = U( 2*n – i – j + 1)*q + p
      EndDO
EndDO
```

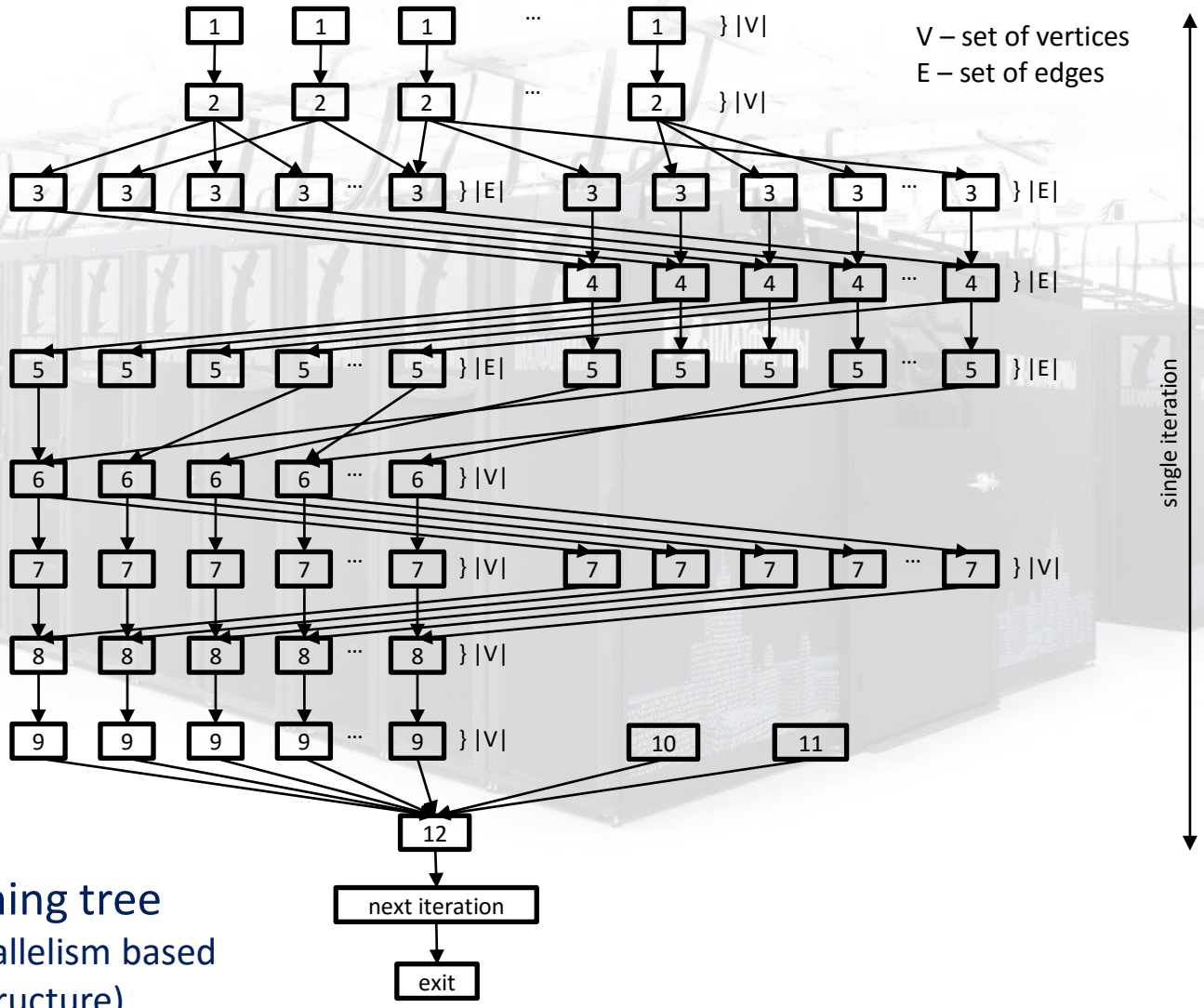
$$\begin{array}{r} 1 \le j; \\ i + j \le n; \\ n + 2 \le 2i + j. \end{array} \quad 2 \qquad \Phi_2 = \begin{pmatrix} 1 & 0 \\ -2 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2n+2 \end{pmatrix}$$




```
DO  i = 1, n
      DO  j = 1, n – i        Parallel loop !
