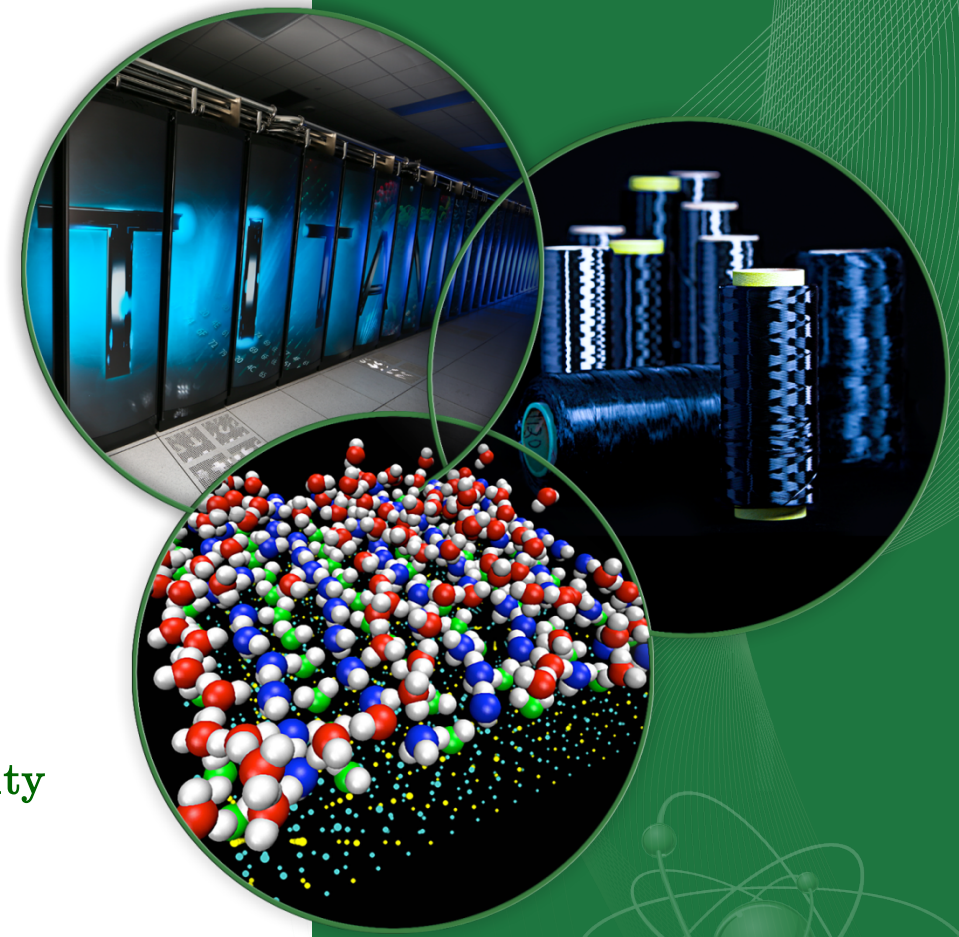
Enabling Portable Processing of Large-Scale Numerical Tensor Algebra Workloads with Applications in Quantum Many-Body Theory on Summit



Dmitry I. Lyakh

liakhdi@ornl.gov

Oak Ridge Leadership Computing Facility

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under contract No. DE-AC05-00OR22725.

DIRAC: Relativistic Electronic Structure

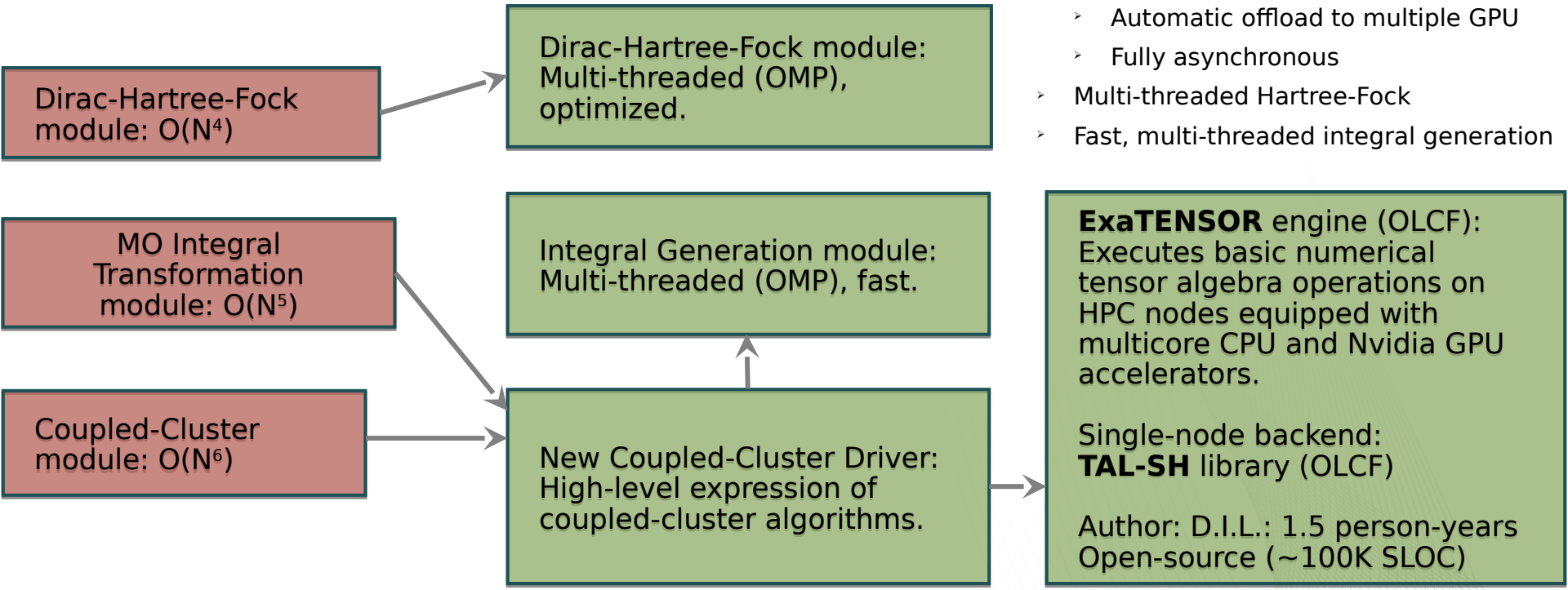
DIRAC team: Lucas Visscher, Hans-Jorgen Jensen, Andre Gomes, Michal Repisky, Stan Papadopoulos

Computational challenges (coupled-cluster module):

