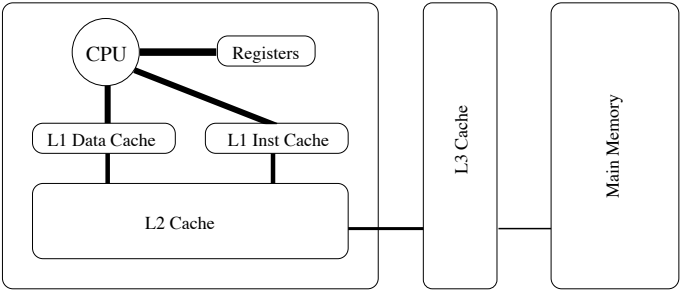


Fig. 1. A typical memory hierarchy containing two on–chip L1 caches, one on–chip L2 cache, and a third level of off–chip cache. The thickness of the interconnections illustrates the bandwidths between the memory hierarchy levels.

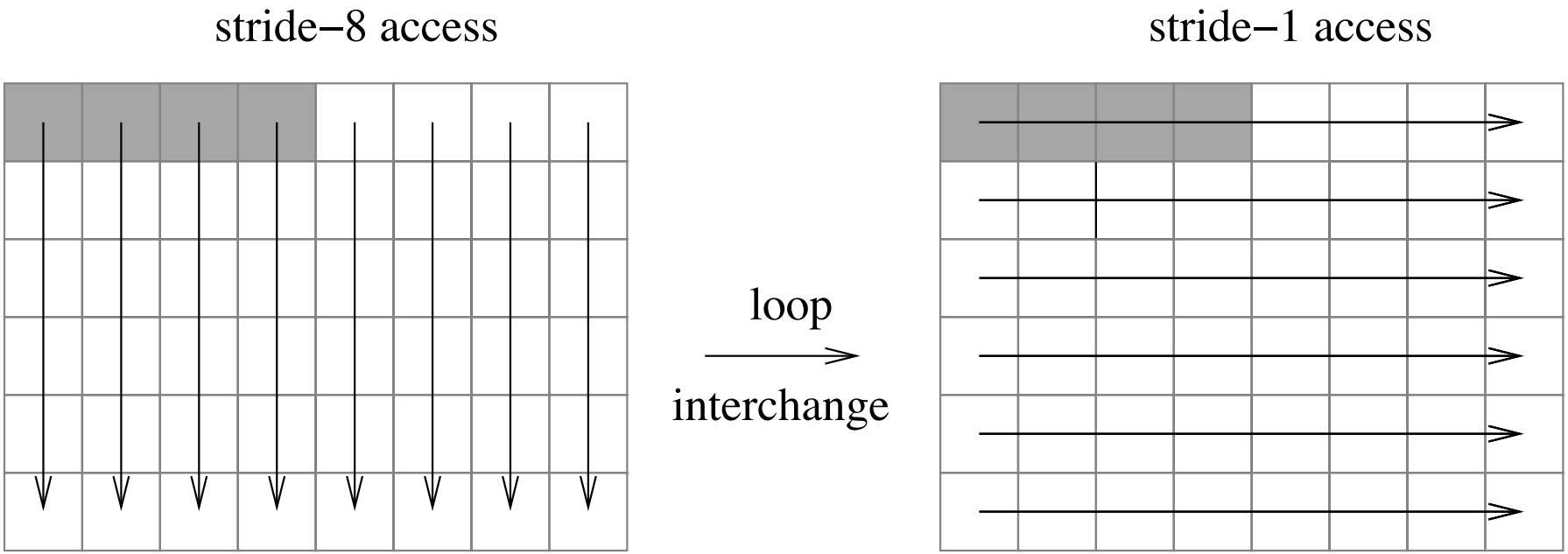


Fig. 2. Access patterns for interchanged loop nests.

Loop Interchange. This transformation reverses the order of two adjacent loops in a loop nest [2, 65]. Generally speaking, loop interchange can be applied if the order of the loop execution is unimportant. Loop interchange can be generalized to *loop permutation* by allowing more than two loops to be moved at once and by not requiring them to be adjacent.