            U( i + j ) = U( 2*n – i – j + 1)*q + p
      End DO
      DO j = n – i + 1, n     Parallel loop !
            U( i + j ) = U( 2*n – i – j + 1)*q + p
      End DO
End DO
```

Potential parallelism: how to extract, describe, show… ?

(challenges of the algorithm description)

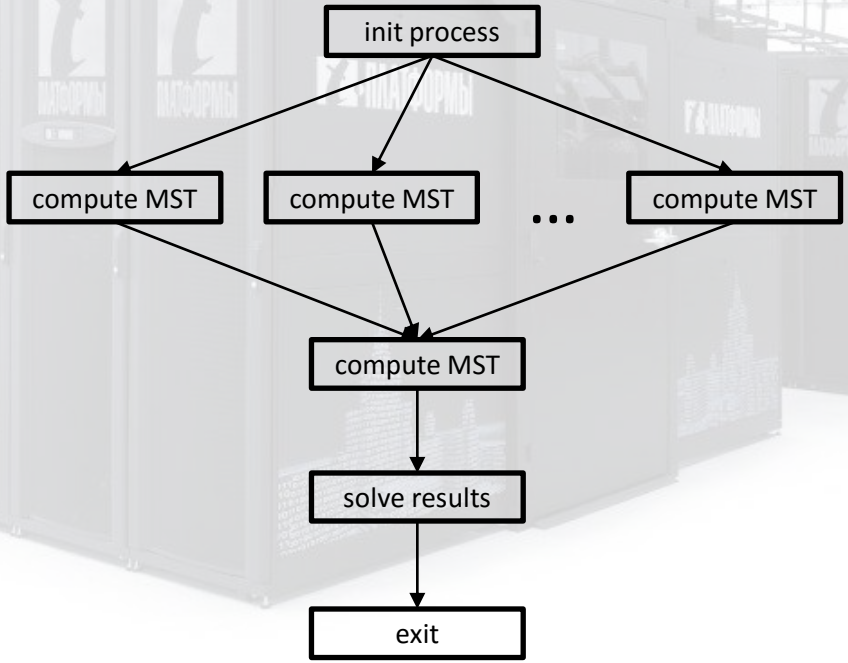


Minimum spanning tree