- Memory bottleneck: Replicated storage of many tensors (on each process), disabling simulations in which these tensors exceed node memory
- Scalability bottleneck: Overreliance on disk I/O, limiting scalability to *O*(10) nodes
- No explicit multi-threading, MPI only
- No support for accelerators

Accomplishments:

- New coupled-cluster driver allows high-level expression of equations
- Tensor storage and operations are delegated to a new library ExaTENSOR:
  - Fully distributed storage of tensors
  - Task-based execution of tensor operations (hierarchical)
  - Multi-threaded execution on CPU
  - Automatic offload to multiple GPU
  - Fully asynchronous
- Multi-threaded Hartree-Fock
- Fast, multi-threaded integral generation



DIRAC: Relativistic Electronic Structure

DIRAC team: Lucas Visscher, Hans-Jorgen Jensen, Andre Gomes, Michal Repisky, Stan Papadopoulos
OLCF partnership: Dmitry Liakh (ExaTENSOR library)

Smaller UF6 molecule (146 e-):

# Sockets	Time to solution, s
258	3271
514	2357
1026	2090

Larger (UF6)2 molecule (292 e-):

# Sockets	Time to solution, s
1536	23324
2048	18468

Time to solution (s): Smaller UF6 molecule (146 e-): 1026 sockets:

Code Section	CPU only code *	GPU code	Speed-up
Integral Transformation	7032	583	~12x (~9x**)
Coupled-Cluster Doubles	~88590	1507	~58x (~45x**)

* The CPU-only ExaTENSOR code is not optimized yet (expect up to ~5x faster)
** Adjusted value due to some differences in execution configuration

Coupled-Cluster Theory

$$|\Psi\rangle = \exp(\hat{T})|0\rangle = \left(1 + \hat{T} + \frac{1}{2!}\hat{T}^2 + \frac{1}{3!}\hat{T}^3 + \frac{1}{4!}\hat{T}^4 + \cdots\right)|0\rangle$$

$$|\Psi_{excited}\rangle = \hat{R}e^{\hat{T}}|0\rangle$$

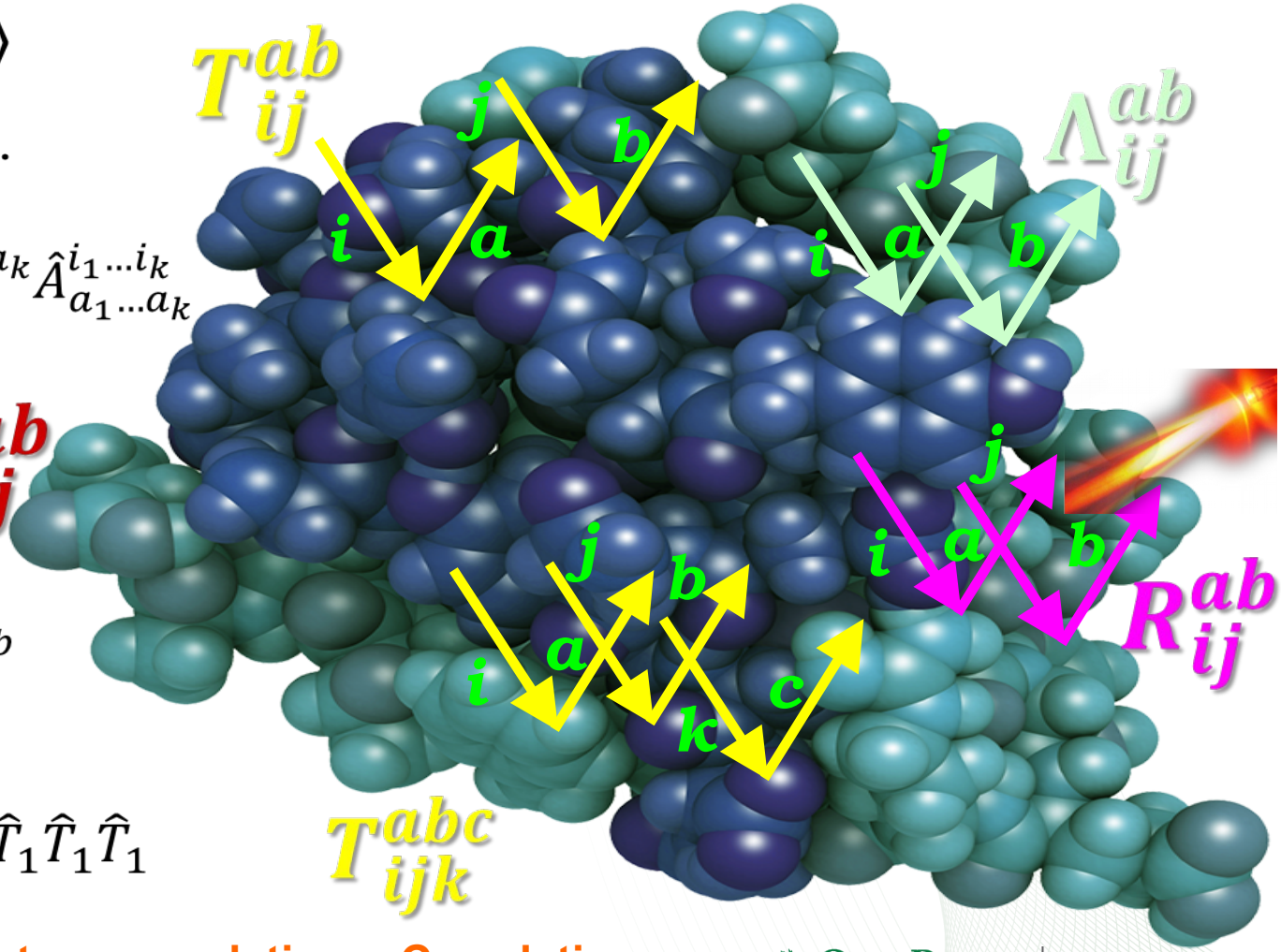
$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \cdots$$

$$\hat{T}_k = \frac{1}{k!\,k!}\sum_{\substack{a_1 \ldots a_k \\ i_1 \ldots i_k}} T_{i_1 \ldots i_k}^{a_1 \ldots a_k} \hat{A}_{a_1 \ldots a_k}^{i_1 \ldots i_k}$$

$$\hat{C}_2 = \hat{T}_2 + \frac{1}{2!}\hat{T}_1\hat{T}_1$$

$$C_{ij}^{ab} = T_{ij}^{ab} + T_i^a \wedge T_j^b$$

$$\hat{C}_3 = \hat{T}_3 + \hat{T}_2\hat{T}_1 + \frac{1}{3!}\hat{T}_1\hat{T}_1\hat{T}_1$$