Loop interchange can improve locality by reducing the *stride* of an array–based computation. The stride is the distance of array elements in memory

Algorithm 3.1 Loop interchange

```
1: double sum;
2: double a[n, n];
3: // Original loop nest:
4: for j = 1 to n do
5:    for i = 1 to n do
6:       sum+ = a[i, j];
7:    end for
8: end for
```

```
1: double sum;
2: double a[n, n];
3: // Interchanged loop nest:
4: for i = 1 to n do
5:    for j = 1 to n do
6:       sum+ = a[i, j];
7:    end for
8: end for
```

Loop Fusion. *Loop fusion* is a transformation which takes two adjacent loops that have the same iteration space traversal and combines their bodies into a single loop [17]. Loop fusion — sometimes also called *loop jamming* — is the inverse transformation of *loop distribution* or *loop fission* which breaks a single loop into multiple loops with the same iteration space. Loop fusion is legal as long as no flow, anti, or output dependencies in the fused loop exist for which instructions from the first loop depend on instructions from the second loop [2].

Fusing two loops results in a single loop which contains more instructions in its body and therefore offers increased instruction level parallelism. Furthermore, only one loop is executed, thus reducing the total loop overhead by approximately a factor of two.

Algorithm 3.2 Loop fusion

```
1: // Original code:
2: for i = 1 to n do
3:     b[i] = a[i] + 1.0;
4: end for
5: for i = 1 to n do
6:     c[i] = b[i] * 4.0;
7: end for
```

```
1: // After loop fusion:
2: for i = 1 to n do
3:     b[i] = a[i] + 1.0;
4:     c[i] = b[i] * 4.0;
5: end for
```

Loop Blocking. *Loop blocking* (also called *loop tiling*) is a loop transformation which increases the depth of a loop nest with depth n by adding additional loops to the loop nest. The depth of the resulting loop nest will be anything from $n+1$ to $2n$. Loop blocking is primarily used to improve data locality by enhancing the reuse of data in cache [2, 52, 64].

Algorithm 3.3 Loop blocking for matrix transposition

```
1: // Original code:
2: for i = 1 to n do
3:    for j = 1 to n do
4:       a[i, j] = b[j, i];
5:    end for
6: end for
```

```
1: // Loop blocked code:
2: for ii = 1 to n by B do
3:    for jj = 1 to n by B do
4:       for i = ii to min(ii + B − 1, n) do
5:          for j = jj to min(jj + B − 1, n) do
6:             a[i, j] = b[j, i];
7:          end for
8:       end for
9:    end for
10: end for
```