(resource of classic parallelism based on information structure)

Potential parallelism: how to extract, describe, show… ?
(challenges of the algorithm description)

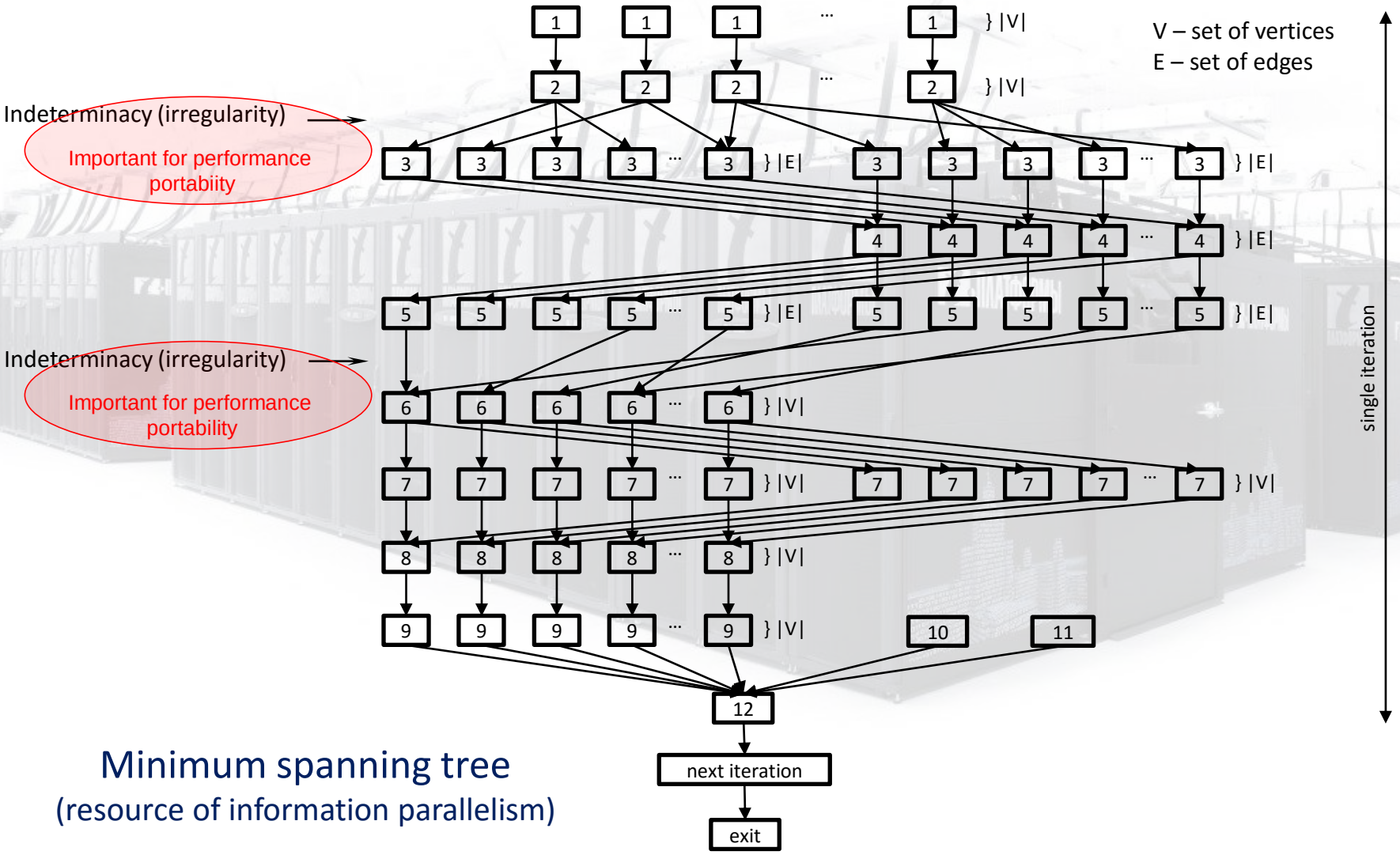
$$MST(E_1 \cup E_2) = MST\big(MST(E_1) \cup MST(E_2)\big),$$
$$E = E_1 \cup E_2$$



Minimum spanning tree
(resource of “mathematical” parallelism)

Potential parallelism: how to extract, describe, show… ?