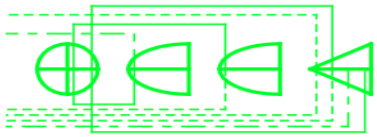


Electron correlation = Correlation between hole-particle excitations

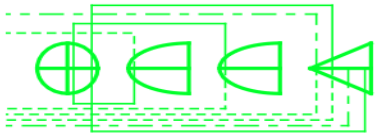
DiaGen: Automated Design and Implementation

```
<domain name="DIP-EOMCC: active space">
set H12=ham(1)+ham(2)
set P0=P()
set Q0=P(2i+;2J+)
set Q1=P(3i+;1a-;2J+)
set Q2=P(4i+;2a-;2J+)
set R0=C(2i-;2J-)
set R1=C(3i-;1a+;2J-)
set R2=C(4i-;2a+;2J-)
set R012=C(2i-;2J-)+C(3i-;1a+;2J-)+C(4i-;2a+;2J-)
set T12=S(1i-;1a+)+S(2i-;2a+)

product Q0*H12*expn(T12,4,8)*R012*P0
 connect(2,3)(2,4)

product Q1*H12*expn(T12,4,8)*R012*P0
 connect(2,3)(2,4)

product Q2*H12*expn(T12,4,8)*R012*P0
 connect(2,3)(2,4)

input H(1i+;1i-)
input H(1i+;1a-)
input H(1a+;1i-)
input H(1a+;1a-)
input H(2i+;2i-)
input H(2i+;1i-;1a-)
input H(2i+;2a-)
input H(1i+;1a+;2i-)
input H(1i+;1a+;1i-;1a-)
input H(1i+;1a+;2a-)
```

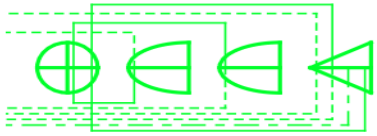
960



962



964



(285) $192.3.896 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^a K_1^a}_{d_1^a,d_2^a} S^{d_1^a}_{I_1^a} S^{d_2^a}_{I_2^a} C^{A_1^b}_{I_1^b,l_1^a K_1^a} \cdot +1/2$

(286) $198.1.932 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^b,l_2^b}_{d_1^b,d_2^b} S^{d_1^b}_{I_1^b} S^{d_2^b}_{l_1^b} C^{A_1^b}_{I_1^a I_2^a,l_2^b}$

(287) $198.2.933 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^b,l_1^a}_{d_1^a,d_1^b} S^{d_1^a}_{I_1^a} S^{d_1^b}_{l_1^b} C^{A_1^b}_{I_2^a I_1^b,l_1^a}$

(288) $198.4.935 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^a,l_1^b}_{d_1^b,d_1^a} S^{d_1^b}_{I_1^b} S^{d_1^a}_{l_1^a} C^{A_1^b}_{I_1^a I_2^a,l_1^b}$

(289) $198.5.936 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^a,l_2^a}_{d_1^a,d_2^a} S^{d_1^a}_{I_1^a} S^{d_2^a}_{l_1^a} C^{A_1^b}_{I_2^a I_1^b,l_2^a}$

(290) $202.1.946 : Z^{A_1^b}_{I_1^a I_2^a I_1^b} += H^{l_1^b,K_1^a}_{d_1^a d_1^b} S^{A_1^b}_{l_1^b} S^{d_1^a d_1^b}_{I_1^a I_1^b} C_{I_2^a,K_1^a}$

(447) $324.85.1.1.3.1.0.20333376.09 : Z^{l_1^b}_{I_1^a I_2^a i_1^b} += H^{l_1^b,l_2^b}_{i_1^b,d_1^b} C^{d_1^b}_{I_1^a I_2^a,l_2^b} \cdot -1.$

(448) $331.86.1.1.2.1.0.10042704.09 : Z^{l_1^b}_{I_1^a I_2^a i_1^b} += H^{l_1^b,K_1^a}_{I_1^a,d_1^b} C^{d_1^b}_{I_2^a i_1^b,K_1^a} \cdot -1.$

(449) $325.85.1.1.3.1.0.20333376.09 : Z^{l_1^b}_{I_1^a I_2^a i_1^b} += H^{l_1^b,l_1^a}_{i_1^b,d_1^a} C^{d_1^a}_{I_1^a I_2^a,l_1^a} \cdot -1.$

(450) $821.177.2.1.2.1.0.49593600.07 : Z^{l_1^b}_{I_1^a I_2^a i_1^b} += S^{d_1^b}_{i_1^b} R^{l_1^b}_{I_1^a I_2^a,d_1^b} \cdot -1.$

(451) $938.199.1.2.2.2.0.11716488.10 : R^{l_1^b,K_1^a}_{I_1^a i_1^b} += H^{l_1^b,K_1^a}_{d_1^a d_1^b} S^{d_1^a d_1^b}_{I_1^a i_1^b}$

Math Framework: Basic Tensor Algebra

- Formal tensor: $T^{pq\ldots}_{rs\ldots} \mapsto T(p,q,r,s,\ldots)$: n-D Array

  Full tensor: $T(p,q,r,s): p \in P, q \in Q, r \in R, s \in S$

  Tensor slice: $T(p,q,r,s): p \in P' \subseteq P, q \in Q' \subseteq Q,$
  $r \in R' \subseteq R, s \in S' \subseteq S$

Few primitive operations:

- Tensor addition:
  $\forall p,q,r,s: \; T^{pq}_{rs} = L^{pq}_{rs} + R^{pq}_{rs}$

Parallelism!

- Tensor product:
  $\forall p,q,r,s: \; T^{pq}_{rs} = L^{p}_{r} R^{q}_{s}$

Parallelism!

- Tensor contraction:
  $\forall p,q,r,s: \; T^{pq}_{rs} = L^{pai}_{bcd} R^{qbcd}_{rsai}$

Compute intensive (potentially)!