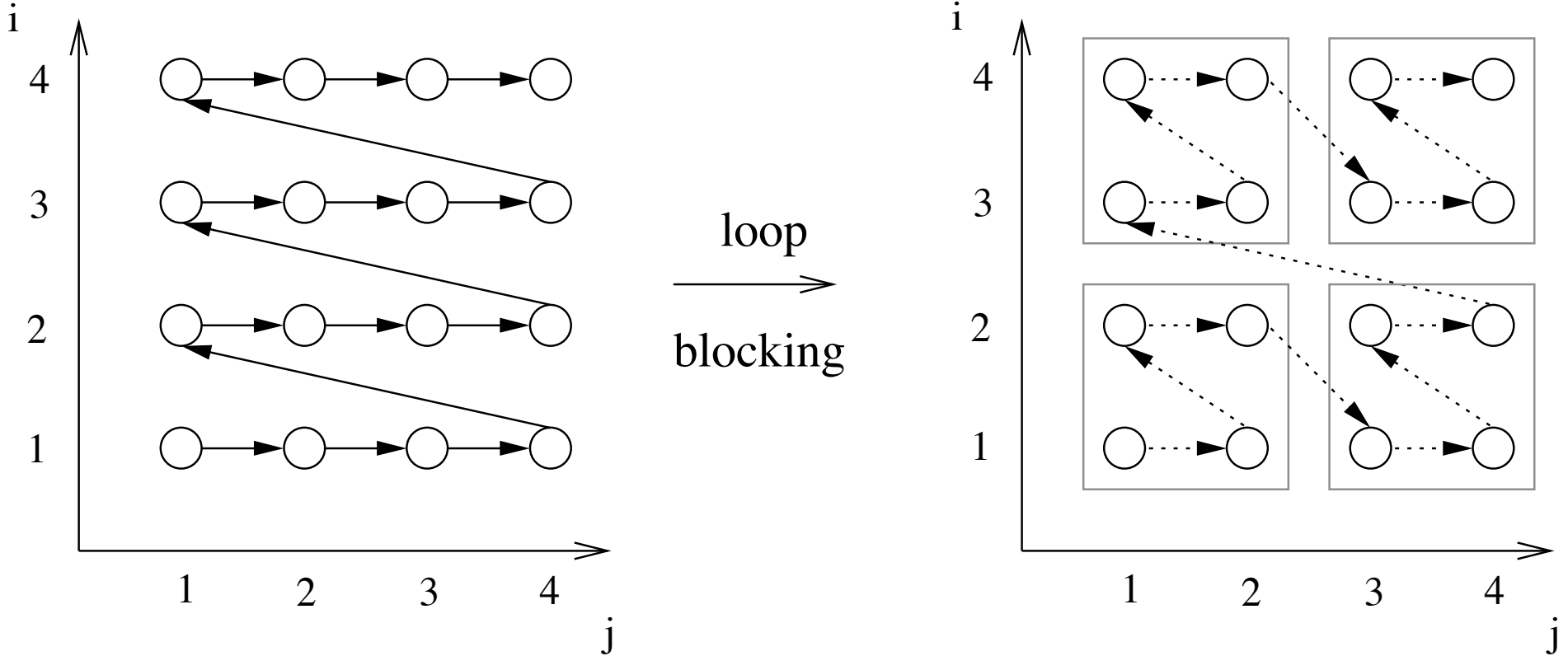


Fig. 3. Iteration space traversal for original and blocked code.

What Is LINPACK?

- LINPACK is a package of mathematical software for solving problems in linear algebra, mainly dense linear systems of linear equations.
- LINPACK: "LINear algebra PACKage"
  - Written in Fortran 66
- The project had its origins in 1974
- The project had four primary contributors: myself when I was at Argonne National Lab, Jim Bunch from the University of California-San Diego, Cleve Moler who was at New Mexico at that time, and Pete Stewart from the University of Maryland.
- LINPACK as a software package has been largely superseded by LAPACK, which has been designed to run efficiently on shared-memory, vector supercomputers.

