

GPU Hardware & LA

Hatem Ltaief

Principal Research Scientist, KAUST

*Advanced School on High-Performance Computing and
Applied AI for High-Resolution Regional Climate Modeling*

September 8-19 2025



**College of
Computing**



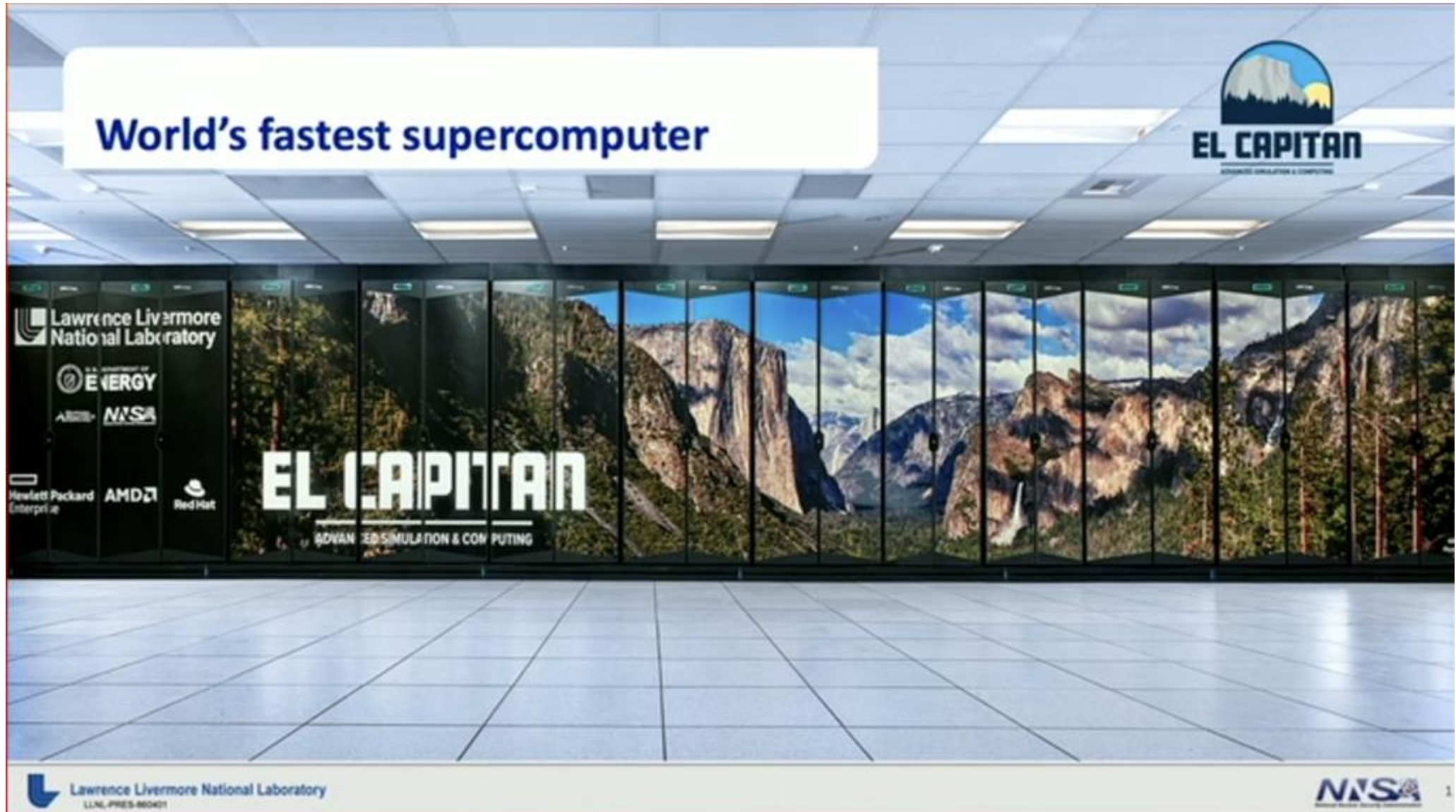
The Abdus Salam
**International Centre
for Theoretical Physics**

Nov'24 Top500 Highlights

- El Capitan becomes #1 and third ExaScale system ever
 - Frontier is #2 and Aurora #3
 - These three system are installed at different DOE laboratories
- Aurora and Frontier achieve over 10 Exaflop/s with mixed precision (MxP)
- 3 new systems in TOP10 (#1 El Capitan, #5 HPC6, #10 Tuolumne)
- 8 NVIDIA Grace Hopper systems on the list
- 6 NVIDIA Grace Hopper and 3 AMD Zen-4 Genoa systems in top10 of Green500
- HPC systems are used longer and replaced less often
 - Technological limits lead to strong concentration at the top
- TOP500 shows further reduced growth-rates since 2017
 - End of Moore's Law -Very unlikely to achieve 10 Exascale(HPL) by 2030
 - However: New Workloads (AI) require new benchmarks!
 - Chinese systems are missing in the equation



Welcome to El Capitan, the new crown!



Welcome to El Capitan, the new crown!

HPE has delivered a highly capable
AMD GPU-accelerated system



- System specifications:
 - Peak 2.7929 DP exaflops
 - Per node peak of 250.8 DP teraflops
 - ~42.3 FP8 exaflops
 - Peak power 34.8 MW
 - AMD MI300A APU - 3D chiplet design w/AMD CDNA 3 GPU, “Zen 4” CPU, cache memory, HBM3
 - Slingshot interconnect

- HPE has provided several critical innovations
 - HPE and LLNL have worked with ORNL jointly on non-recurring engineering (NRE) activities
 - MI300A, world’s first data center APU directly addresses multiple challenges
 - Uses TOSS software stack, enhanced with HPE software
 - El Capitan includes an innovative near node local storage solution: the “Rabbits”

Late binding of the processor solution has ensured El Capitan provides the best possible value

Under the hood...

AMD INSTINCT™ MI300A: The world's first data center APU



- 4th Gen AMD Infinity Architecture:
AMD CDNA™ 3 and EPYC™ CPU “Zen 4” together
 - CPU cores and GPU compute units share a unified on-package pool of memory
- Groundbreaking 3D packaging
 - CPU | GPU | Cache | HBM
 - 24 Zen4 cores, 146B transistors, 128GB HBM3
- Designed for leadership memory bandwidth and application latency
- APU architecture designed for power savings
 - compared to discrete implementation



The Beast

AMD Instinct™ MI300 Modular Chiplet Package

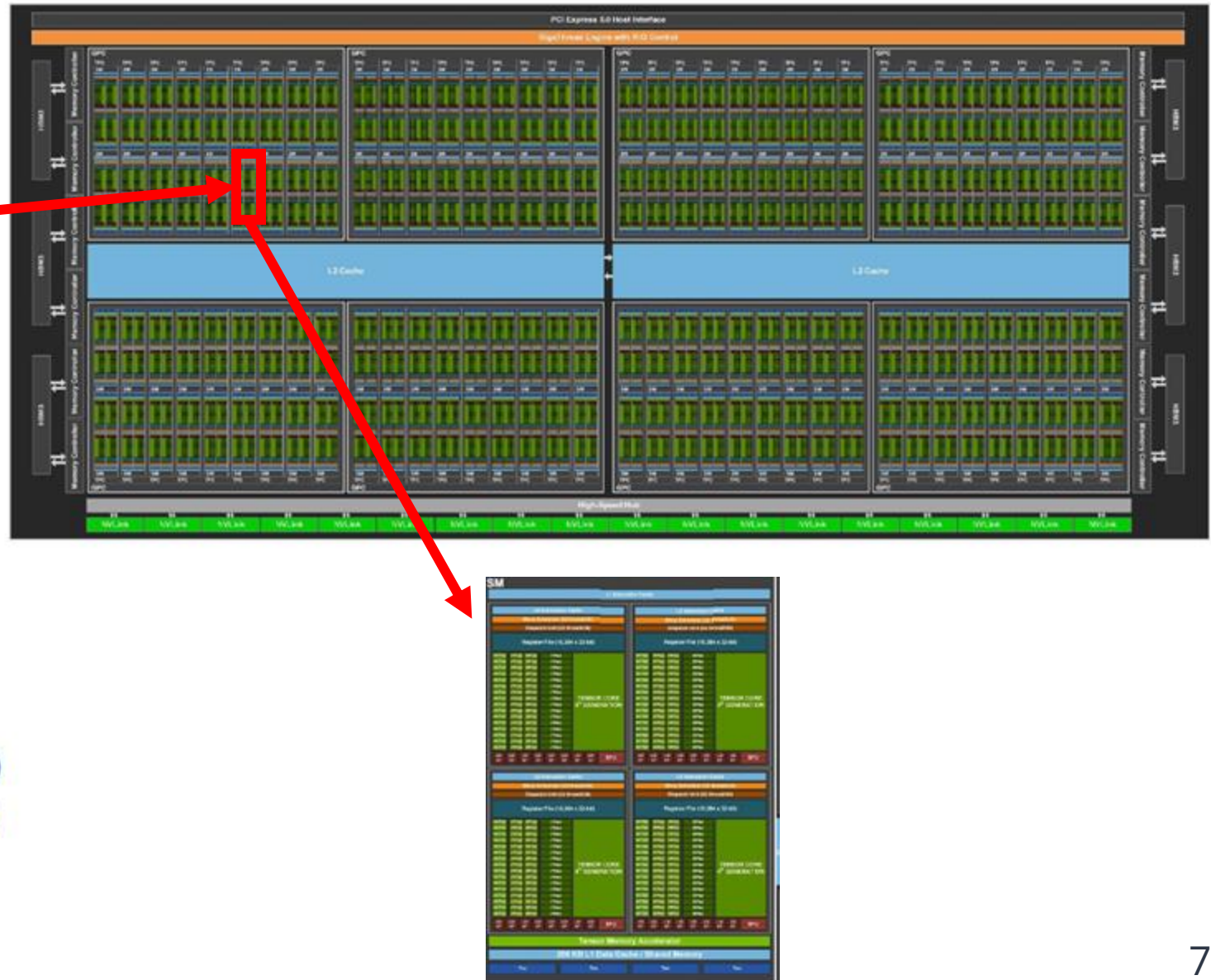


-
- I/O Die (IOD)
 - 128 Channel HBM3 Interface
 - 256MB AMD Infinity Cache™
 - Infinity Fabric Network-on-Chip
 - 4 x16 PCIe® 5 + 4th Gen Infinity Fabric™ Links
 - 4 x16 4th Gen Infinity Fabric™ Links
 - AMD Infinity Fabric™ AP Interconnect
 - HBM3
 - 8 physical stacks
 - AMD Instinct™ MI300A: 128 GB (8H)
 - AMD Instinct™ MI300X: 192 GB (12H)
 - Accelerator Complex Die (XCD)
 - 6 x 38 AMD CDNA™ 3 Compute Units
 - CPU Complex Die (CCD)
 - 3 x 8 “Zen 4” Cores
 - Replaced by additional XCDs on MI300X
 - 3.5D Package
 - 3D hybrid bonding
 - 2.5D silicon interposer

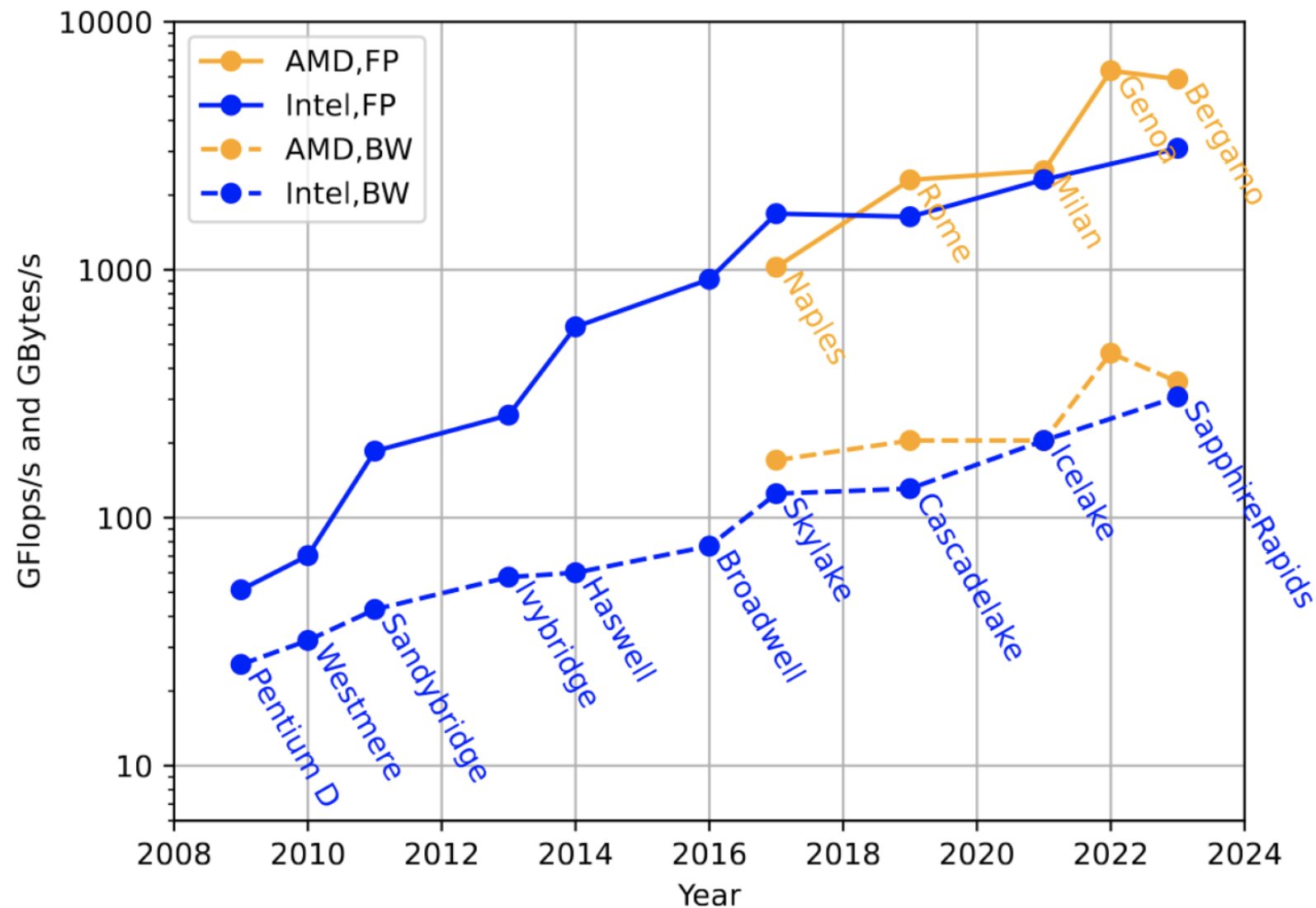
Nvidia H100 “Hopper” SXM5 specs

Architecture

- 80 B Transistors
- ~ 1.8 GHz clock speed
- ~ 144 “SM” units
 - 128 SP “cores” each (FMA)
 - 64 DP “cores” each (FMA)
 - 4 “Tensor Cores” each
 - 2:1 SP:DP performance
- ~ 34 TFlop/s DP peak (FP64)
- 50 MiB L2 Cache
- 80 GB HBM3
- MemBW ~ 3300 GB/s (theoretical)
- MemBW ~ 3000 GB/s (measured)