Math Framework: Tensor Decompositions

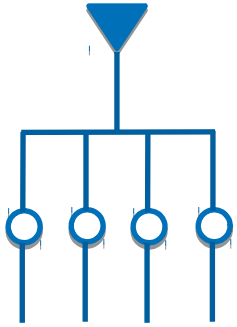
- Graphical (diagrammatic) representation:

Matrix:  Matrix*Matrix: 

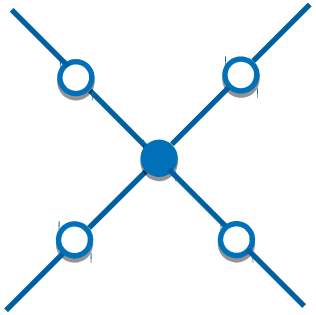
- Linear algebra: SVD is optimal in the 2-norm:

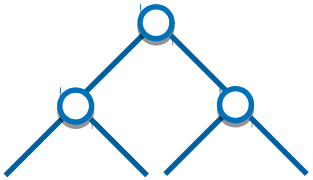
- Tensor (multi-linear) algebra: Many choices:



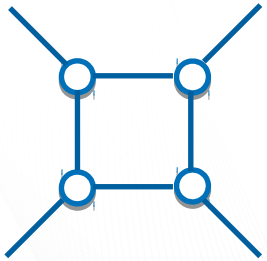
Canonical polyadic



Tucker



Tensor tree

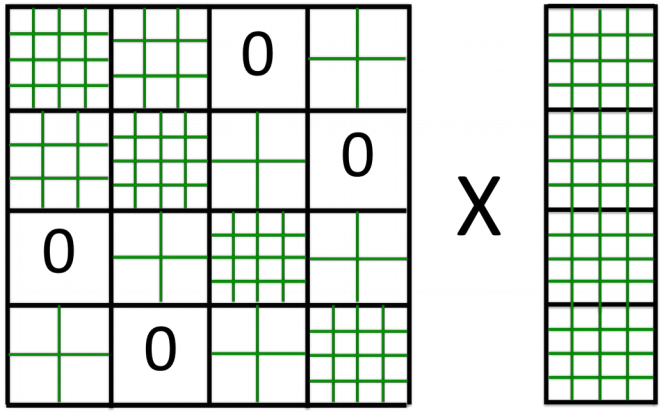


Tensor train (ring MPS)

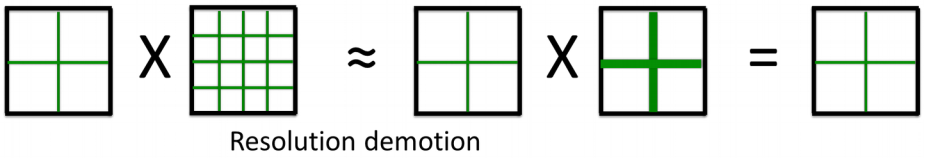
Adaptive (+Hierarchical) Tensor Algebra

Inspired by Multiresolution Analysis

Adaptive dynamic block-sparse representation of many-body tensors

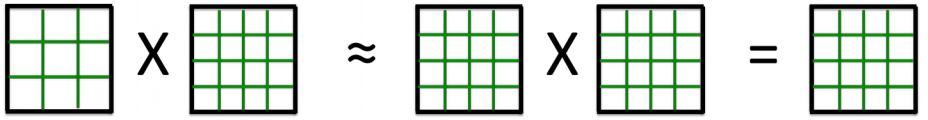


X

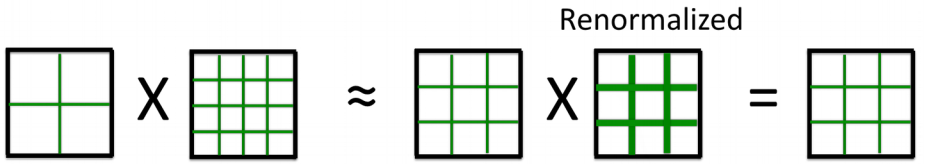


Renormalized

Resolution demotion



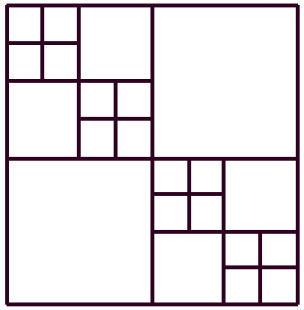
Resolution promotion

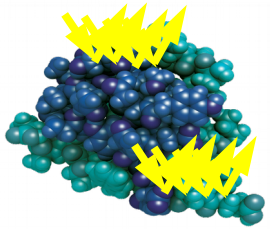


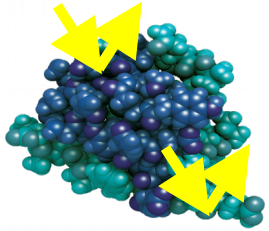
Renormalized

Resolution intermediate match

The resolution of each tensor block can be dynamically adjusted if needed in a specific tensor operation (computational cost reduction)