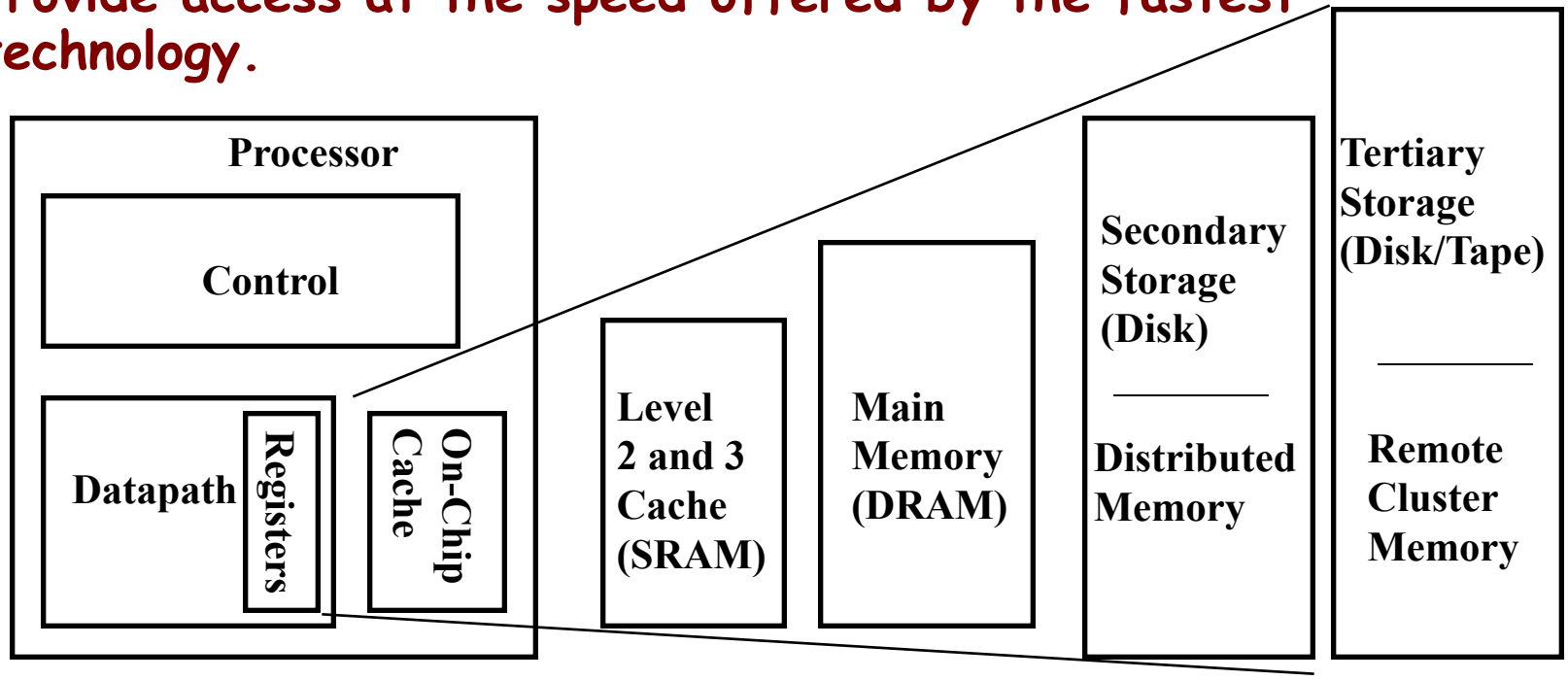
LINPACK Benchmark?

- The Linpack Benchmark is a measure of a computer's floating-point rate of execution.
  - It is determined by running a computer program that solves a dense system of linear equations.
- Over the years the characteristics of the benchmark has changed a bit.
  - In fact, there are three benchmarks included in the Linpack Benchmark report.
- LINPACK Benchmark
  - Dense linear system solve with LU factorization using partial pivoting
  - Operation count is: $2/3\ n^3 + O(n^2)$
  - Benchmark Measure: MFlop/s
  - Original benchmark measures the execution rate for a Fortran program on a matrix of size 100x100.

Memory Hierarchy

By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

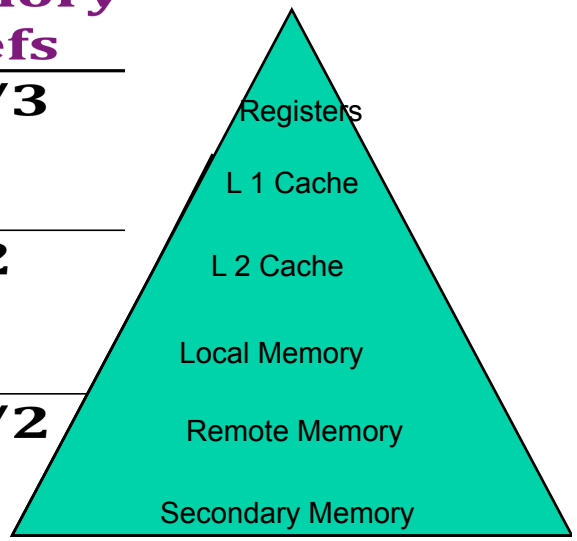


Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	100,000 s (.1s ms) Gs	10,000,000 s (10s ms) Ts

Why Higher Level BLAS?