(challenges of the algorithm description)



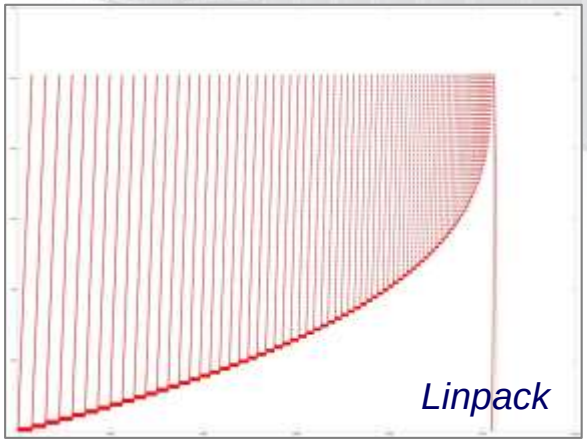
Minimum spanning tree

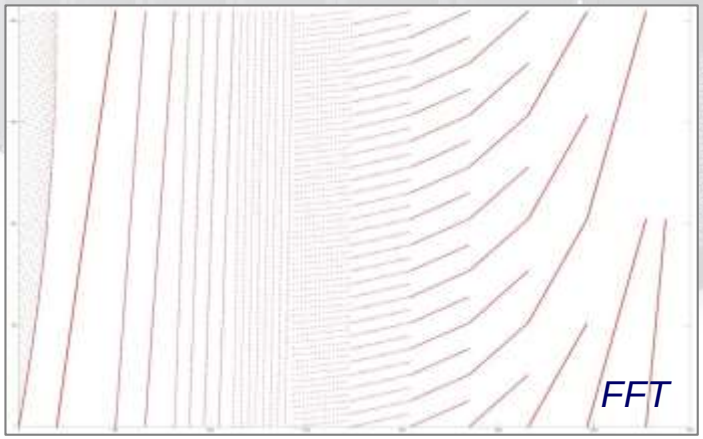
(resource of information parallelism)

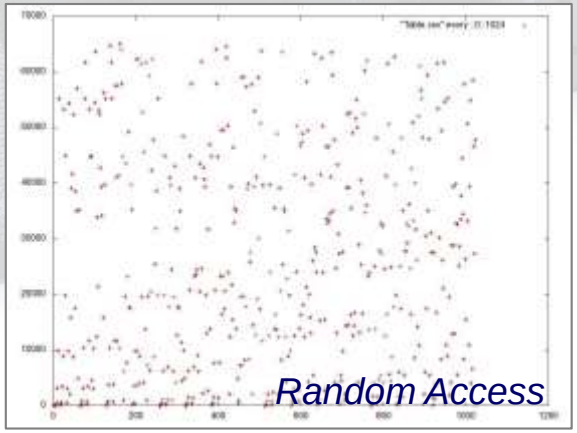
Data locality: a number of open questions
(challenges of the algorithm description)

How to evaluate spatial and temporal data locality of a program ?

How to compare spatial and temporal data locality of programs ?







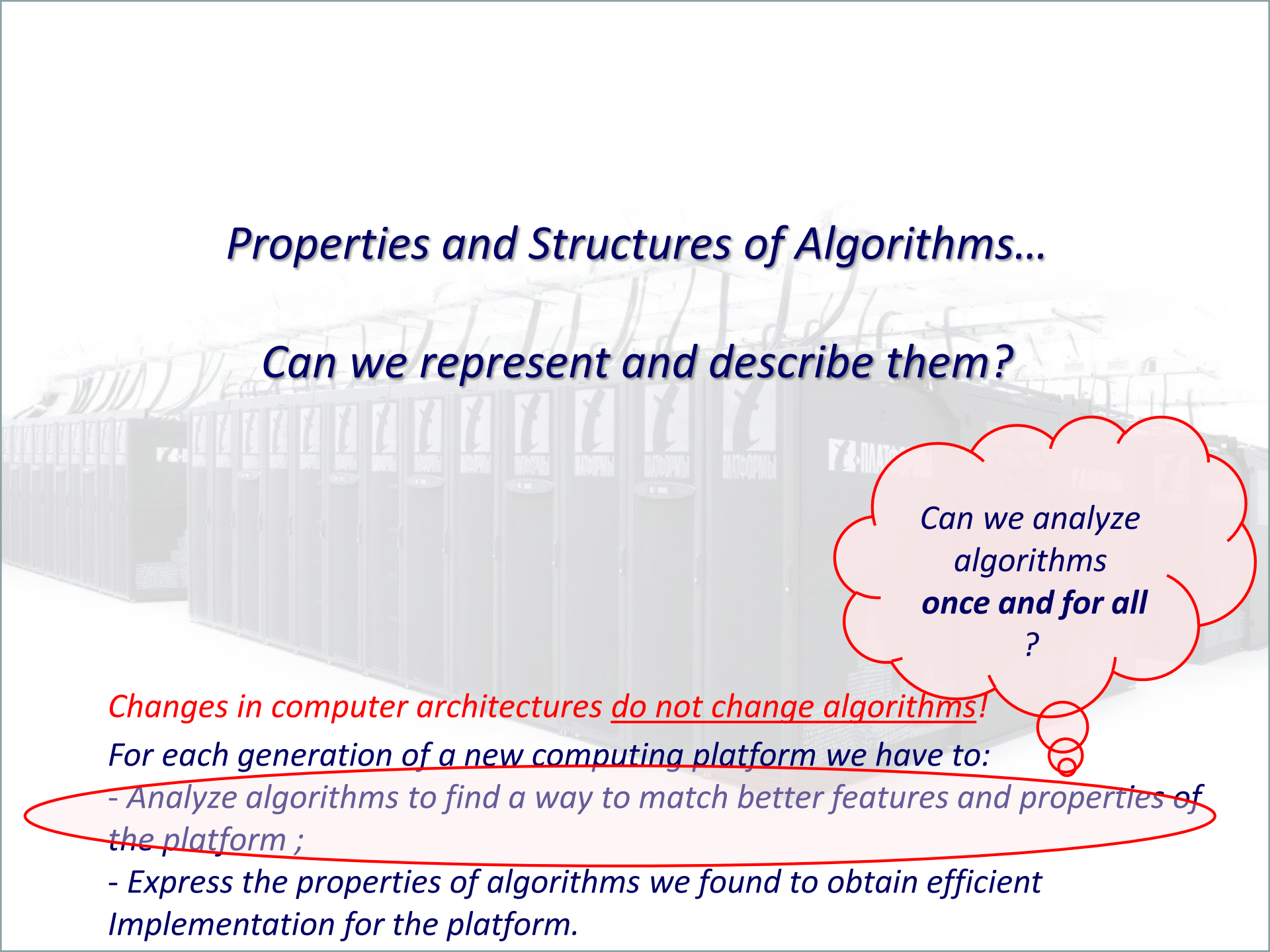
Data locality: a number of open questions
(challenges of the algorithm description)

How to evaluate spatial and temporal data locality of a program ?

How to compare spatial and temporal data locality of programs ?

Can we predict data locality in future implementations by using information from algorithms only ?

What does it mean “data locality of algorithm” ?
What does it mean “algorithm with good/bad locality” ?
There are no data structures in algorithms, locality can’t be applied to algorithms!
At the same time algorithms form the basis of programs...



Properties and Structures of Algorithms…

Can we represent and describe them?

Can we analyze algorithms ***once and for all*** *?*

Changes in computer architectures do not change algorithms!

For each generation of a new computing platform we have to:

- Analyze algorithms to find a way to match better features and properties of the platform ;

- Express the properties of algorithms we found to obtain efficient Implementation for the platform.