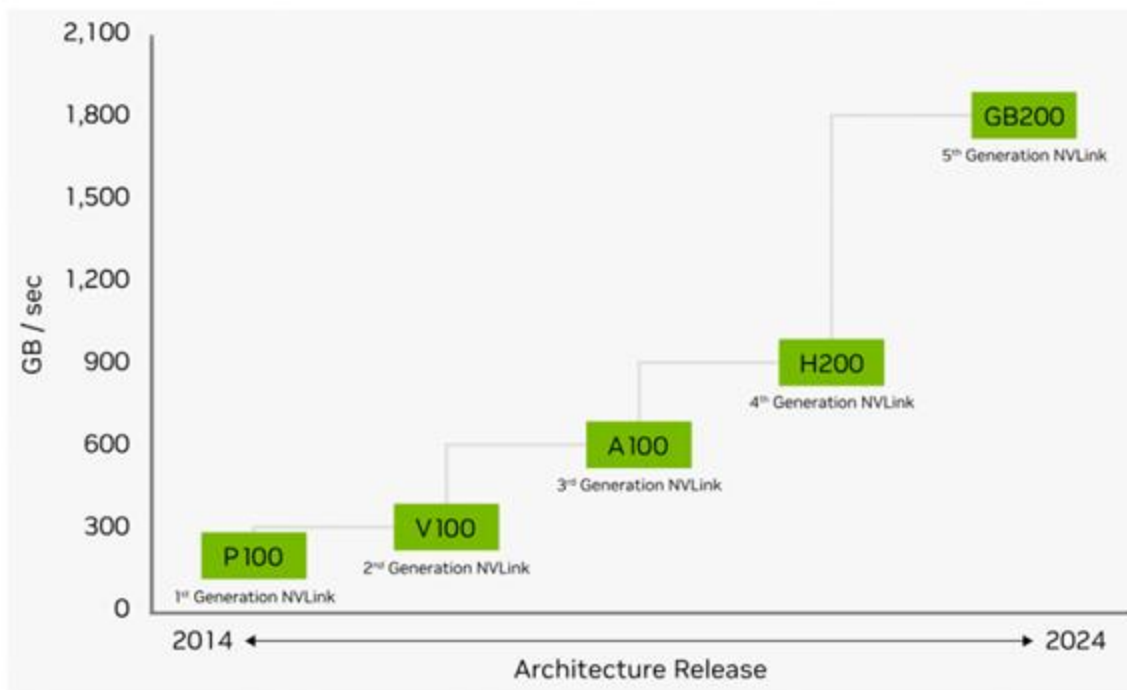


But we have a major bottleneck



But we have a major bottleneck

- High-performance interconnect for emerging dense GPU systems
 - Allows Load-Store operations between all GPUs



| | Second Generation | Third Generation | Fourth Generation | Fifth Generation |
|---------------------------------|----------------------------|----------------------------|-----------------------------|-------------------------------|
| NVLink bandwidth per GPU | 300GB/s | 600GB/s | 900GB/s | 1,800GB/s |
| Maximum Number of Links per GPU | 6 | 12 | 18 | 18 |
| Supported NVIDIA Architectures | NVIDIA Volta™ architecture | NVIDIA Ampere architecture | NVIDIA Hopper™ architecture | NVIDIA Blackwell architecture |

NVLink Performance Trends

Courtesy: [NVIDIA](#)

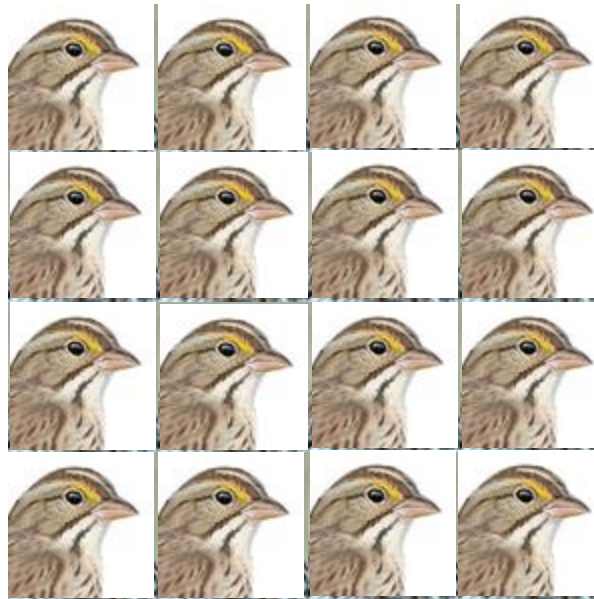
Specialization: Nature's way of Extracting More Performance in Resource Limited Environment

Powerful General Purpose



Xeon, Power

Many Lighter Weight
(post-Dennard scarcity)



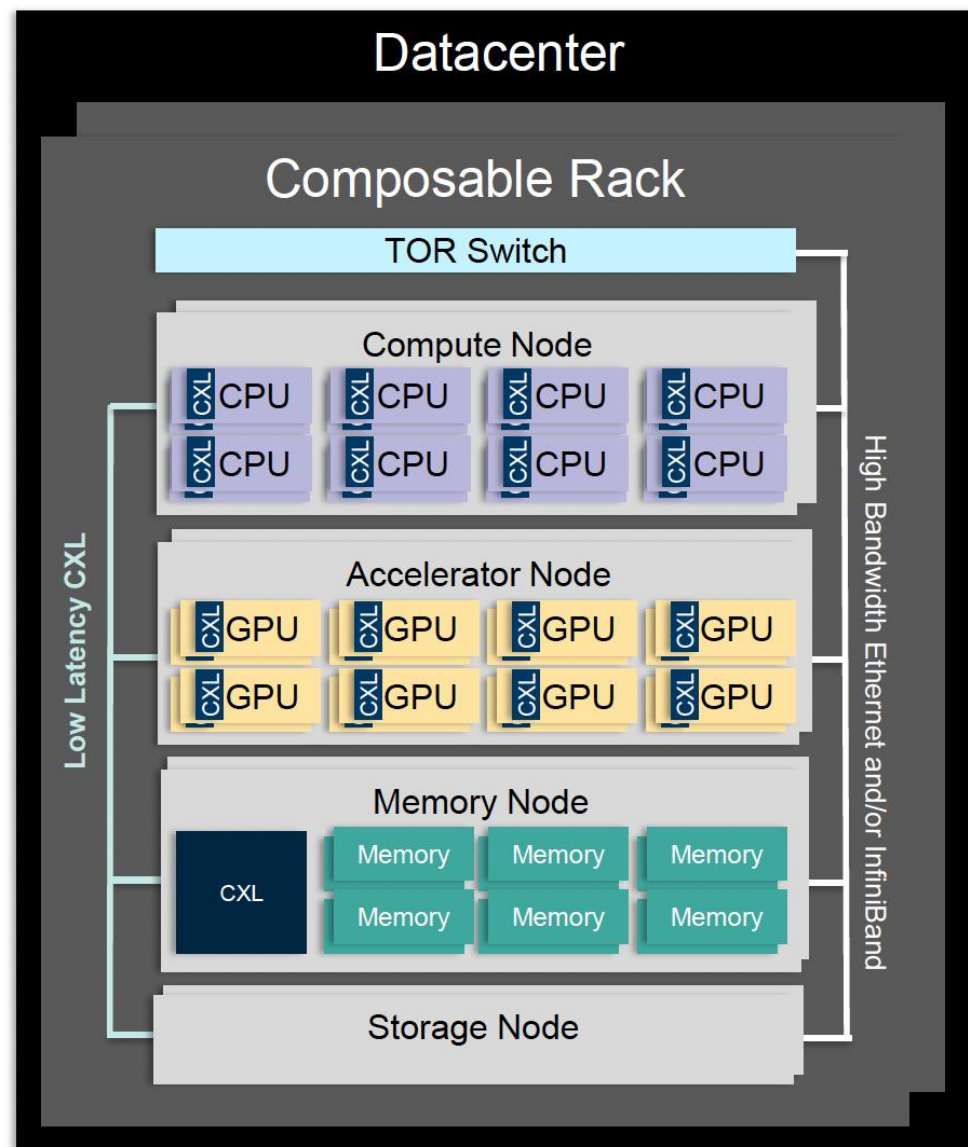
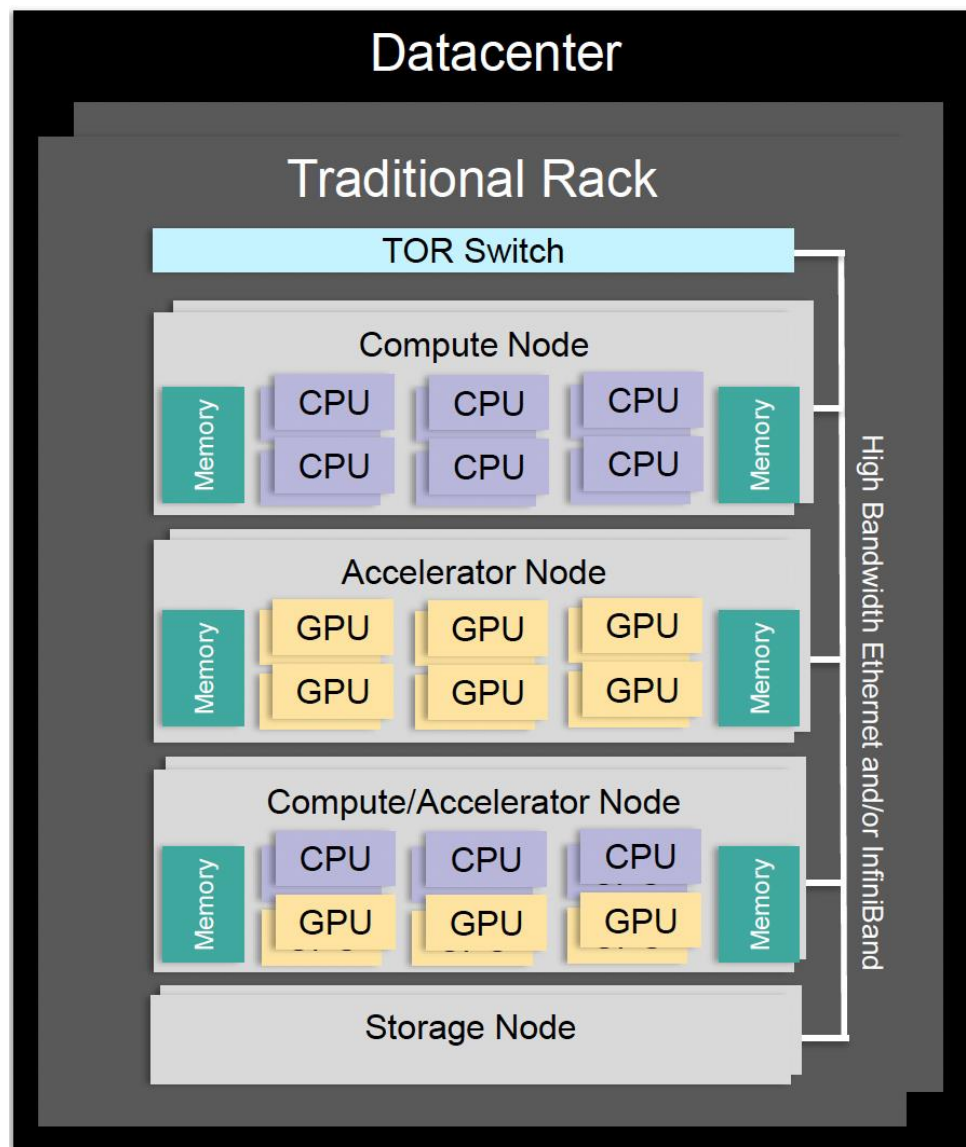
KNL, AMD, Cavium/Marvell, GPU

Many Different Specialized
(Post-Moore Scarcity)

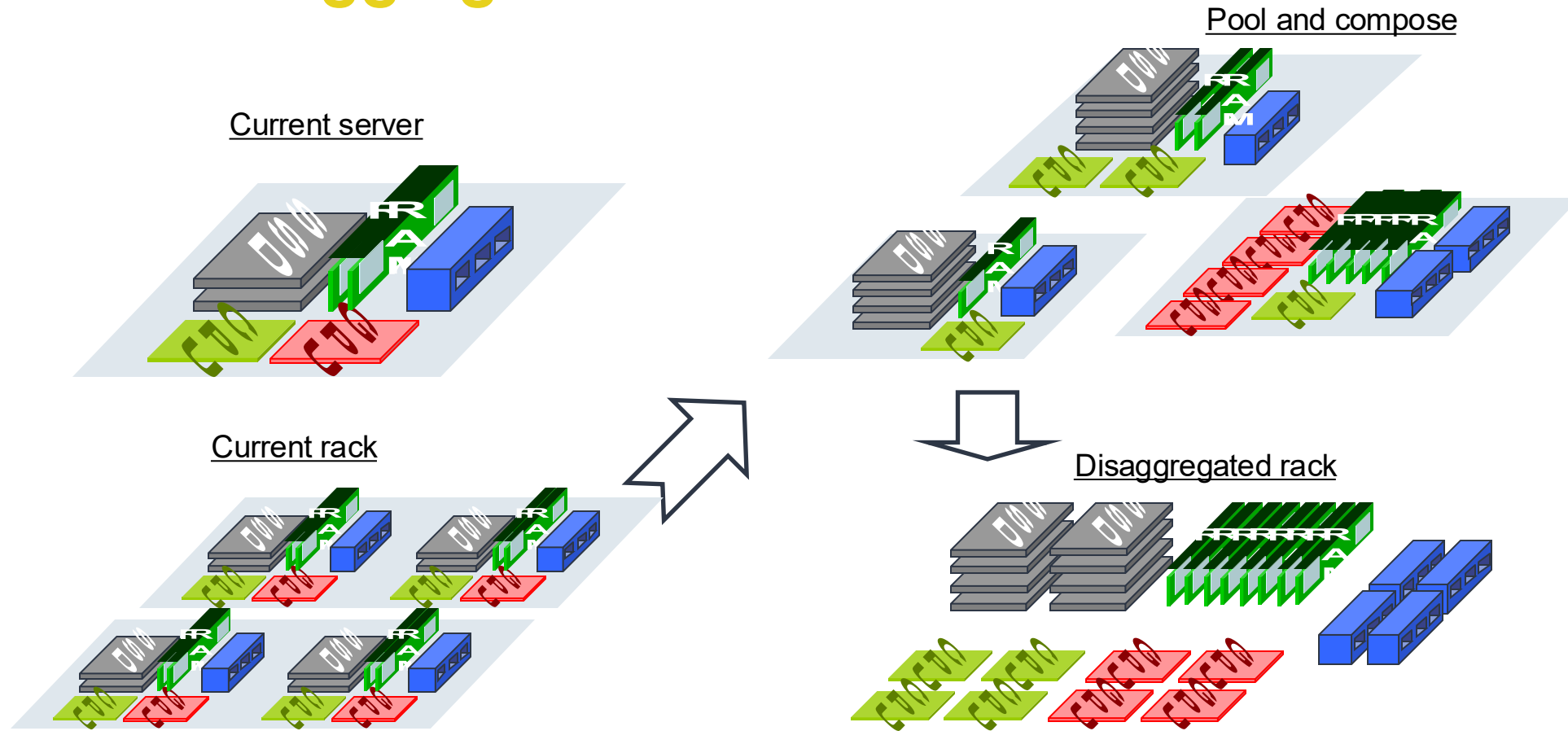


Apple, Google, Amazon
Microsoft

You may then want to compose architectures!