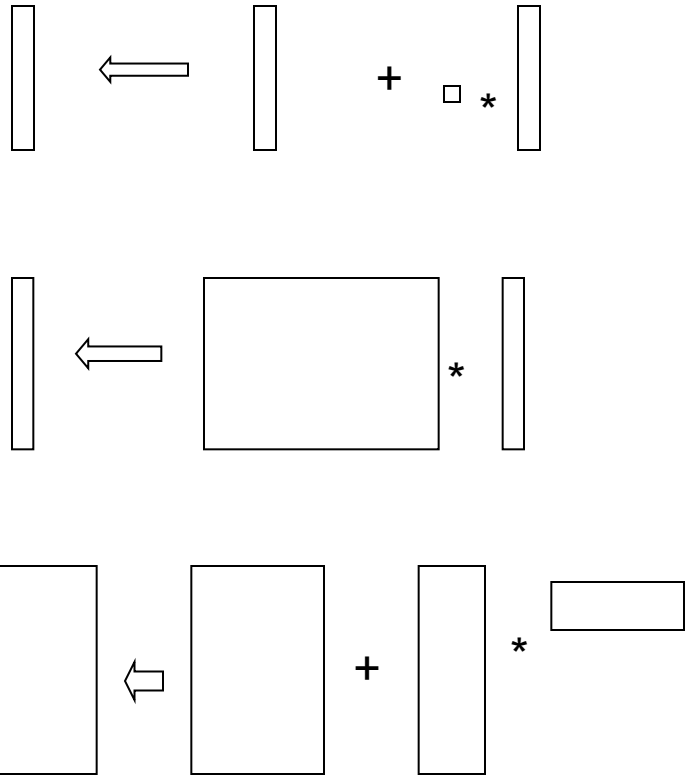
- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

BLAS	Memory Refs	Flops	Flops/ Memory Refs
Level 1 $y=y+\alpha x$	$3n$	$2n$	$2/3$
Level 2 $y=y+Ax$	n^2	$2n^2$	2
Level 3 $C=C+AB$	$4n^2$	$2n^3$	$n/2$



Level 1, 2 and 3 BLAS

- Level 1 BLAS
  Vector-Vector operations
- Level 2 BLAS
  Matrix-Vector operations
- Level 3 BLAS
  Matrix-Matrix operations



A brief history of (Dense) Linear Algebra software

LAPACK – "Linear Algebra PACKage" - uses BLAS-3 (1989 – now)

- Ex: Obvious way to express Gaussian Elimination (GE) is adding multiples of one row to other rows – BLAS-1
  - How do we reorganize GE to use BLAS-3 ? (details later)
- Contents of LAPACK (summary)
  - Algorithms we can turn into (nearly) 100% BLAS 3
    - Linear Systems: solve $Ax=b$ for x
    - Least Squares: choose x to minimize $||Ax - b||_2$
  - Algorithms that are only 50% BLAS 3 (so far)
    - "Eigenproblems": Find λ and x where $Ax = \lambda x$
    - Singular Value Decomposition (SVD): $(A^TA)x=\sigma^2x$
  - Generalized problems (eg $Ax = \lambda Bx$)
  - Error bounds for everything
  - Lots of variants depending on A's structure (banded, $A=A^T$, etc)
- How much code? (Release 3.4, Nov 2011) (www.netlib.org/lapack)
  - Source: 1674 routines, 490K LOC, Testing: 448K LOC

A brief history of (Dense) Linear Algebra software

- **Is LAPACK parallel?**
  - Only if the BLAS are parallel (possible in shared memory)
- **ScaLAPACK – "Scalable LAPACK" (1995 – now)**
  - For distributed memory – uses MPI
  - More complex data structures, algorithms than LAPACK
    - Only (small) subset of LAPACK's functionality available
  - All at www.netlib.org/scalapack

LAPACK

- http://www.netlib.org/lapack/

- LAPACK (Linear Algebra Package) provides routines for
  - solving systems of simultaneous linear equations,
  - least-squares solutions of linear systems of equations,
  - eigenvalue problems,
  - and singular value problems.

- LAPACK relies on BLAS

- The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers.

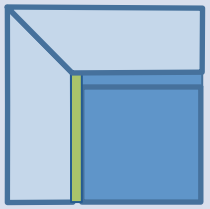
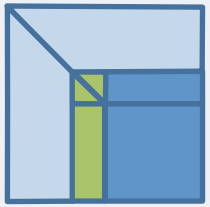
- Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

LAPACK is in FORTRAN Column Major

LAPACK is SEQUENTIAL

LAPACK is a REFERENCE implementation

A new generation of algorithms?