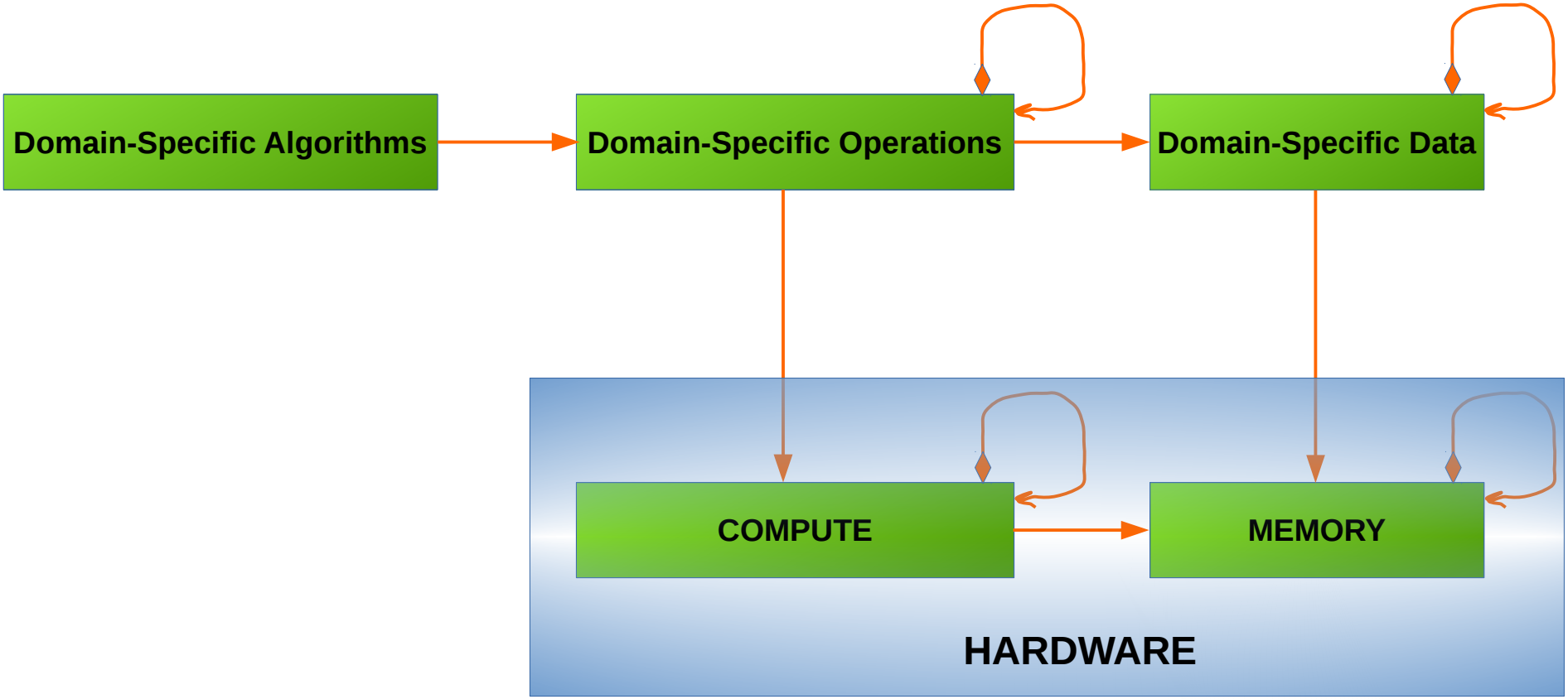


Extrapolation of H/H2-matrix algebra to TENSORS

- Large tensor elements can become tensors themselves (higher resolution);
- Weak tensor slices can be compressed by lowering the resolution, up to a single (complex) number;
- Adapt to the calculated electronic state and available HPC resources;
- Should be better than just black-and-white discarding.

D.I.L. Int. J. Quantum Chem. 114, 1607 (2014); D.I.L. Mol. Phys. 116, 588 (2017)

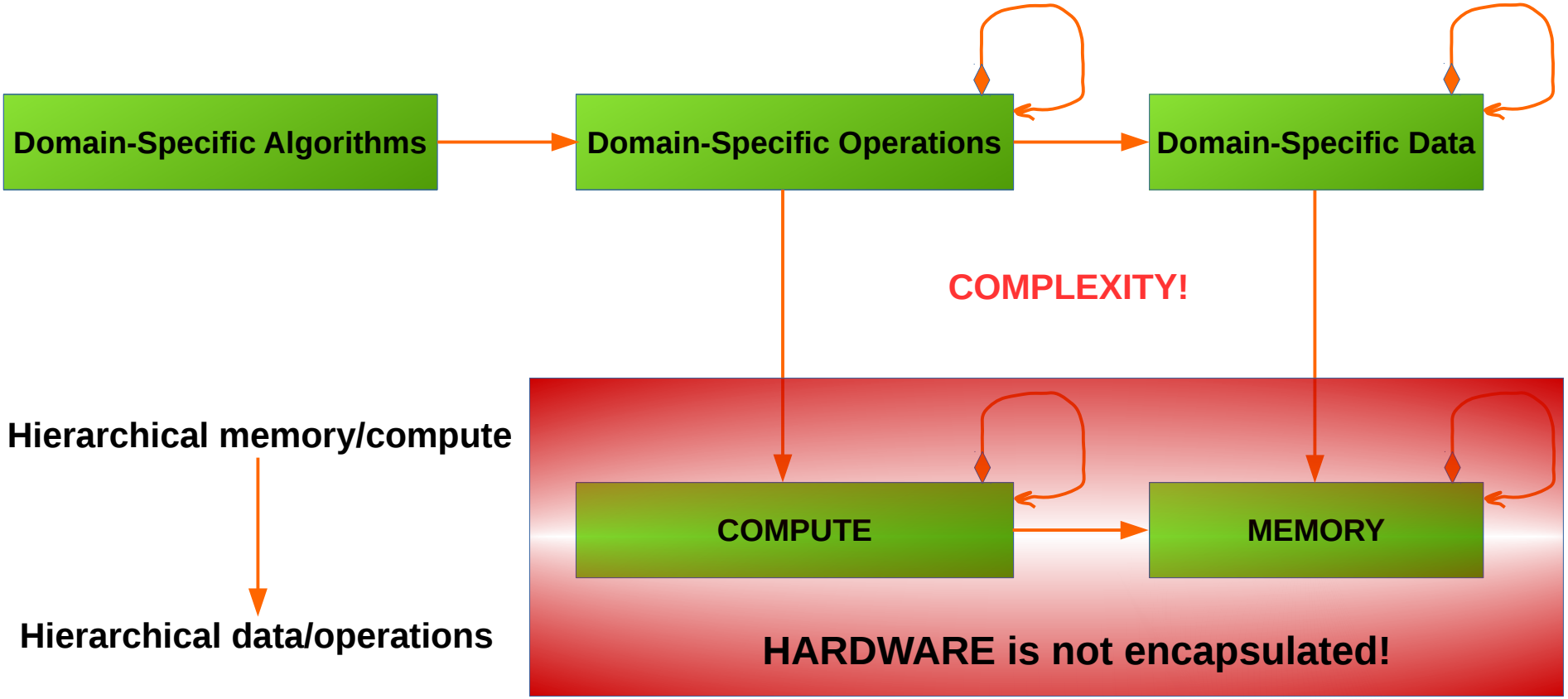
Portable Software Architecture



PORTABILITY: Multiple targets, one code, maybe minor extension (not modification)

PERFORMANCE: Minimization/optimization of data movement to keep compute busy:
Optimal mapping of data and operations