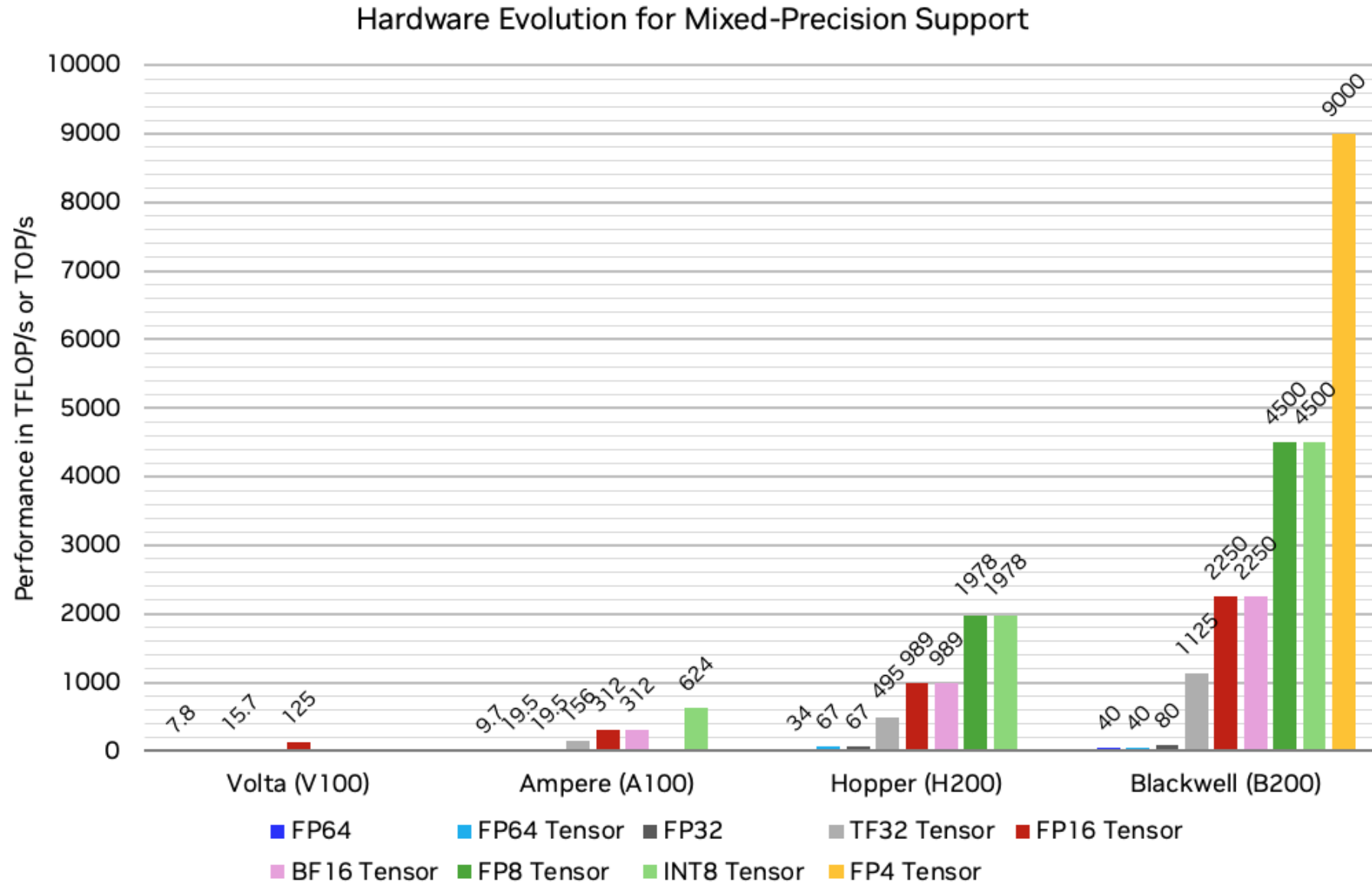


Or even disaggregate hardware resources!



Current disaggregation solutions use Interconnect bandwidth (1 – 10 GB/s)
But this is significantly inferior to RAM bandwidth (100 GB/s – 1 TB/s)

Peak performance of four generations of NVIDIA GPUs

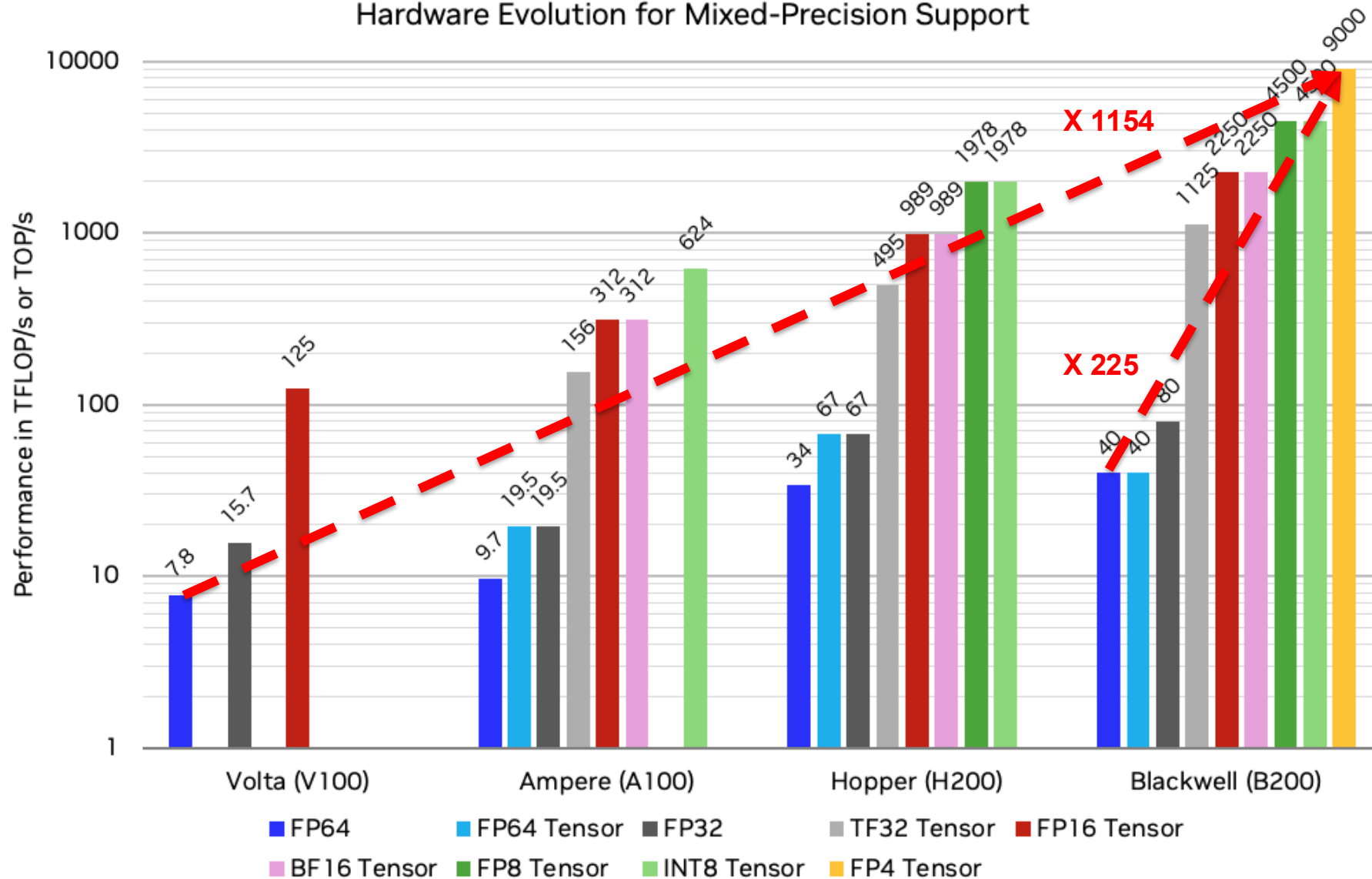


Slide courtesy
H. Bayraktar, NVIDIA

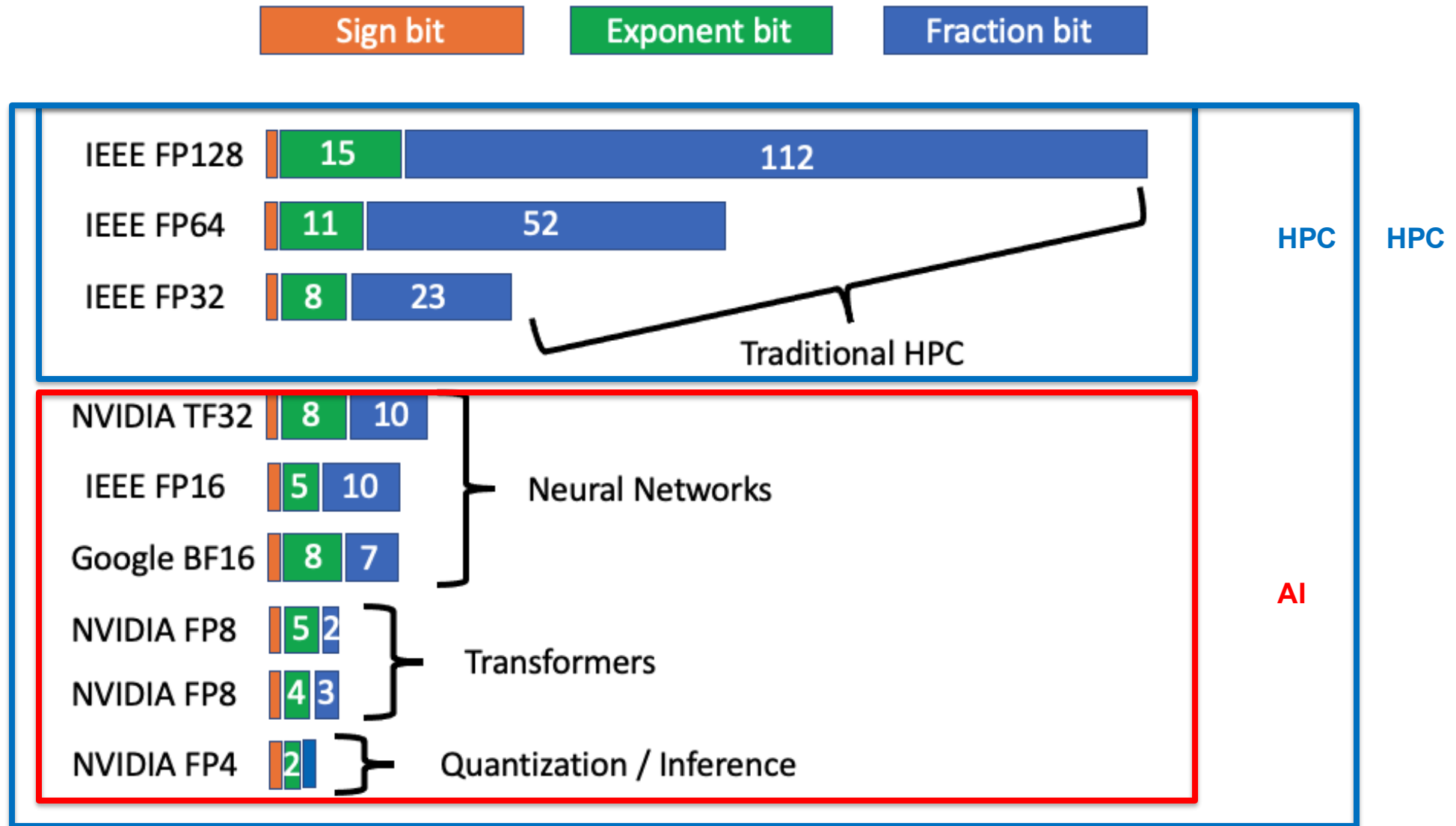
Peak performance of four generations of NVIDIA GPUs

Hardware Evolution for Mixed-Precision Support

Log scale



Feeling like a kid in a candy store



Motivations for Mixed Precisions (2)

| IBM Cell Broadband Engine | Apple ARM Cortex-A9 | NVIDIA Kepler K10, K20, K40, K80 | NVIDIA Volta/Turing | NVIDIA Volta/Turing |
|---------------------------------|---------------------------|--|------------------------|------------------------|
| 14x | 7x | 3x | 2x | 16x |
| 32 bits / 64 bits | 32 bits / 64 bits | 32 bits / 64 bits | 32 bits / 64 bits | 16 bits / 64 bits |

| Peak Performance in TF/s | V100 NVLink | A100 NVLink | H100 SXM | B200 |
|--------------------------|-------------|-------------|----------|------|
| FP64 | 7.5 | 9.7 | 34 | 90 |
| FP32 | | 19.5 | 67 | 180 |
| FP64 Tensor Core | 15 | 19.5 | 67 | 40 |
| FP/TF32 Tensor Core | | 156 | 495 | 1125 |
| FP16 Tensor Core | 120 | 312 | 990 | 2250 |
| FP8/INT8 Tensor Core | - | 624 | 1980 | 4500 |
| FP4 Tensor Core | - | - | - | 9000 |

The Landscape of Mixed Precision Hardware

.Mixed-Precision Startups and their hardware

- GraphCore
 - Colossus
- Habana
 - Labs Gaudi
- Cerebras
 - Wafer Scale Chip
- Blaize
 - Graph Streaming Processor
- Groq
 - Tensor Streaming Processor
- SambaNova
 - Cardinal
- Tenstorrent
 - Grayskull

.NVIDIA mixed-precision hardware

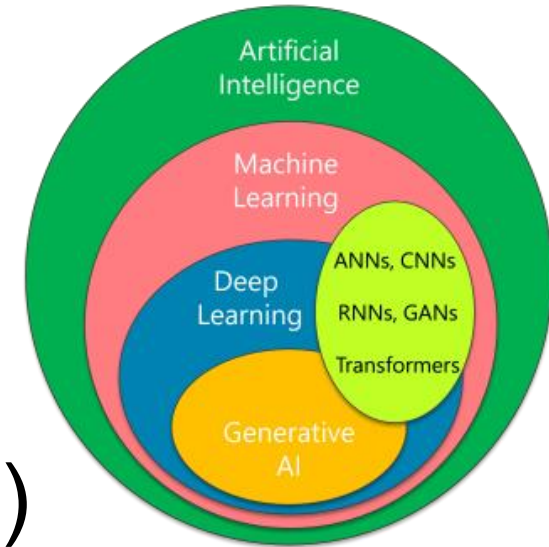
- Pascal
 - FP16 units only
- Volta
 - Tensor Cores and FP16
- Turing
 - Tensor Cores and FP16
- Ampere
 - Tensor Cores for FP16 and FP64
- Hopper
 - Double-, single-, half-, and quarter-precision formats in scalar and tensor units

HPC is not the driver for hardware technology, AI/ML are!

❖ AI and ML have been around for a long time as research efforts

❖ Why now?