Properties and Structures of Algorithms…

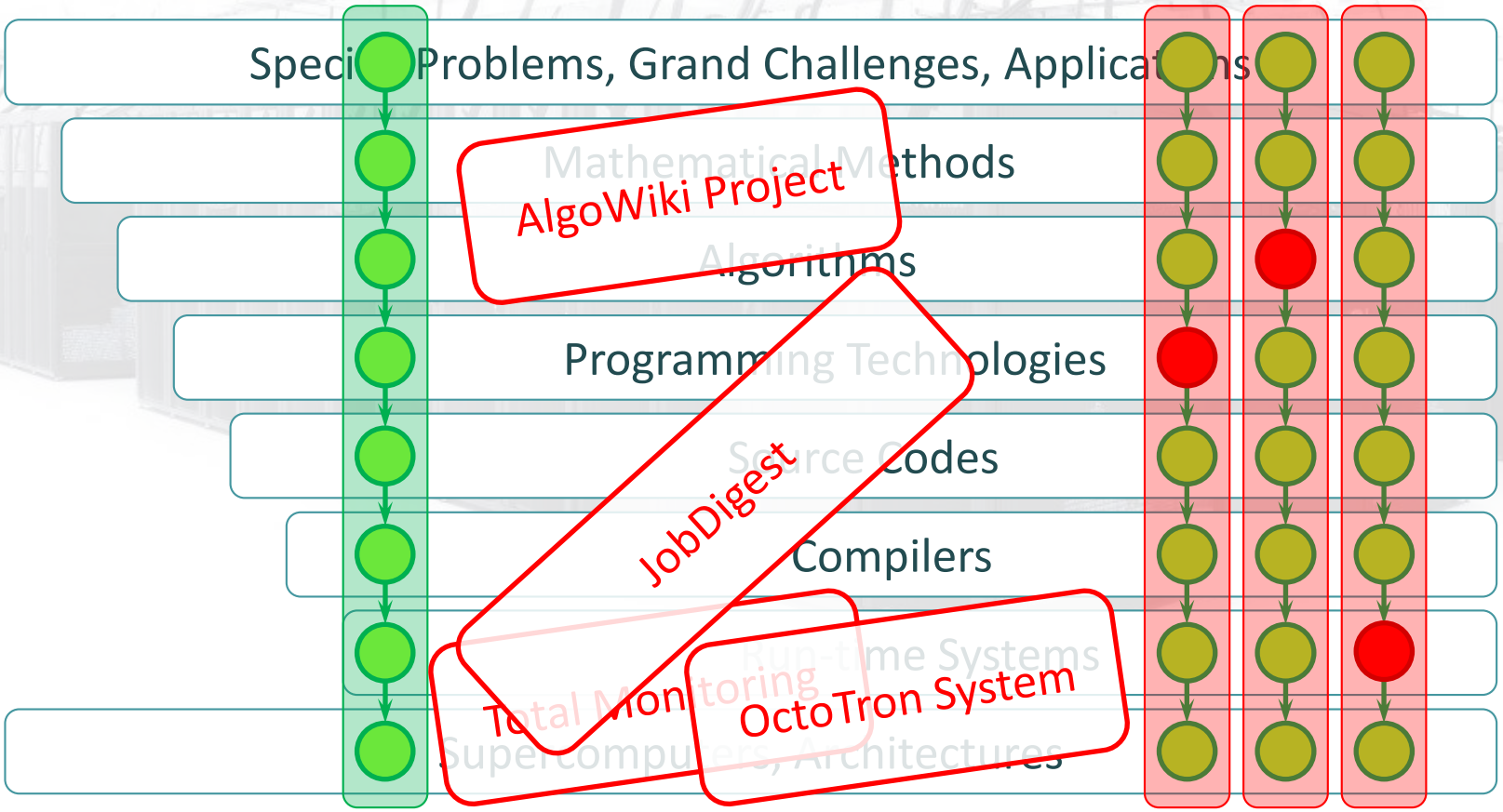
Yes, we can! **AlgoWiki**

http://AlgoWiki-Project.org

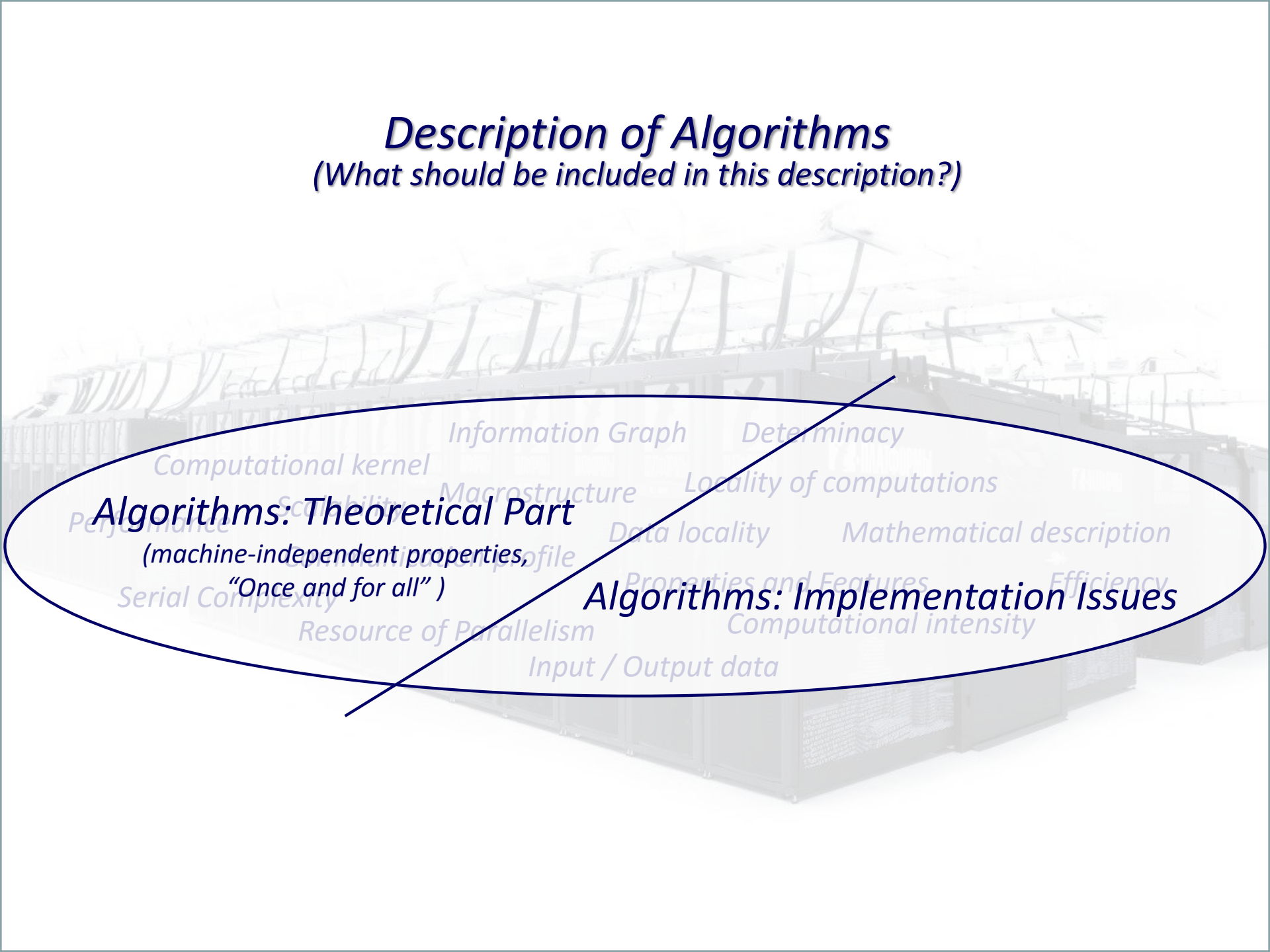
Can we analyze algorithms ***once and for all*** *?*

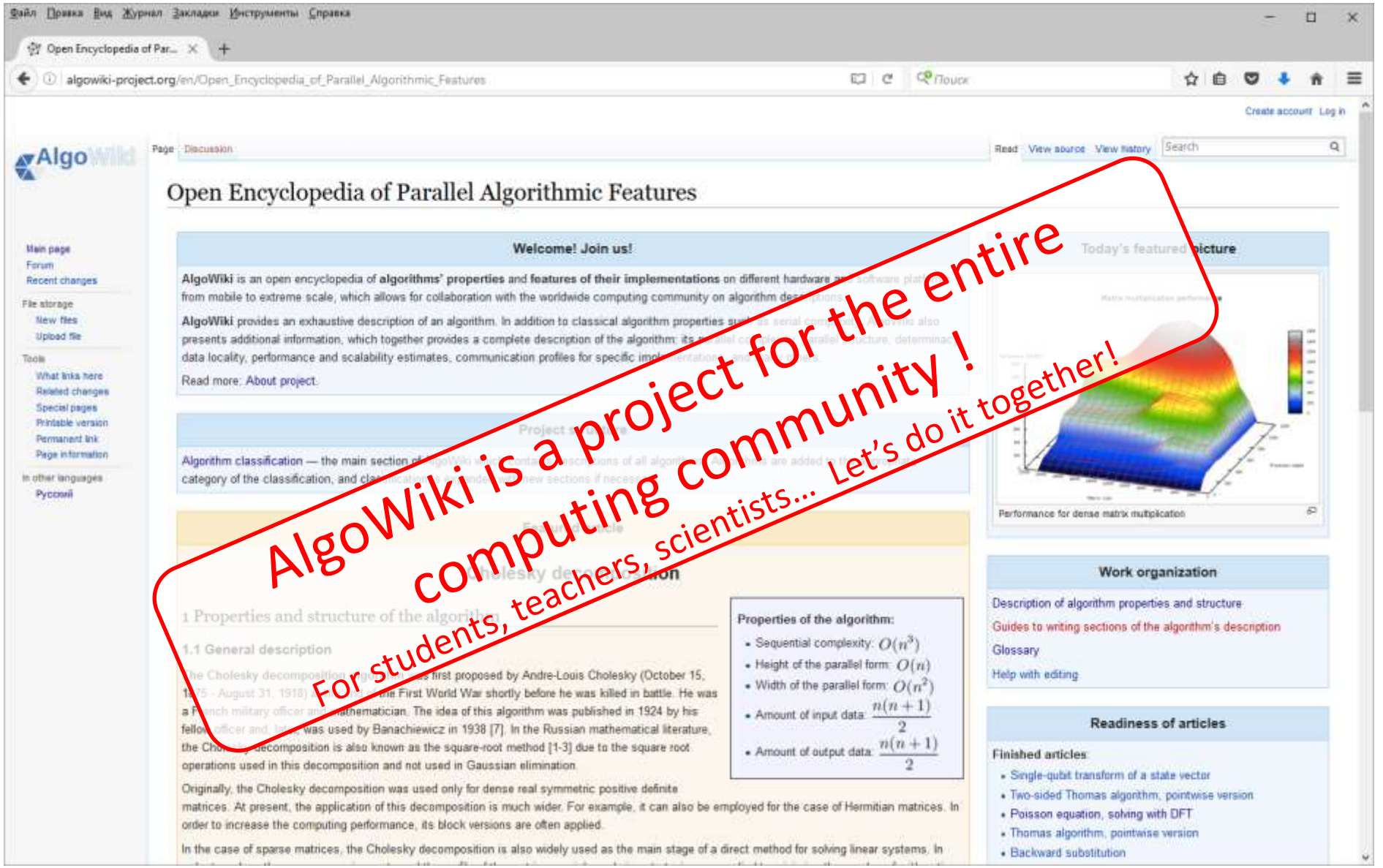
- Analyze algorithms to find a way to match better features and properties of the platform ;

Supercomputing Co-Design Technologies and Tools
(A practical approach)



Description of Algorithms
(What should be included in this description?)





AlgoWiki is a project for the entire computing community !

For students, teachers, scientists… Let's do it together!

Building Up Intelligible Parallel Computing World

A final remark…

Building Up Parallel Computing World

Problems and Challenges




Parallel Computing Technologies


Parallel Computing Education

Thank you!

Prof. Vladimir Voevodin
Deputy Director, Research Computing Center, MSU
Head of Department on Supercomputers and Quantum Informatics, CMC, MSU

voevodin@parallel.ru

February 6th, 2017, NARFU, Arkhangelsk

VIII International Youth Scientific School
"High-Performance Computing Using GRID Systems"

Благодарю за внимание!

Prof. Vladimir Voevodin
Deputy Director, Research Computing Center, MSU
Head of Department on Supercomputers and Quantum Informatics, CMC, MSU

voevodin@parallel.ru

February 6th, 2017, NARFU, Arkhangelsk