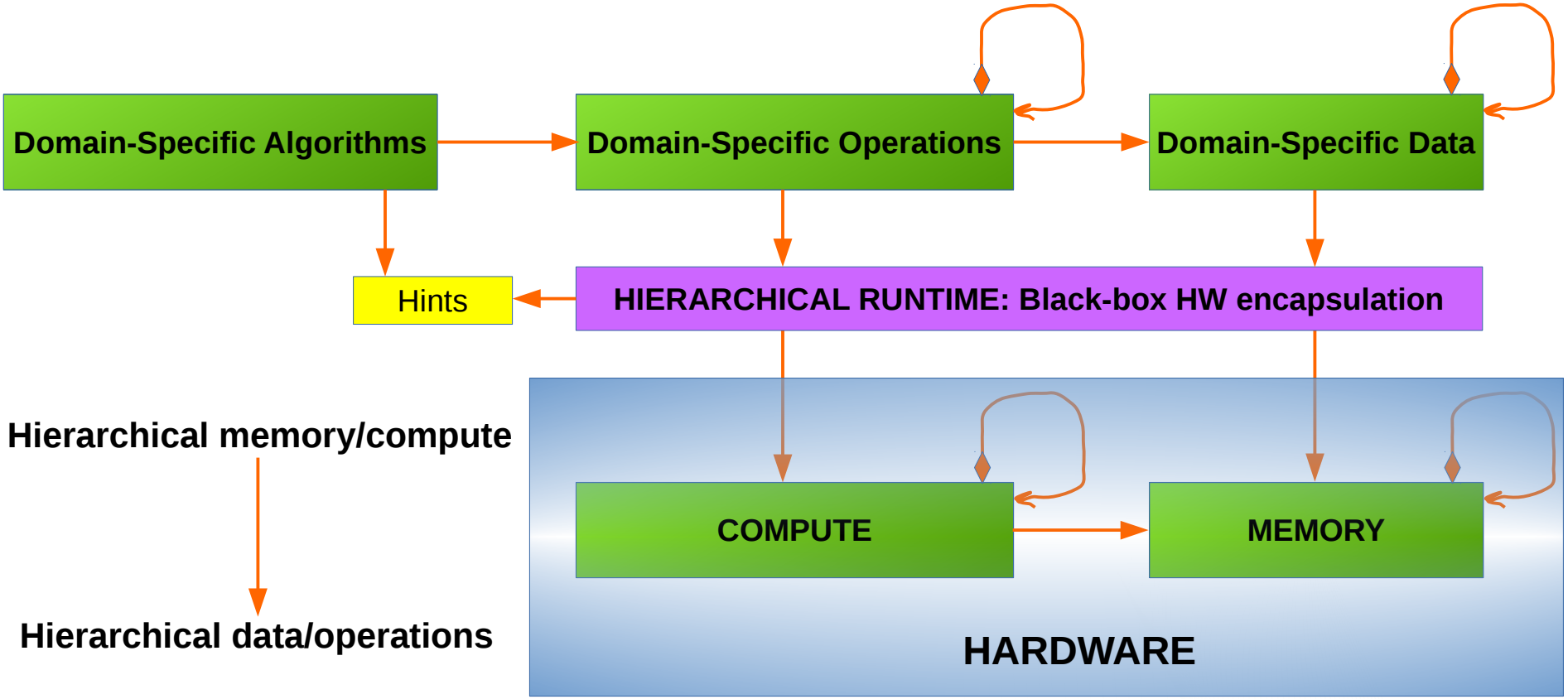
Portable Software Architecture



PORTABILITY: Multiple targets, one code, maybe minor extension (not modification)

PERFORMANCE: Minimization/optimization of data movement to keep compute busy:
Optimal mapping of data and operations

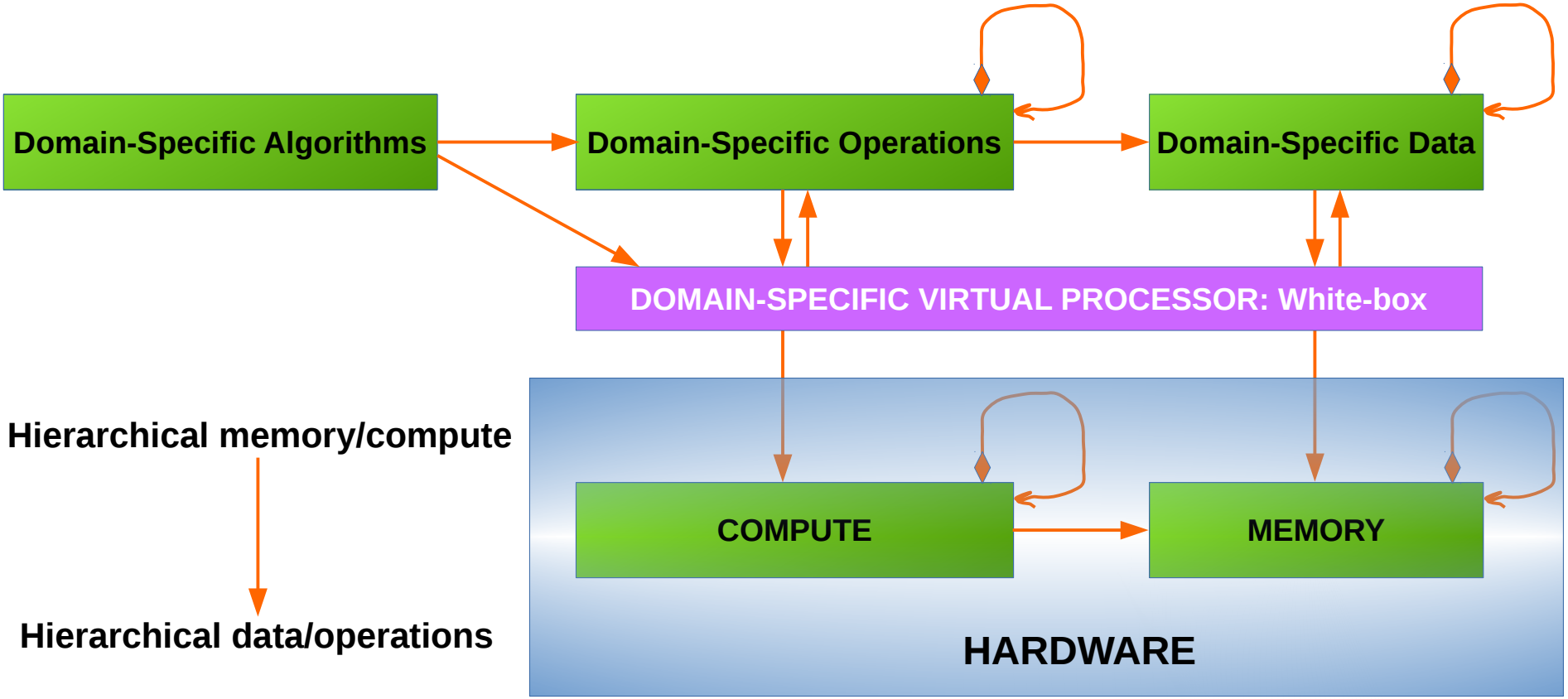
Portable Software Architecture



PORTABILITY: Multiple targets, one code, maybe minor extension (not modification)

PERFORMANCE: Minimization/optimization of data movement to keep compute busy:
Optimal mapping of data and operations