- Deluge of data (IoT, Internet)
- Flops are free (hardware technology scaling)
- Development of new AI/ML algorithms (asynchronous, 2nd order methods, Federated Learning)
- *Market! Market! Market! Oh, did I say Market?!*



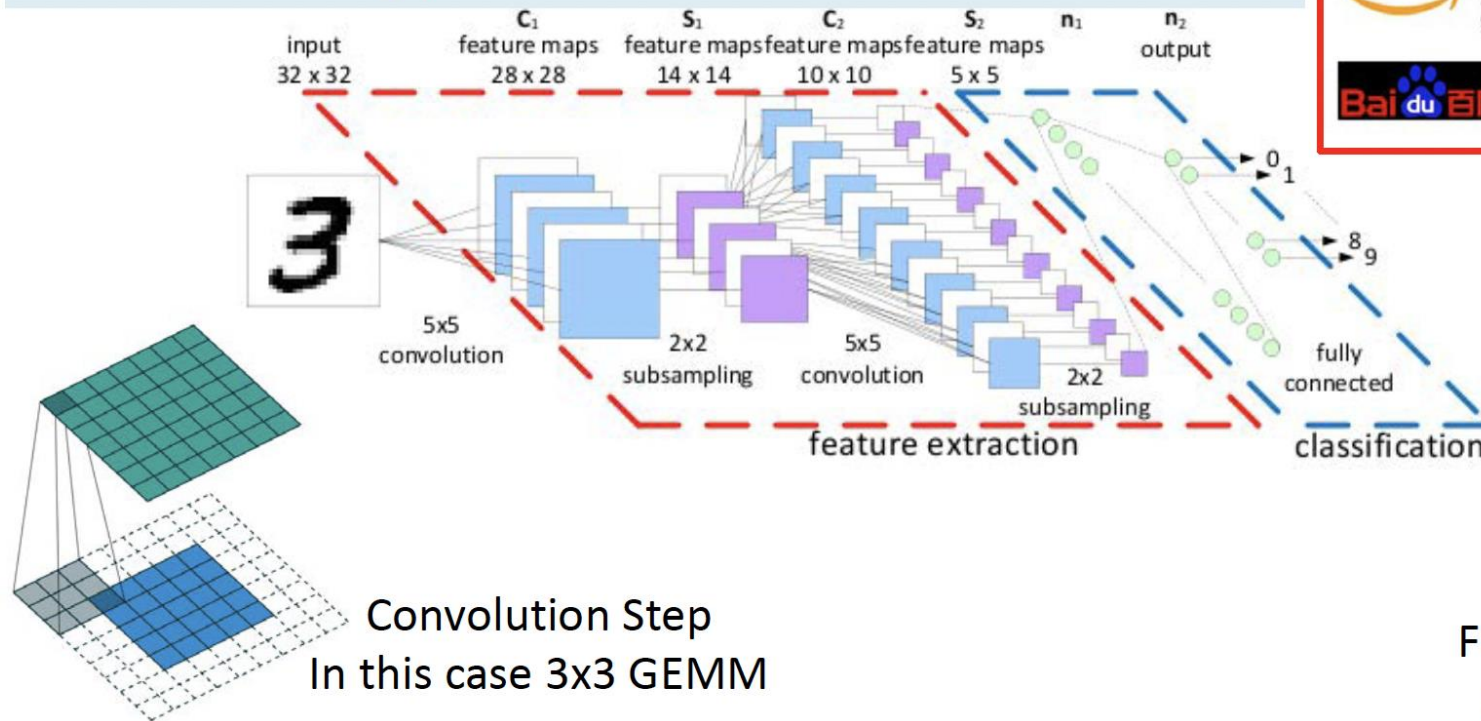
Batching (not bashing!) AI workloads

Deep Learning Needs Small Matrix Operations

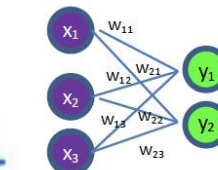
Matrix Multiply is the time-consuming part.

Convolution Layers and Fully Connected Layers require matrix multiply

There are many GEMM's of small matrices, perfectly parallel, can get by with 16-bit floating point



Emergence of AI-Specific Hardware Ecosystem



$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Fully Connected
Classification

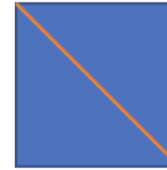
Motivations for Mixed Precisions (1)

- ❖ Less communication: reduce memory and network traffic
- ❖ Reduce memory footprint
- ❖ Increase throughput: more flops per seconds
- ❖ Reduce time-to-solution
- ❖ Reduce energy consumption
- ❖ More science per joule!

What if HPC algorithms could exploit these AI hw features?

- ❖ Use a mathematical technique
 - Get an approximation in lower precision then use something like Newton's method to enhance accuracy
- ❖ Transfer less bytes, data transfer is expensive
 - Store data in primary storage in full precision
 - Transfer the data in short precision
 - Once in registers, compute in full precision
- ❖ Apply algebraic compression
- ❖ Combine all above

The HPL-MxP Benchmark



1. Generate random linear system $Ax=b$
2. Represent the matrix A in low precision (16-bit floating point)
3. Factor A in lower precision into LU by Gaussian elimination
4. Compute approximate solution with LU factors in low precision
5. Perform up to 50 iterations of refinement, e.g., GMRES to get accuracy up to 64-bit floating point
 - a. Use LU factors for preconditioning



Iterative refinement for dense systems, $Ax = b$, can work this way.

$L U = lu(A)$

$x = U \setminus (L \setminus b)$

GMRes preconditioned by the LU to solve $Ax=b$

Lower precision

Lower precision

FP64 precision

$O(n^3)$

$O(n^2)$

$O(n^2)$


6. Validate the answer is correct: scaled residual small $\frac{\|Ax - b\|}{\|A\| \|x\| + \|b\|} \times \frac{1}{n\epsilon} \leq O(10)$
7. Compute performance rate as $\frac{2}{3} \times \frac{n^3}{\text{time}}$

Algorithms Matter, Perhaps More Than Hardware! (1)

HPC algorithmic efficiency tracked by Poisson solvers

Consider a Poisson solve in a 3D $n \times n \times n$ box; natural ordering gives bandwidth of n^2

| <i>Year</i> | <i>Method</i> | <i>Reference</i> | <i>Storage</i> | <i>Flops</i> |
|-------------|---------------|-------------------------|----------------|------------------|
| 1947 | GE (banded) | Von Neumann & Goldstine | n^5 | n^7 |
| 1950 | Optimal SOR | Young | n^3 | $n^4 \log n$ |
| 1971/77 | MILU-CG | Reid/Van der Vorst | n^3 | $n^{3.5} \log n$ |
| 1984 | Full MG | Brandt | n^3 | n^3 |



If $n = 64$, this implies an overall reduction in flops of ~16 million *

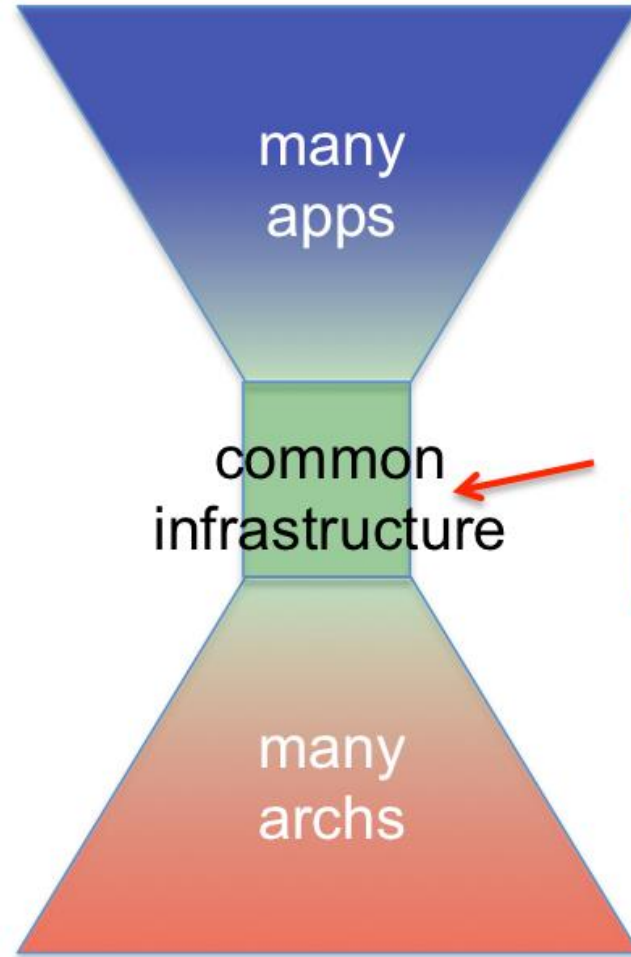
*Six months is reduced to 1 second (recall: 3.154×10^7 seconds per year)

Algorithms Matter, Perhaps More Than Hardware! (2)

Algorithms improve exponents; Moore only adjusts the base

- To scale to extremes, one must start with algorithms with optimal asymptotic complexity, $O(N \log^p N)$, $p = 0, 1, 2$
- These are typically (not exclusively) recursively hierarchical
- Some such algorithms through the decades:
 - Fast Fourier Transform (1960's): $N^2 \rightarrow N \log N$
 - Multigrid (1970's): $N^{4/3} \log N \rightarrow N$
 - Fast Multipole (1980's): $N^2 \rightarrow N$
 - Sparse Grids (1990's): $N^d \rightarrow N (\log N)^{d-1}$
 - \mathcal{H} matrices (2000's): $N^3 \rightarrow k^2 N (\log N)^2$
 - MLMC (2000's): $N^{3/2} \rightarrow N (\log N)^2$
 - Randomized matrix algorithms (2010's): $N^3 \rightarrow N^2 \log k$
 - ??? (2020's): $??? \rightarrow ???$

Revisiting the Hourglass



Shifting
your **I/O-bound**
applications
to **compute-bound**

ECRC is
right

here



@KAUST_ECRC



<https://www.facebook.com/ecrckaust>

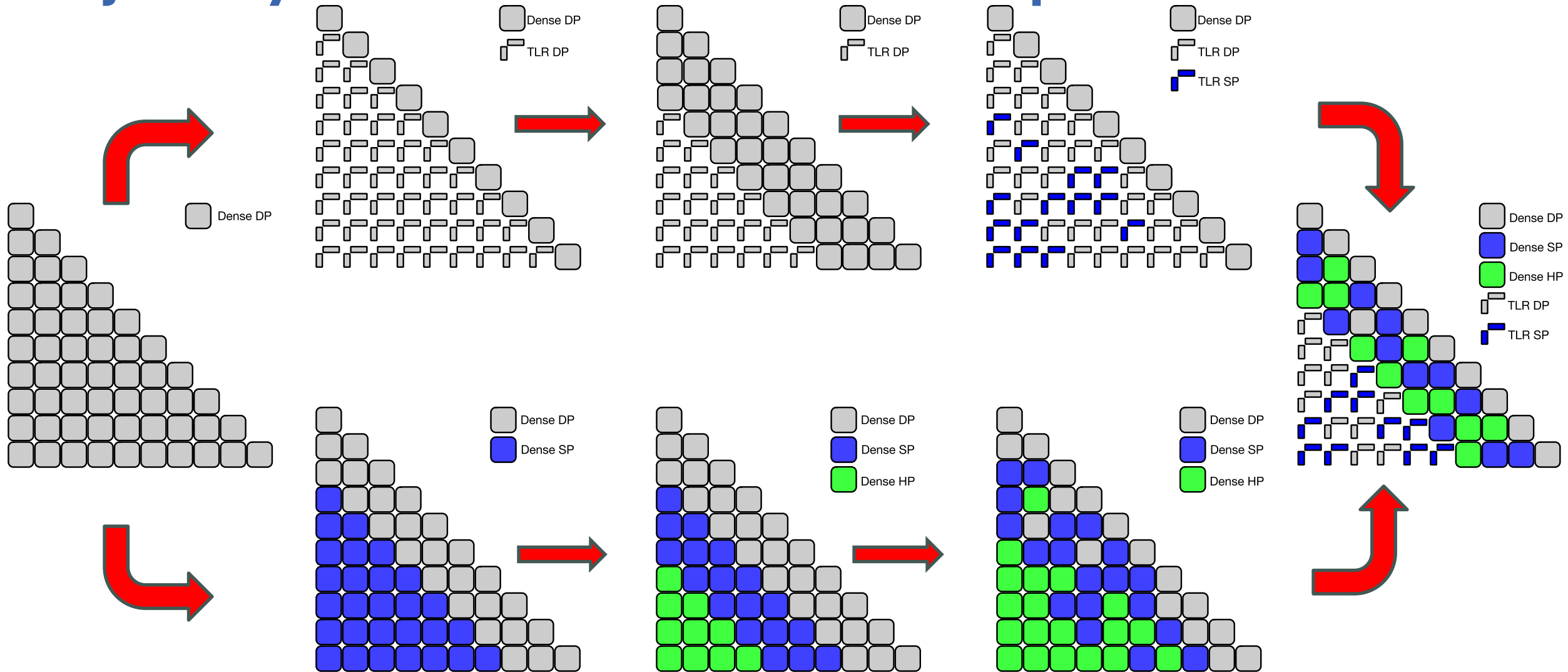
Reshaping Linear Algebra for Massively Parallel Architectures

- Enhance user-productivity using layers of abstraction
- Expose parallelism using fine-grained computations
- Achieve scalability using asynchronous executions
- Exploit data sparsity using low-rank approximations
- Maintain code portability using standard basic blocks

Are you willing to redesign your algorithm?

One possible productive solution: **Matricization**

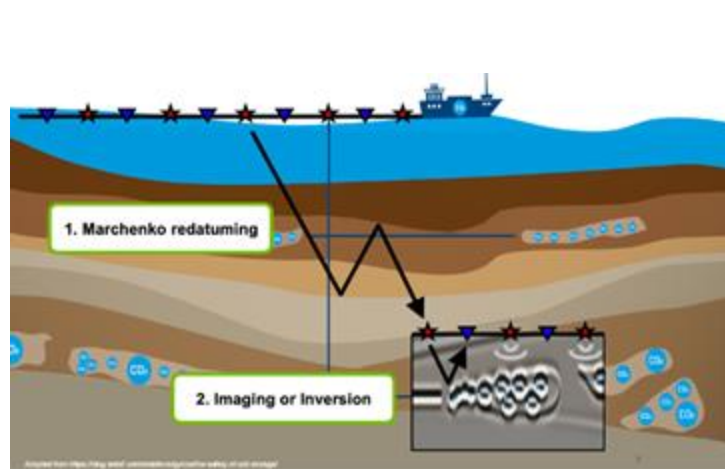
Computational efficiency through tuned approximation: A journey with tile low-rank and mixed precisions



1. Don't oversolve: maintain just enough accuracy for the application purpose
2. Economize on storage: no extra copies of the original matrix

Example of Accelerated Applications

Seismic Imaging



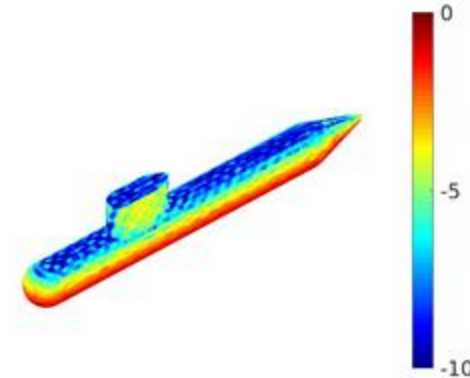
Memory: up to 63X smaller
Analytics: up to 150X faster

Computational Astronomy



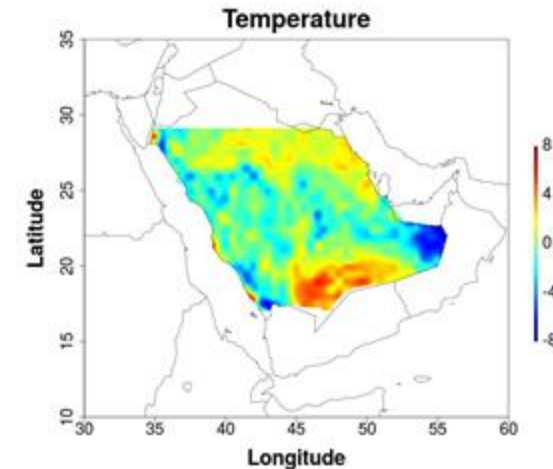
Memory: up to 4X smaller
Analytics: up to 100X faster