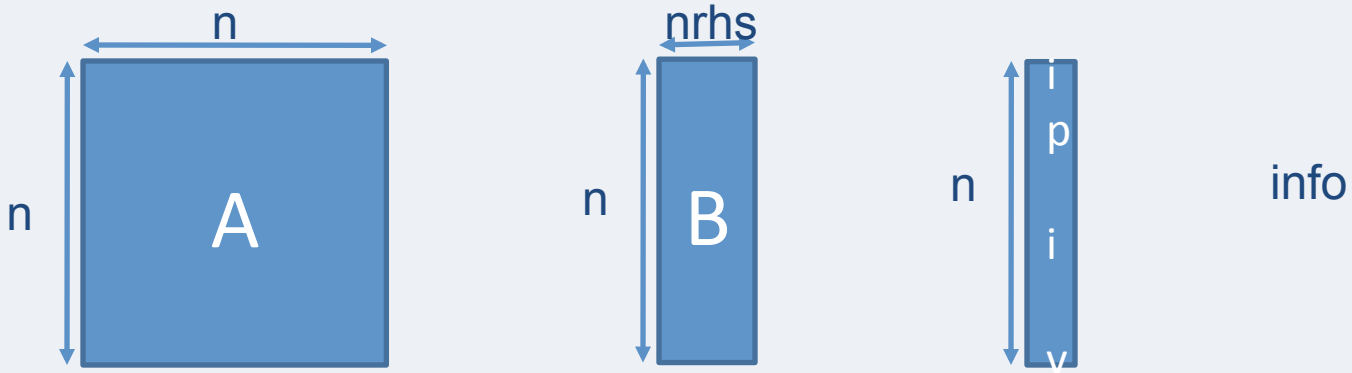
Algorithms follow hardware evolution along time.		
LINPACK (80's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (90's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations

Example with GESV

Solve a system of linear equations using a LU factorization

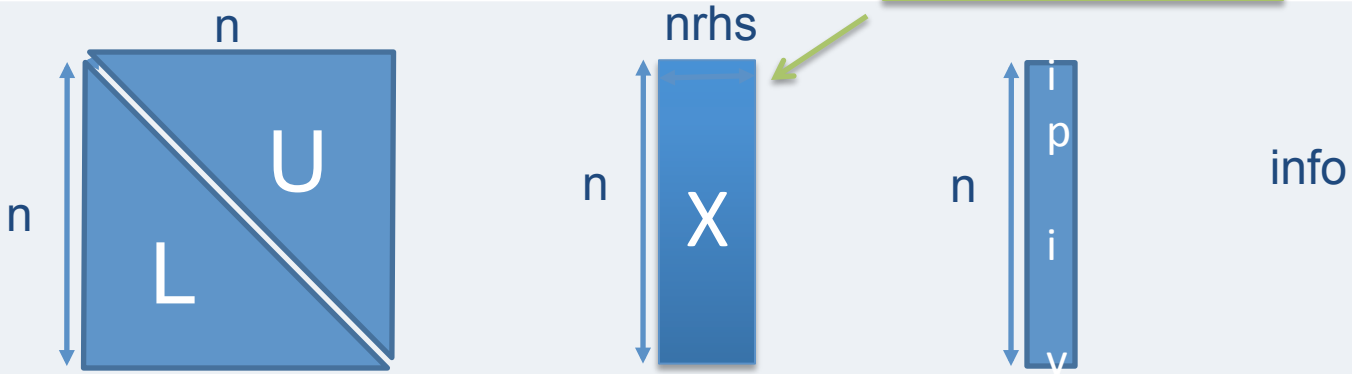
subroutine dgesv(n, nrhs, A, lda, ipiv, b, ldb, info)

input:



output:

Solution of Ax=b



Functionalities in LAPACK

Type of Problem	Acronyms
Linear system of equations	SV
Linear least squares problems	LLS
Linear equality-constrained least squares problem	LSE
General linear model problem	GLM
Symmetric eigenproblems	SEP
Nonsymmetric eigenproblems	NEP
Singular value decomposition	SVD
Generalized symmetric definite eigenproblems	GSEP
Generalized nonsymmetric eigenproblems	GNEP
Generalized (or quotient) singular value decomposition	GSVD (QSVD)