Portable Software Architecture

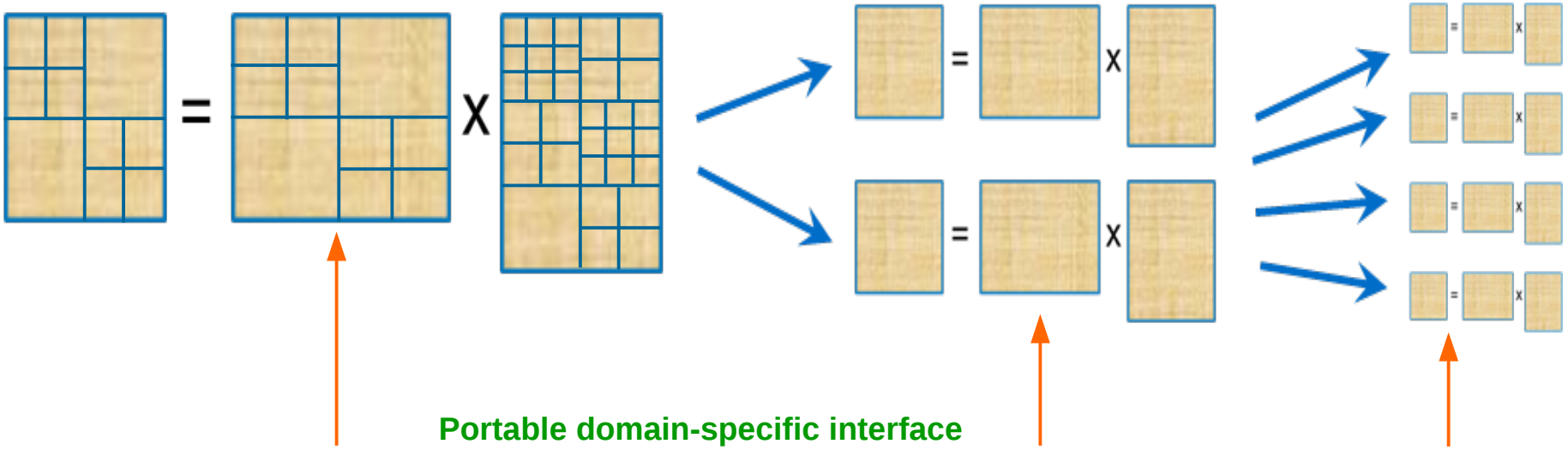


PORTABILITY: Multiple targets, one code, maybe minor extension (not modification)

PERFORMANCE: Minimization/optimization of data movement to keep compute busy:
Optimal mapping of data and operations

Portable Software Architecture

Domain-specific algorithmic workload (*e.g.* hierarchical tensor algebra)