Radar Applications



Memory: up to 4X smaller
Analytics: up to 30X faster

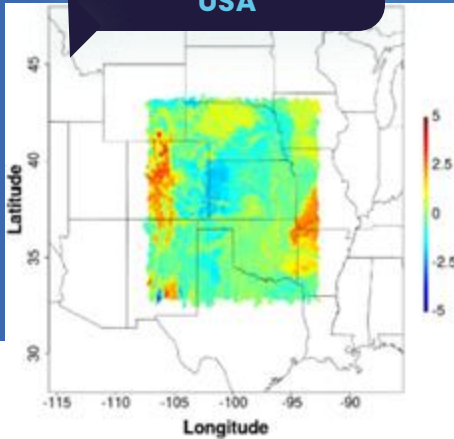
Climate/Weather Prediction



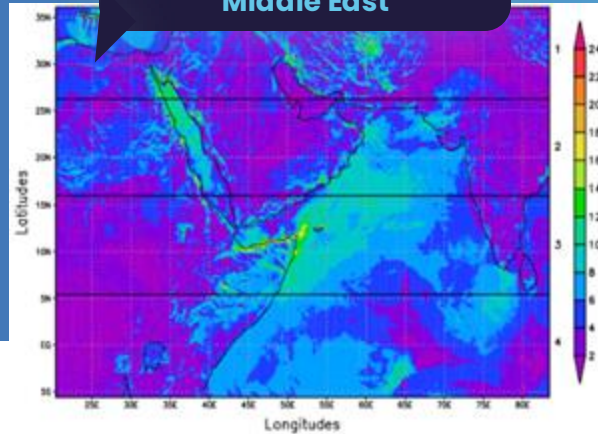
Memory: up to 50X smaller
Analytics: up to 12X faster

Compress to Impress

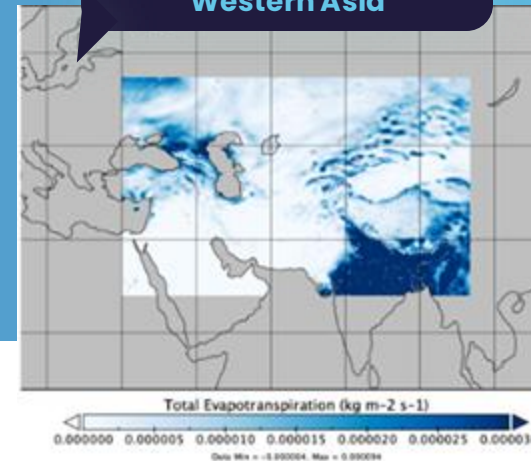
Soil Moisture
USA



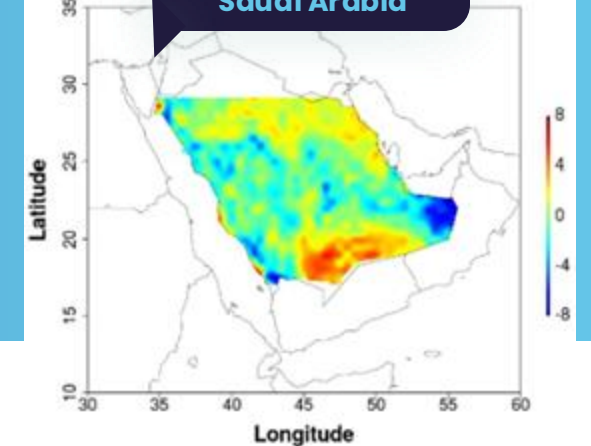
Wind Speed
Middle East



Evapotranspiration
Western Asia



Temperature
Saudi Arabia



Memory

up to **50X** smaller

Analytics

up to **10X** faster

Storage

From **\$750M** to **\$15M**

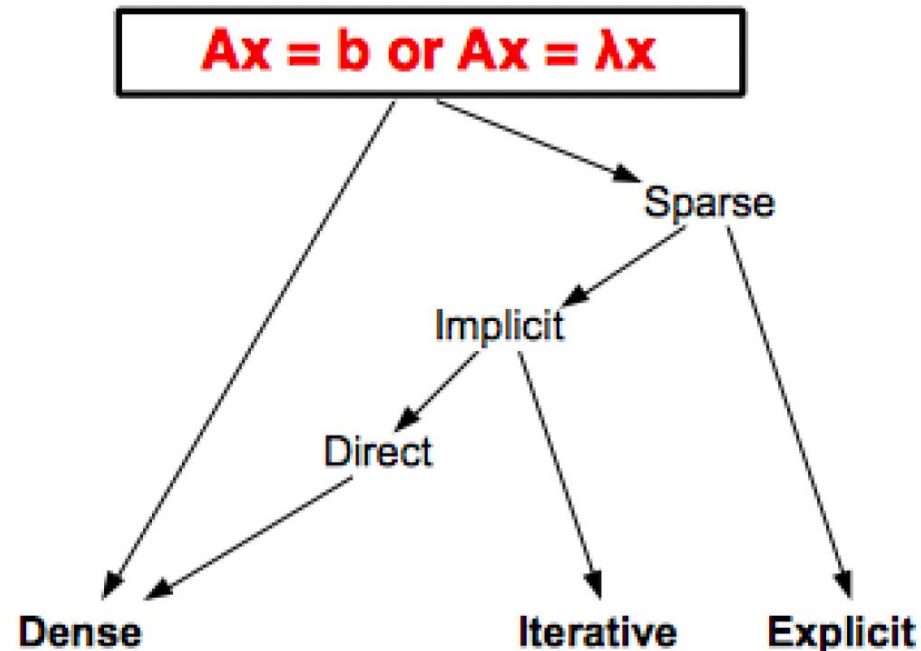
Compute

From **\$400M** to **\$40M**

Linear Algebra 101

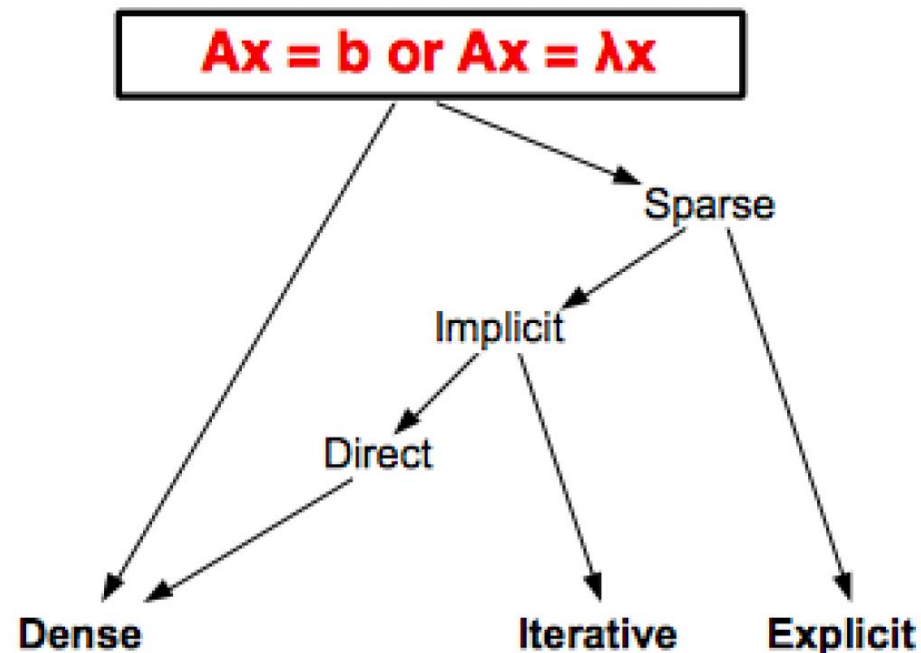
Critical component for many scientific applications:

- cardio-magnetism, wave guide propagation, image processing, quantum chemistry/physics, atomic structure calculations, electromechanics, geophysics/seismology, nonlinear mechanics, computational astronomy, computational fluid dynamics, geospatial statistics, climate/weather prediction, smart health, smart agriculture, smart satellite, etc...



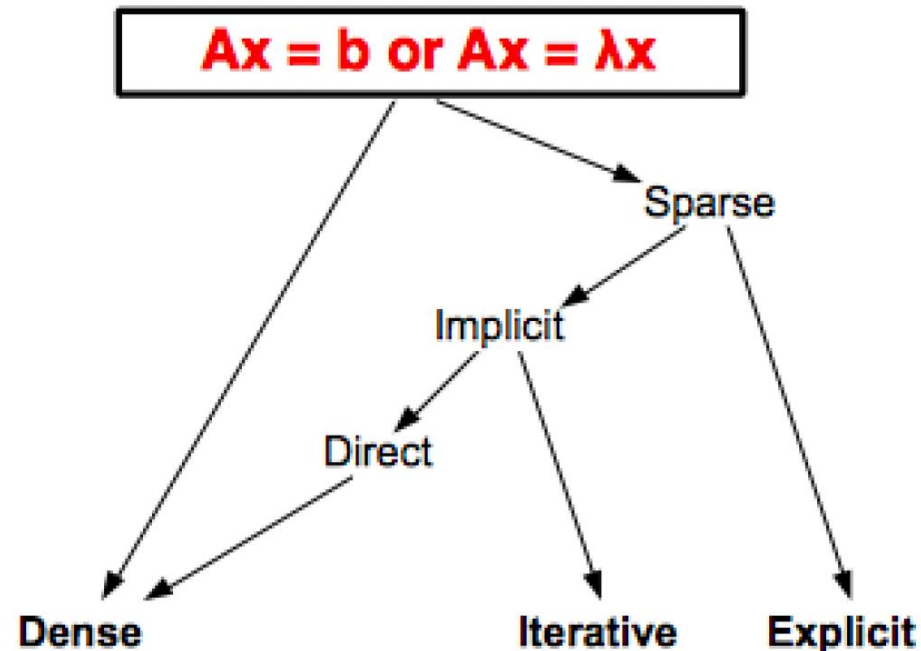
Linear Algebra 101

- Dense solvers: systems of linear equations, eigenvalue / singular value problems
- Sparse iterative solvers: sparse matrix-vector multiplication (SpMV)
- Sparse explicit solvers: stencil computations



Linear Algebra 101

- **Dense solvers: systems of linear equations, eigenvalue / singular value problems**
- Sparse iterative solvers: sparse matrix-vector multiplication (SpMV)
- Sparse explicit solvers: stencil computations



The Two Distinct Optimization Paths

- By using the existing hardware features (sometimes comes for free)
- By redesigning the numerical methods and optimizing the actual implementation (requires effort and time)

Separation of concerns:

- Abstracting the hardware complexity: dynamic runtime systems
- Novel Algorithmic challenges: numerical accuracy/stability

Why software libraries are so important?

- Basic blocks for large applications
- Highly-tuned by vendors if the software becomes mainstream
- Provide abstraction
- Impacting the scientific community

Survey on LA software

Freely Available Software for Linear Algebra (August 2021)

Here is a list of freely available software for the solution of linear algebra problems. The interest is in software for high-performance computers that's available in "open source" form on the web for solving problems in numerical linear algebra, specifically dense, sparse direct and iterative systems, and sparse iterative eigenvalue problems. Please let us know about updates and corrections.

[Send corrections and updates to Dalal.](#)

An old survey of Iterative Linear System Solver Packages can be found at:

<http://www.netlib.org/utk/papers/iterative-survey/>

Thanks,

[Jack Dongarra](#) and

[Dalal Suikkari](#)

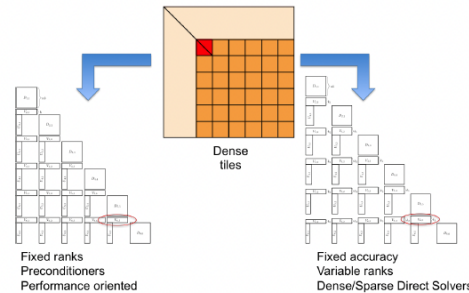
| SUPPORT ROUTINES | License | Support | Type | | Language | Mode | | | Dense | Sparse | Last release date | Updated | New |
|----------------------------------|--------------------------|---------|------|---------|---------------|--------|--------|------|-------|--------|-------------------|---------|-----|
| | | | Real | Complex | | Shared | Accel. | Dist | | | | | |
| Armadillo | Mozilla | yes | X | X | C++ | X | | | X | X | 2018-06-26 | X | |
| Armas | LGPL | yes | X | | C | X | | | X | | 2015-12-22 | | |
| ATLAS | BSD like | yes | X | X | F77/F95/C | X | | | X | | 2018-10-05 | X | |
| BLAS | BSD | yes | X | X | F77/F95/C | X | | | X | | 2017-11-01 | | |
| BLIS | New BSD | yes | X | X | F77/F95/C | X | | | X | | 2021-03-22 | X | |
| Blitz++ | LGPLv3+ | yes | X | X | C++ | X | | | X | | 2019-08-01 | X | |
| BML | BSD | yes | X | X | F77/F95/C | X | X | X | X | X | 2020-09-10 | | X |
| cBLAS | Apache | yes | X | X | C/C++ | X | O | | X | | 2017-01-18 | X | |
| GHOST | BSD | yes | X | X | C/C++ | X | C/X | M | | X | 2020-09-03 | | X |
| GraphBLAS | Apache2 | yes | X | X | C | X | | | X | X | 2021-01-19 | X | |
| KBLAS | BSD | yes | X | X | C/C++ | X | C | | X | | 2017-11-15 | X | |
| KSVD | BSD | yes | X | | C | X | | M | SVD | | 2018-11-08 | | X |
| librsb | LGPLv3 | yes | X | X | F77/F95/C/C++ | X | | | | X | 2017-06-04 | X | |
| LINALG * | ? | ? | | | | | | | | | n/a | | |
| MR3-SMP | New BSD | yes | X | X | F77/F95/C | X | | | X | | 2013-06-24 | | |
| MTL | Boost | yes | X | X | C++ | X | | | X | | 2014-05-22 | | |
| NEWMAT | Own | yes | X | | C++ | X | | | X | | 2008-11-20 | | |
| NIST Sparse BLAS | PD | yes | X | X | C/C++ | X | | | | X | 2009-04-27 | | |
| OpenBLAS | BSD | yes | X | X | F77/F95/C | X | | | X | | 2020-12-12 | X | |
| PMRRR | New BSD | yes | X | X | F77/F95/C | X | | | X | | 2014-02-23 | | |
| pOSKI | BSD | yes | X | X | F77/F95/C/C++ | X | | | | X | 2012-04-27 | | |
| PSBLAS | BSD | yes | X | X | F90 | X | | M | | X | 2020-06-30 | X | |
| QDWH | BSD | yes | X | | C | X | | M | X | | 2017-02-27 | | X |
| Scotch | CeCILL-C | yes | | | F77/F95/C | X | | M | | X | 2020-09-03 | | |
| SparseLib++ | PD | yes | X | X | C/C++ | X | | | | X | 2008-10-30 | | |
| Trilinos/Epetra | BSD | yes | X | | F77/F95/C/C++ | X | | M | X | | 2015-05-07 | | |

<https://docs.google.com/spreadsheets/d/11ESR3uucNvVKEolcalP9gR7ApaOEILwmE5sAS-VRMOM/edit?gid=90156307#gid=90156307>

A Look back to software evolution

Software infrastructure and algorithmic design follow hw evolution in time:

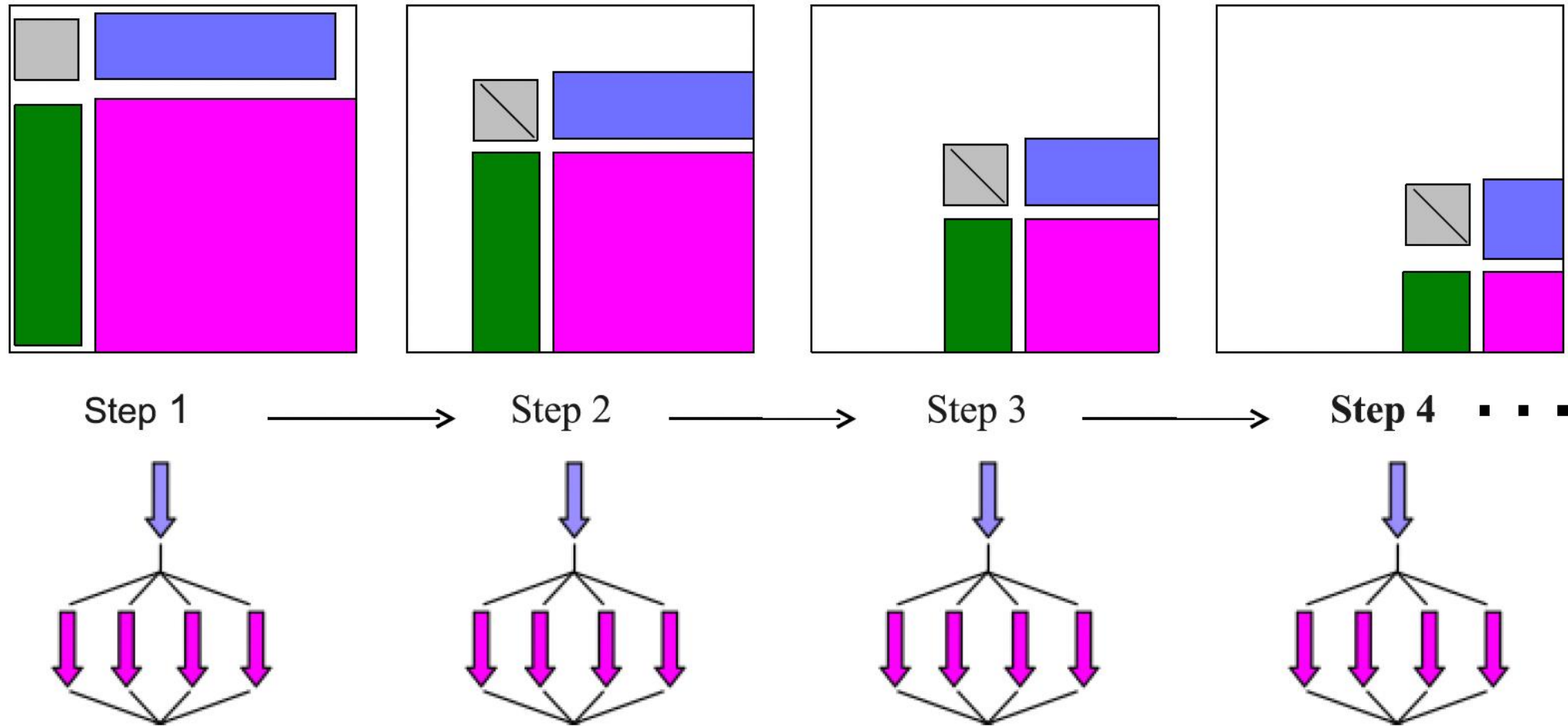
- 70's - LINPACK, vector operations:
Level-1 BLAS operation
- 80's - LAPACK, block, cache-friendly:
Level-3 BLAS operation
- 90's - ScaLAPACK, distributed memory:
PBLAS Message passing
- 00's:
 - PLASMA, MAGMA:
x86 multicore + GPU, DAG scheduler
- 10's:
 - SLATE:
Standard (MPI + OpenMP), Applications
- 20's:
 - HiCMA, numerical approximations:
algebraic compression, mixed precisions



Facts on LAPACK / ScaLAPACK

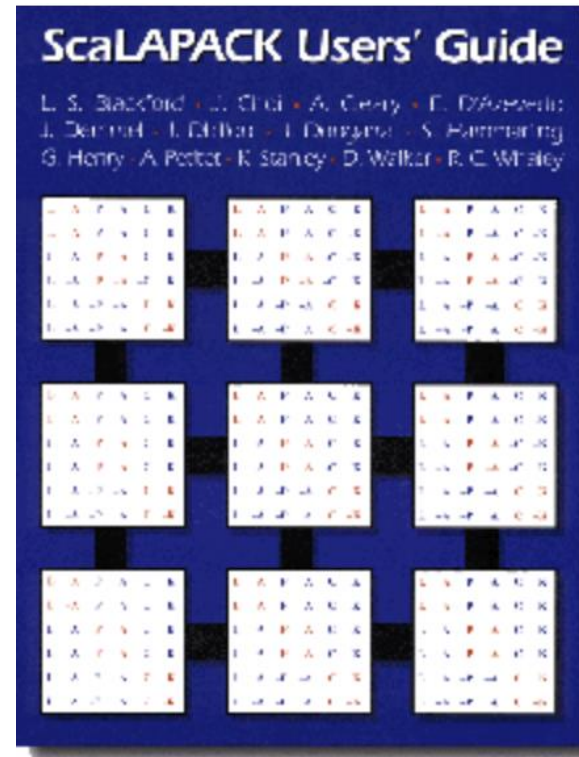
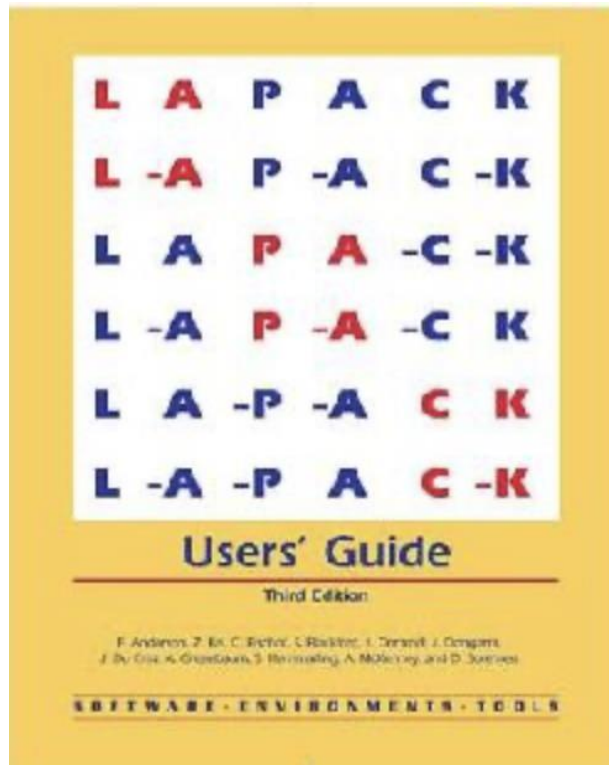
- Open-source packages (downloaded more than 60 million times)
- Large community contributors
- Integrated in open-source libraries (PETSc, SLEPc, MUMPS, CPMD, CP2K ...)
- Integrated in vendor numerical software (Mathworks, Intel, Cray, IBM, HP, Fujitsu ...)
- Critical library for many scientific application

Block Algorithms: Fork-Join Paradigm



The Three Siblings

- BLAS: kernels for dense linear algebra
- LAPACK: sequential dense linear algebra
- ScaLAPACK: parallel distributed dense linear algebra



BLAS: Basic Linear Algebra Subroutines (BLAS)

- Level 1 BLAS

Consider AXPY ($y = \alpha \cdot x + y$): $2n$ flops on $3n$ read/write operations

Computational intensity = $(2n)/(3n) = 2/3$

Too low to run near peak speed (read/write dominates).

- Level 2 BLAS

Standard library of 25 operations (mostly) on matrix/vector pairs

“GEMV”: $y = \alpha \cdot A \cdot x + \beta \cdot y$, “GER”: $A = A + \alpha \cdot x \cdot y^T$

Up to 4 versions of each (S/D/C/Z), 66 routines

Why BLAS 2 ? They do $O(n^2)$ ops on $O(n^2)$ data

So computational intensity still just $(2n^2)/(n^2) = 2$

OK for vector machines, but not for machine with cache memory.

BLAS: Basic Linear Algebra Subroutines (BLAS)

- Level 3 BLAS

Standard library of 9 operations (mostly) on matrix/matrix pairs

“GEMM”: $C = \alpha \cdot A \cdot B + \beta \cdot C$, $C = \alpha \cdot A \cdot A^T + \beta \cdot C$

Up to 4 versions of each (S/D/C/Z), 30 routines

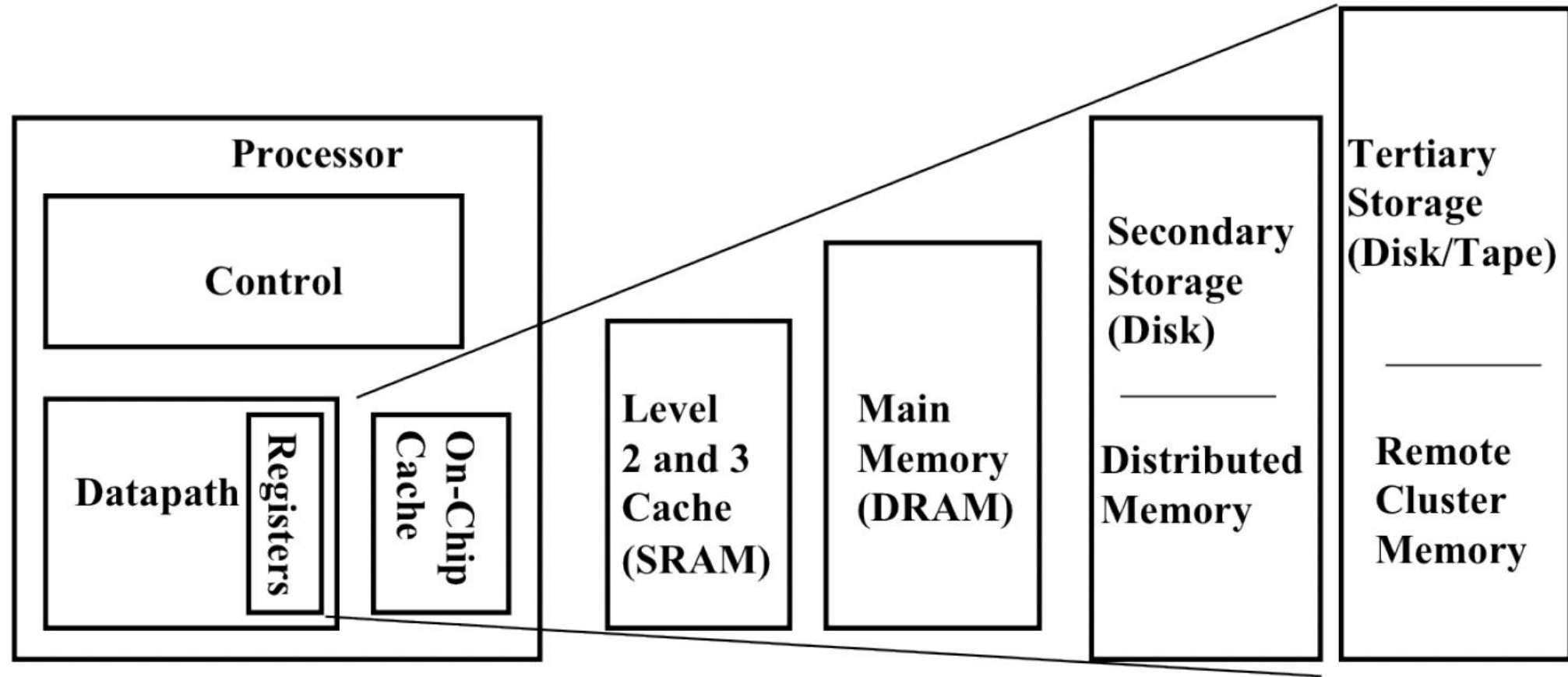
Why BLAS 3 ? They do $O(n^3)$ ops on $O(n^2)$ data

So computational intensity $(2n^3)/(4n^2) = n/2$ – big at last!

Good for machines with caches and many memory hierarchy levels

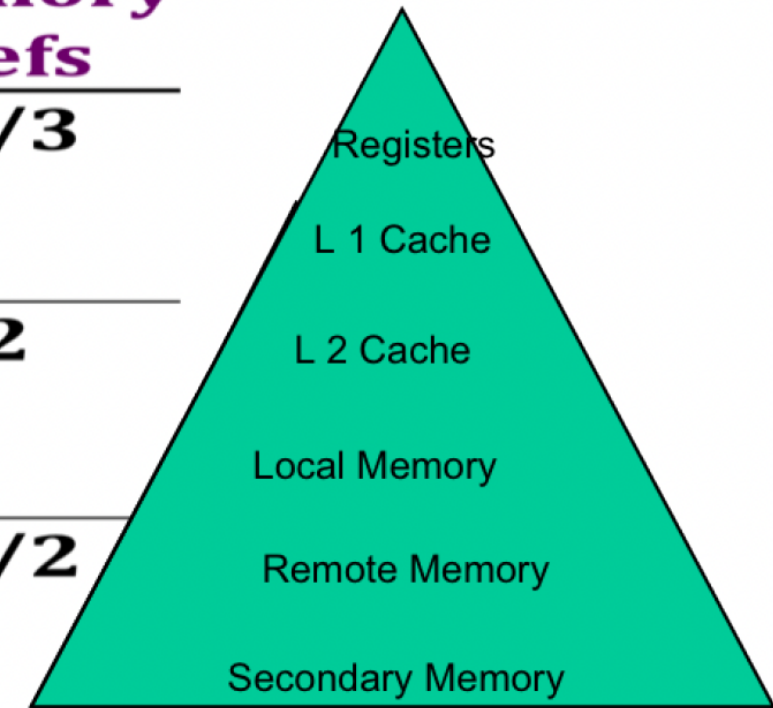
Data Locality

- Can only do arithmetic on data at the top of the hierarchy

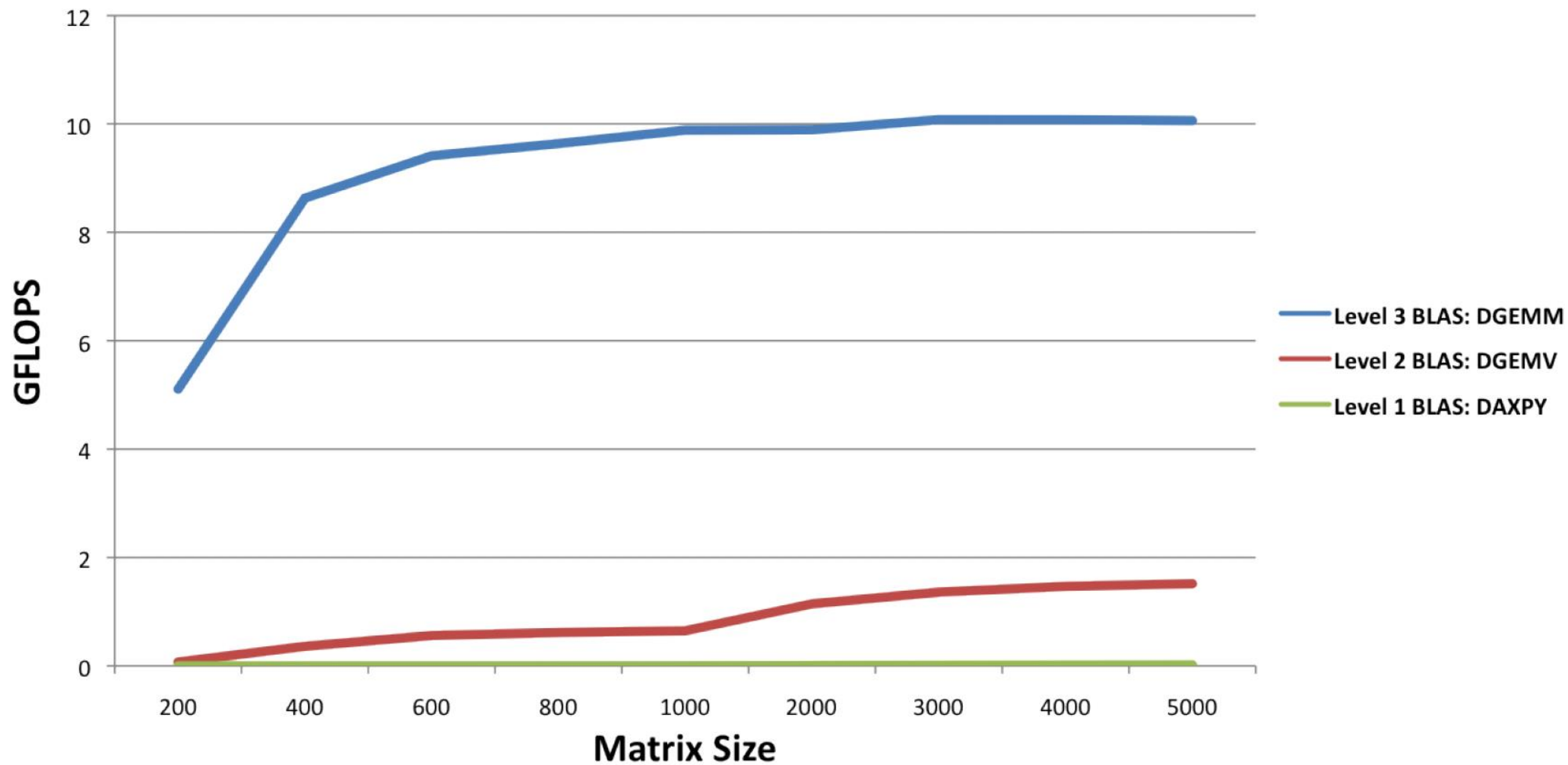


Why Higher Level BLAS?

| 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 $C = C + AB$ | $4n^2$ | $2n^3$ | $n/2$ |



BLAS Performance



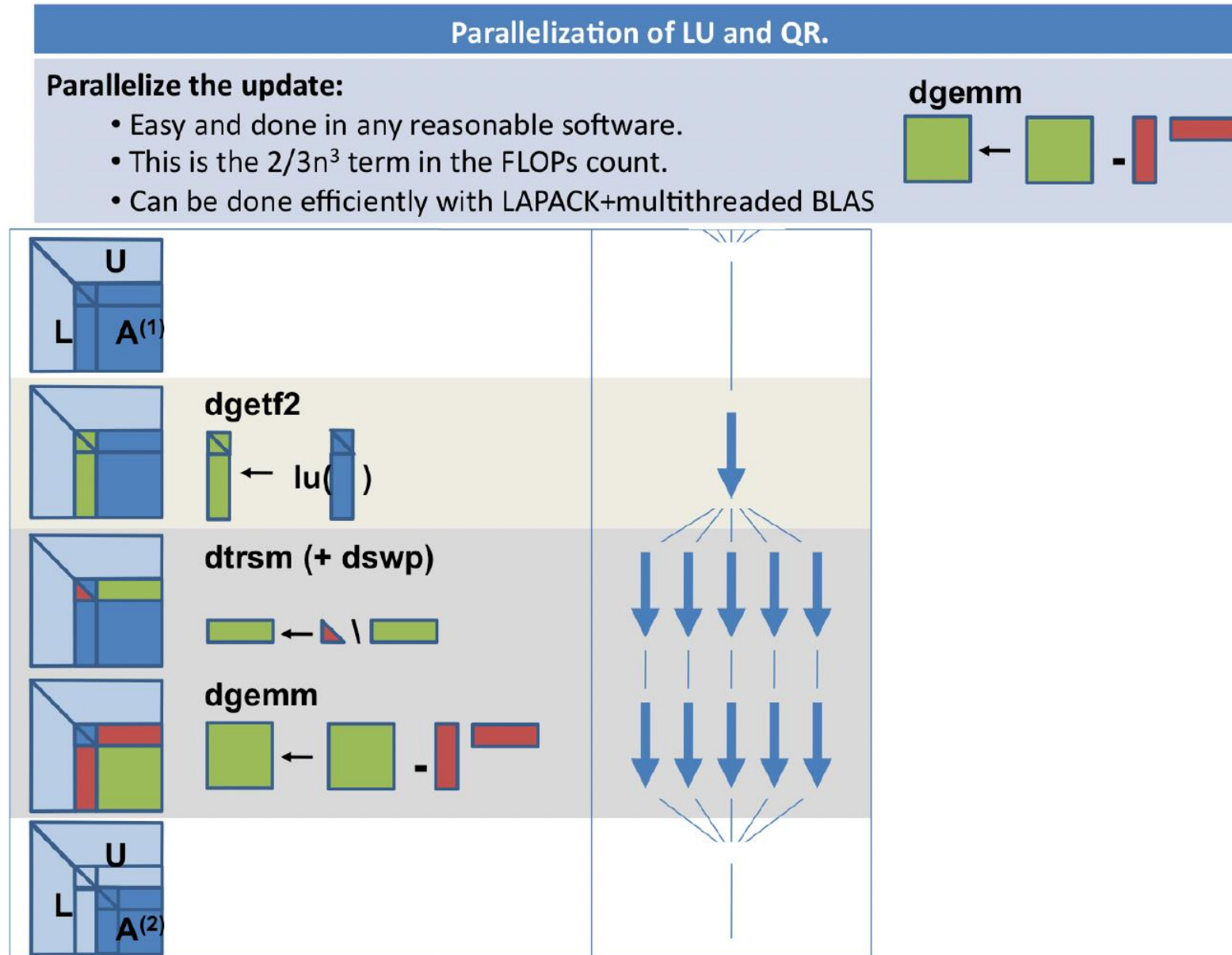
Linear Algebra Package (LAPACK)

- Algorithms we can turn into (nearly) 100% BLAS 3
Linear Systems: solve $A.x = b$ 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): $(A^T.A).x = \sigma^2.x$
- Generalized problems (eg $A.x = \lambda B.x$)
- Error bounds for everything
- Lots of variants depending on A 's structure (banded, $A = A^T$, etc)

Linear Algebra Package (LAPACK)

- LAPACK is in FORTRAN Column Major
- LAPACK is SEQUENTIAL
- LAPACK is a REFERENCE implementation

Fork-Join Paradigm

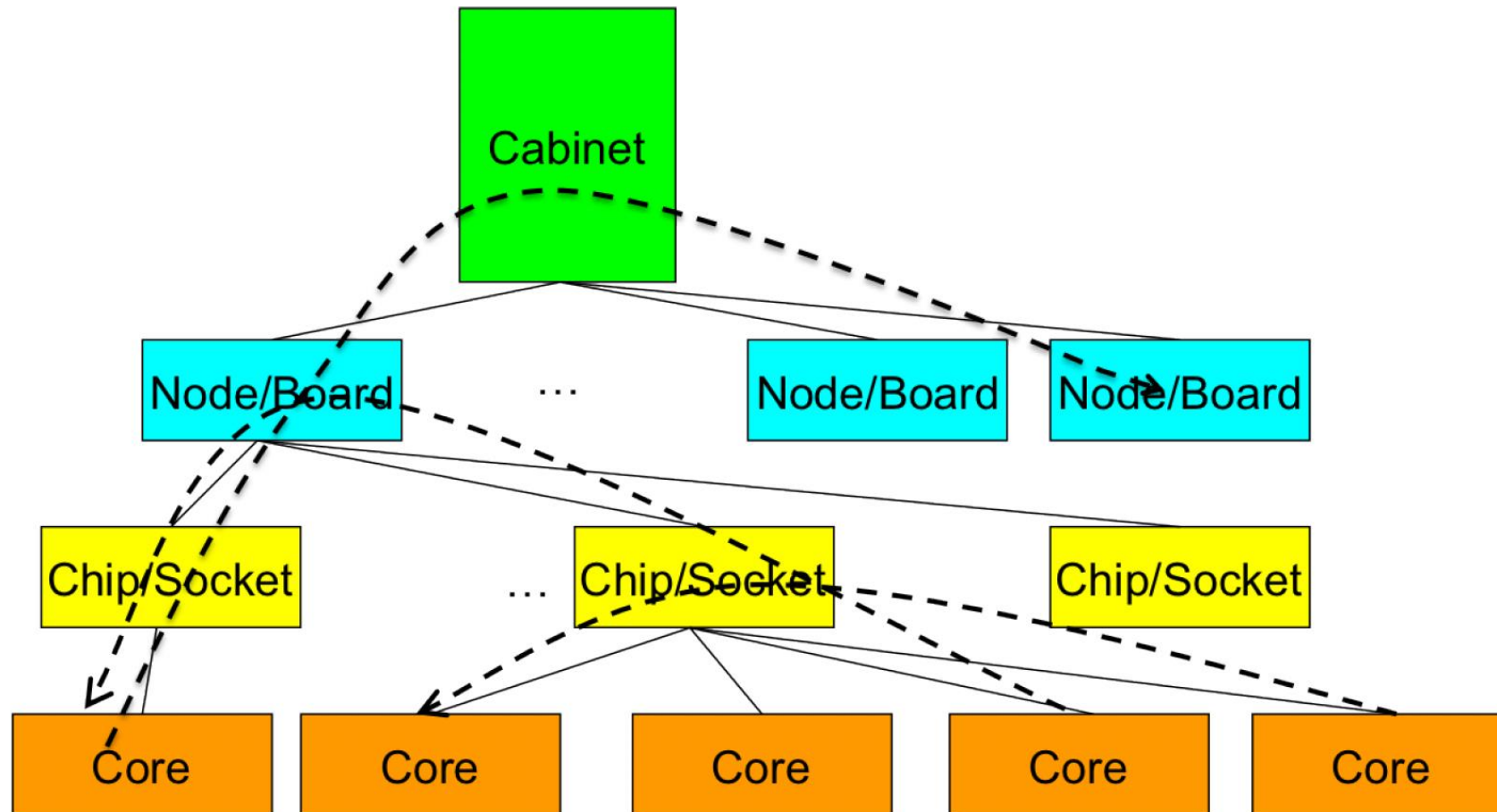


Scalable Linear Algebra Package (ScaLAPACK)

- Library of software dealing with dense & banded routines
- Distributed Memory - MPI
- MIMD Computers and Networks of Workstations
- Clusters of SMPs
- Relies on LAPACK / BLAS and BLACS / MPI
- Includes PBLAS (Parallel BLAS)
- ScaLAPACK is in FORTRAN and C
- ScaLAPACK is for PARALLEL DISTRIBUTED
- ScaLAPACK is a REFERENCE implementation

Example of typical parallel machine

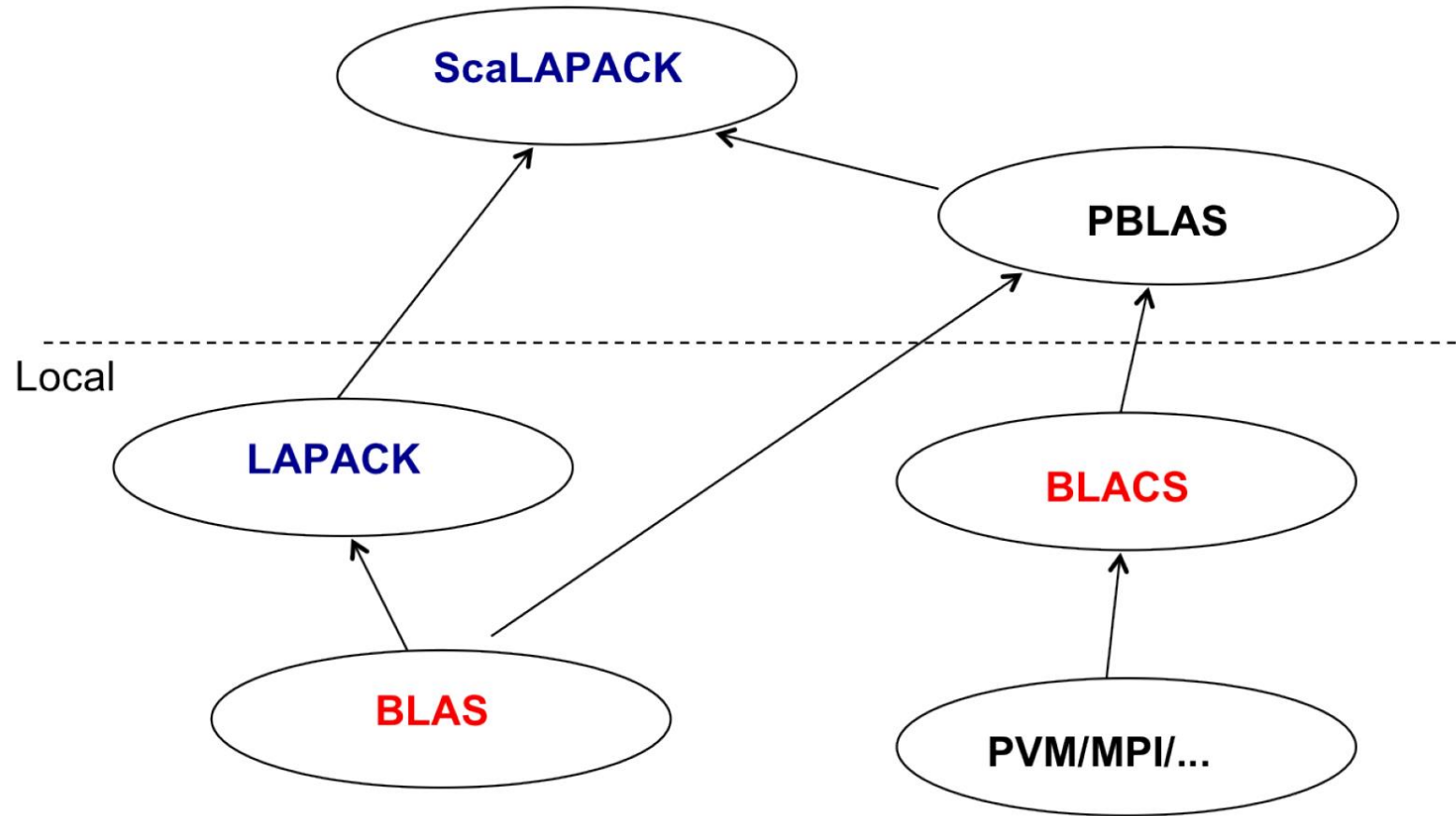
- Shared memory within a node
- Distributed within a single cabinet



ScaLAPACK's Overall Structure

- Object based - Array descriptor
Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
Provides a flexible framework to easily specify additional data distributions or matrix types.
- Currently supports dense, banded matrix structure
- Using the concept of context

Putting them all together...



Distributed Data Layout

2D Block-Cyclic Data Distribution:

| | | | | |
|---|---|---|---|--|
| | | Q | | |
| P | 0 | 1 | 2 | |
| | 3 | 4 | 5 | |
| | 6 | 7 | 8 | |

Processes Grid



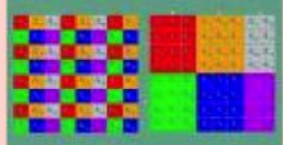

| | | | | |
|----------|----------|----------|----------|----------|
| A_{11} | A_{12} | A_{13} | A_{14} | A_{15} |
| A_{21} | A_{22} | A_{23} | A_{24} | A_{25} |
| A_{31} | A_{32} | A_{33} | A_{34} | A_{35} |
| A_{41} | A_{42} | A_{43} | A_{44} | A_{45} |
| A_{51} | A_{52} | A_{53} | A_{54} | A_{55} |

Logical View (Matrix)

| | | | | |
|----------|----------|----------|----------|----------|
| A_{11} | A_{14} | A_{12} | A_{15} | A_{13} |
| A_{41} | A_{44} | A_{42} | A_{45} | A_{43} |
| A_{21} | A_{24} | A_{22} | A_{25} | A_{23} |
| A_{51} | A_{54} | A_{52} | A_{55} | A_{53} |
| A_{31} | A_{34} | A_{32} | A_{35} | A_{33} |

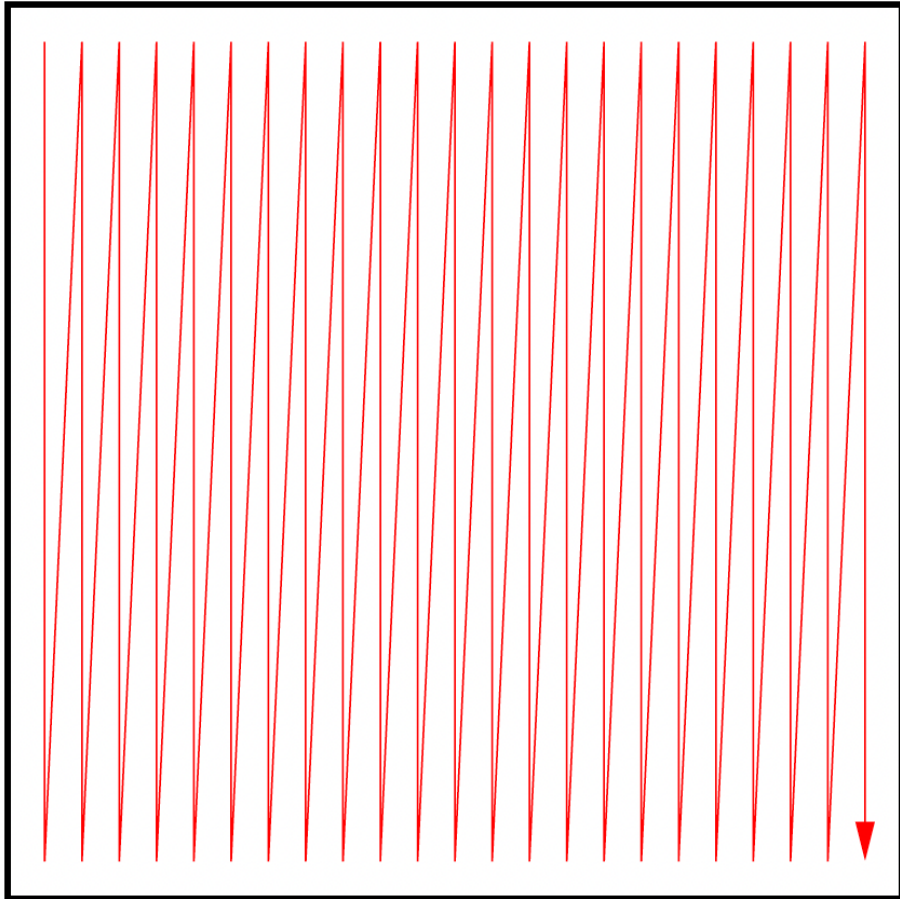
Local View (CPUs)

(R)Evolution

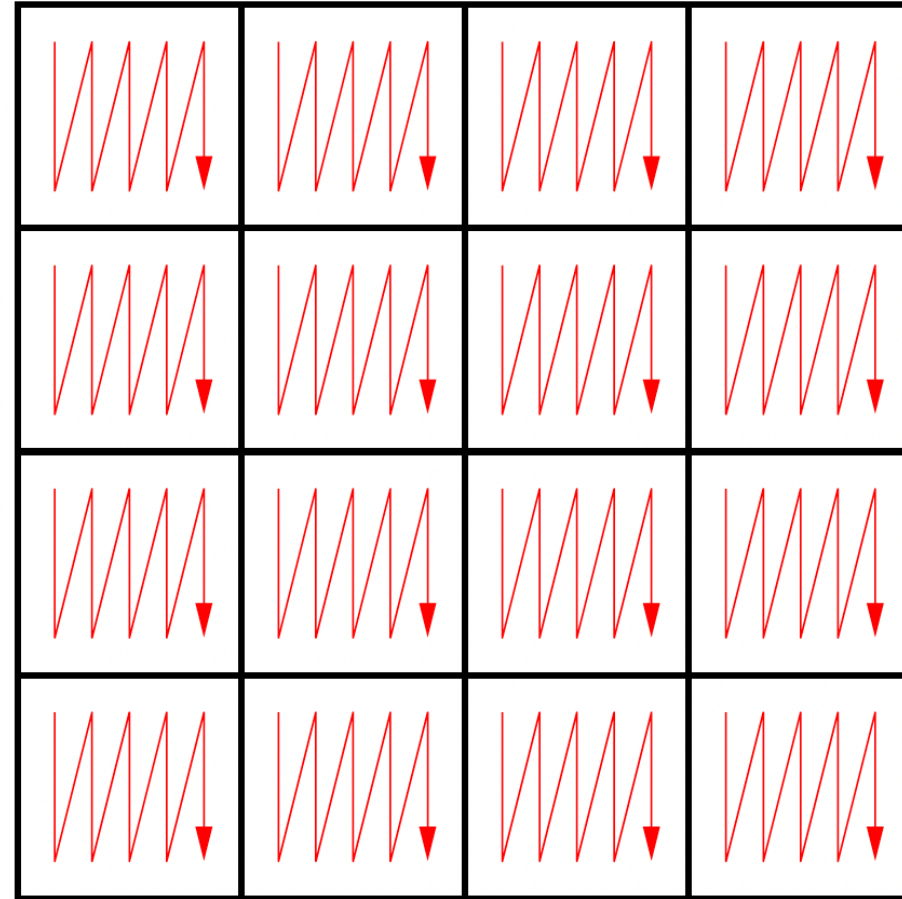
| Software/Algorithms follow hardware evolution in time | | |
|---|--|---|
| LINPACK (70's) (Vector operations) |  | Rely on - Level-1 BLAS operations |
| LAPACK (80's) (Blocking, cache friendly) |  | Rely on - Level-3 BLAS operations |
| ScaLAPACK (90's) (Distributed Memory) |  | Rely on - PBLAS Mess Passing |
| PLASMA (00's) New Algorithms (many-core friendly) |  | Rely on - a DAG/scheduler - block data layout - some extra kernels |

Tile Data Layout Format

LAPACK: column-major format



PLASMA: tile format



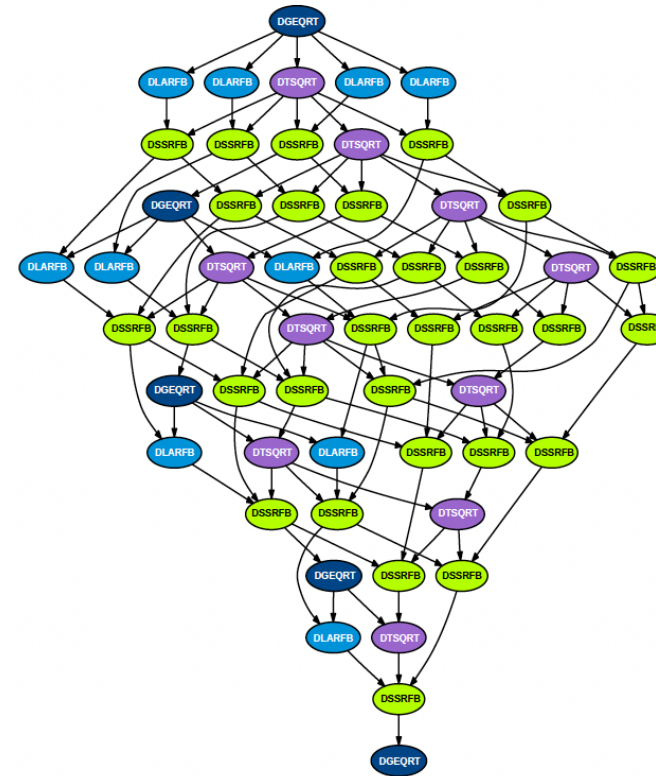
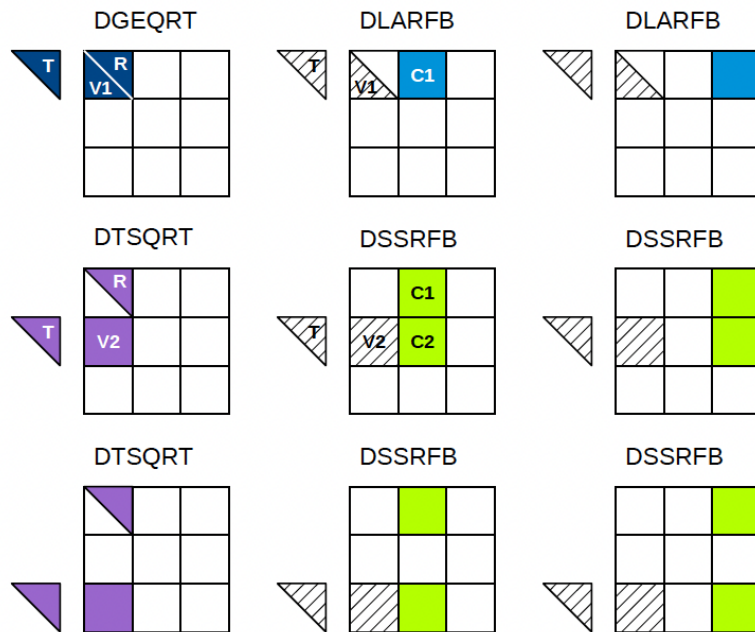
PLASMA: Tile Algorithms

- Parallelism is brought to the fore
- May require the redesign of linear algebra algorithms
- Tile data layout translation
- Remove unnecessary synchronization points between Panel-Update sequences
- DAG execution where nodes represent tasks and edges define dependencies between them
- Dynamic runtime system environment QUARK + **OpenMP**

Example: Least-square solver

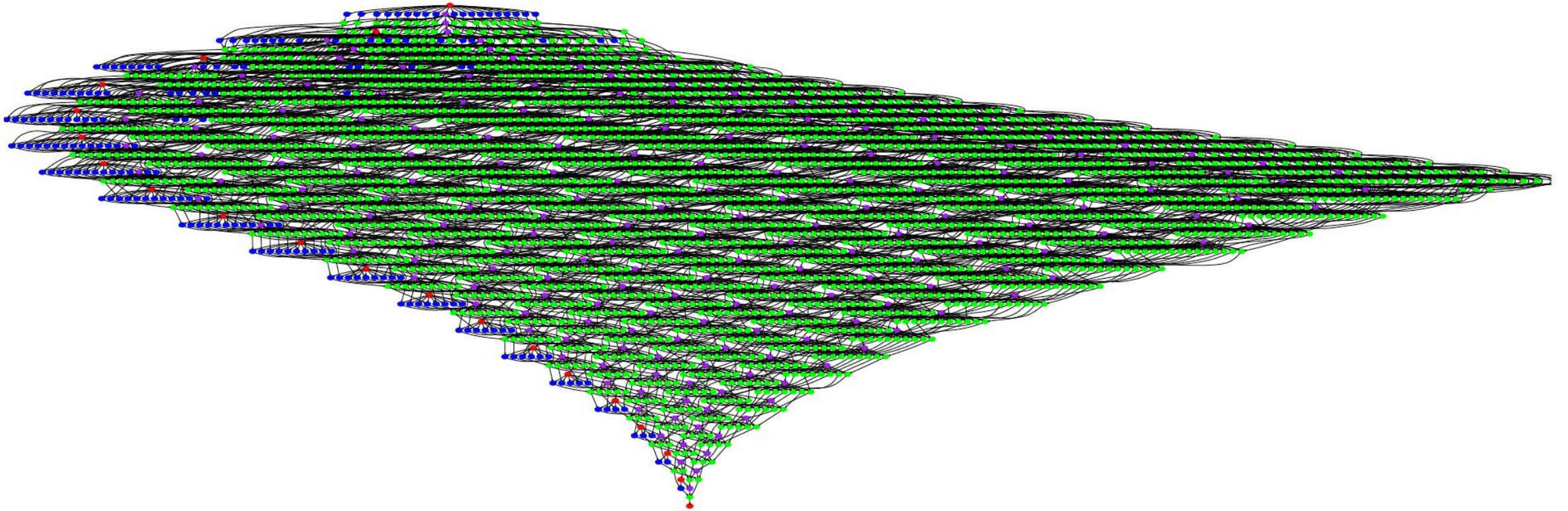
```

FOR k = 0..TILES-1
  A[k][k], T[k][k] ← DGRQRT(A[k][k])
  FOR m = k+1..TILES-1
    A[k][k], A[m][k], T[m][k] ← DTSQRT(A[k][k], A[m][k], T[m][k])
  FOR n = k+1..TILES-1
    A[k][n] ← DLARFB(A[k][k], T[k][k], A[k][n])
    FOR m = k+1..TILES-1
      A[k][n], A[m][n] ← DSSRFB(A[m][k], T[m][k], A[k][n], A[m][n])
  
```



- Fine granularity;
- Tile algorithms;
- Productivity
- **DAG** scheduler framework.

DAG can be Large and Complex



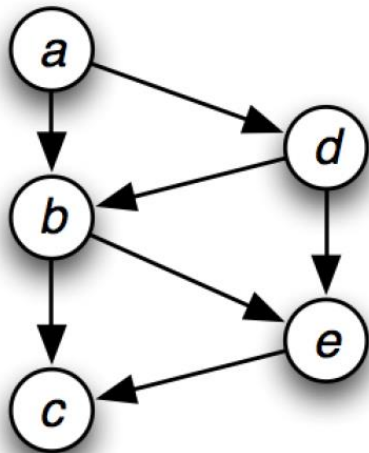
Dynamic Runtime System

- Conceptually similar to out-of-order processor scheduling because it has:

- Dynamic runtime DAG scheduler
- Out-of-order execution flow of fine-grained tasks
- Task scheduling as soon as dependencies are satisfied
- *Producer-Consumer*

DataFlow Programming

- Five decades **OLD** concept
- Programming paradigm that models a program as a directed graph of the data flowing between operations (cf. Wikipedia)
- Think "how things connect" rather than "how things happen"
- *Assembly line*
- Inherently parallel



SLATE-CPU Vs SLATE-GPU