Portable domain-specific interface

HIERARCHICAL DOMAIN-SPECIFIC VIRTUAL PROCESSOR

Domain-specific co-design

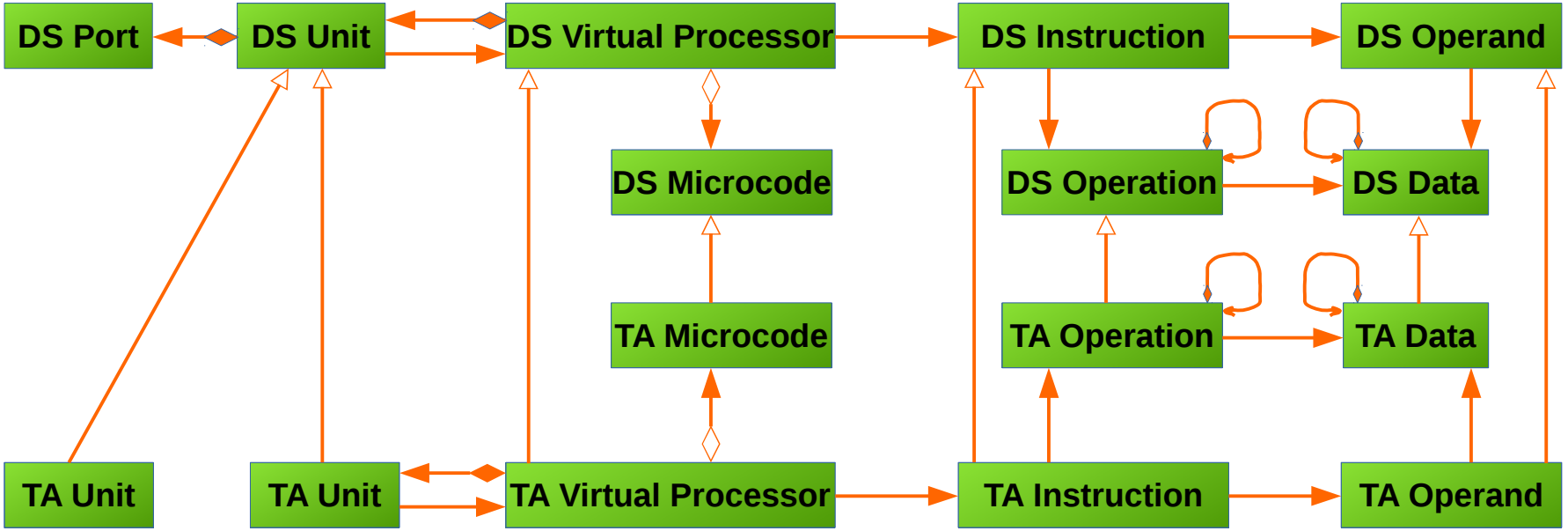
HIERARCHICAL COMPUTE | **HIERARCHICAL MEMORY** | **HIERARCHICAL INTERCONNECT**

HARDWARE

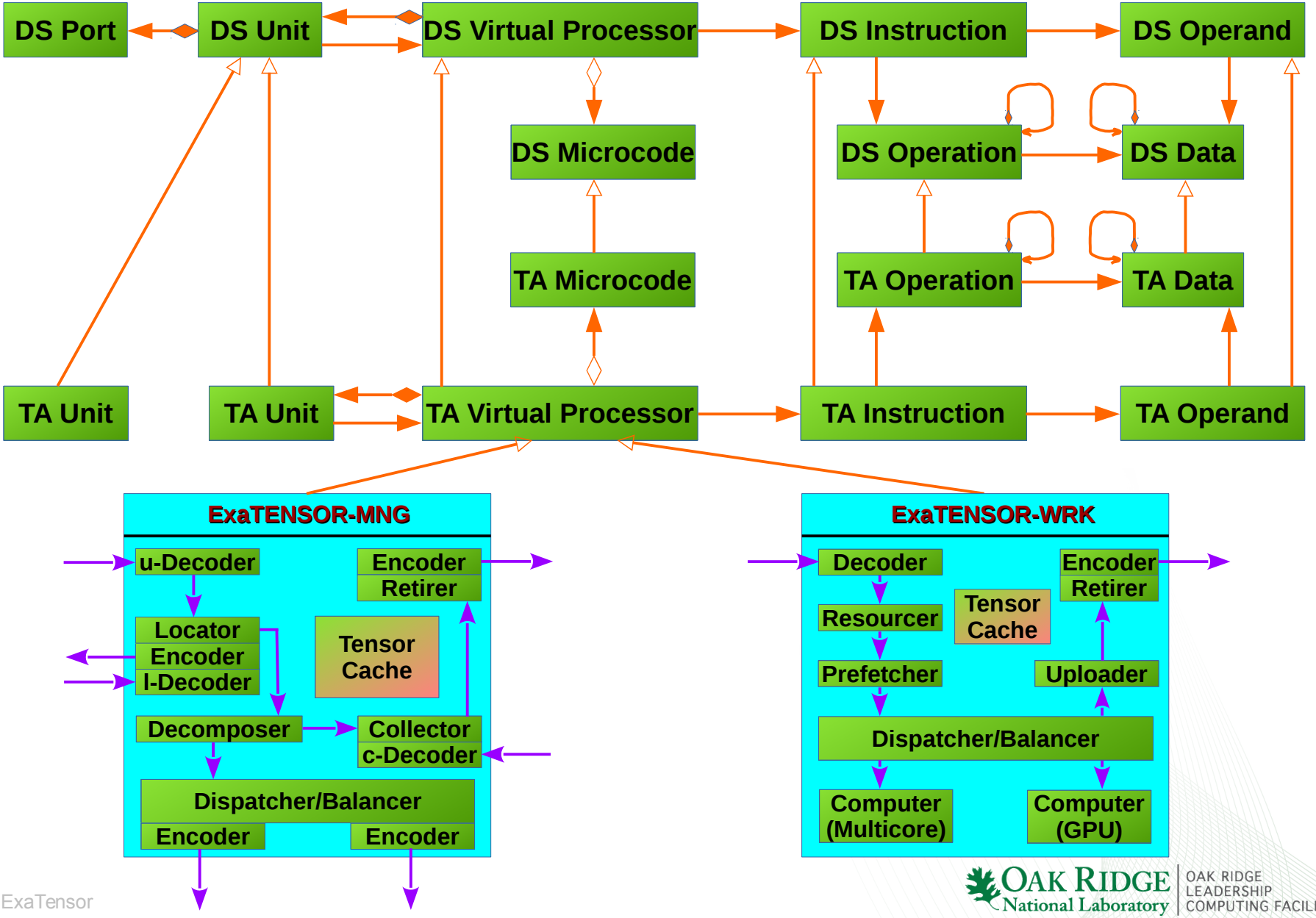
Domain-Specific Virtual Processor

- Generalization, elaboration, and formalization of previous efforts
- Explicitly structured runtime system that formally resembles a **processor**, but specialized to **domain-specific** workloads
- Virtualizes physical hardware by encapsulating it with a **static** virtual processing architecture **best suited** for domain algorithms
- Encapsulates a wide class of HPC architectures via a virtual node architecture **template**: Opportunities for **co-design**
- Understands the specificity of domain data, operations and algorithms: Better opportunity for optimization (**performance**)
- Domain algorithms are expressed **once**, either via a standalone or an embedded DSL, then compiled and executed (or interpreted)
- Debugging/profiling in terms of domain-specific abstractions

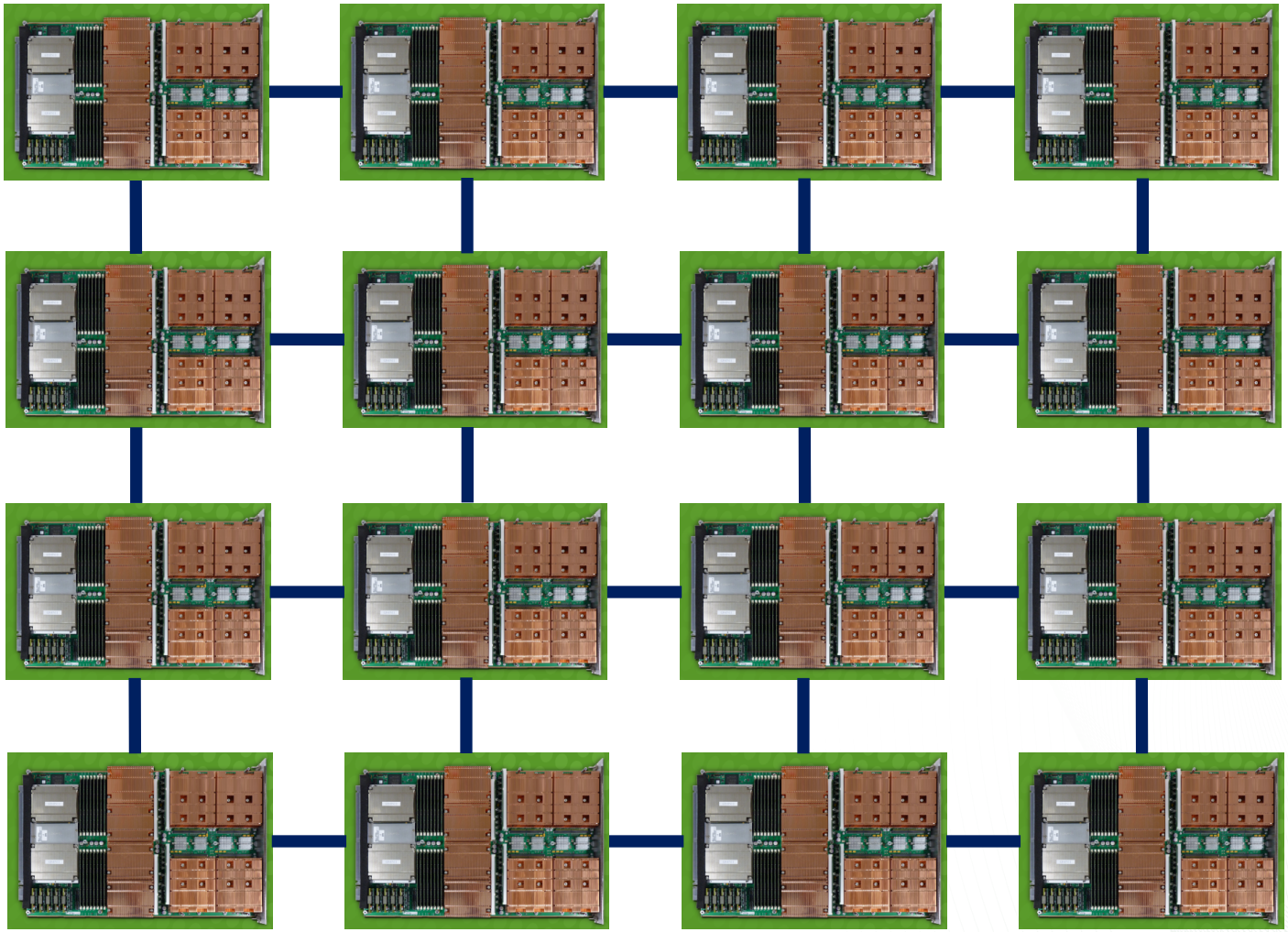
Domain-Specific Virtual Processor Architecture



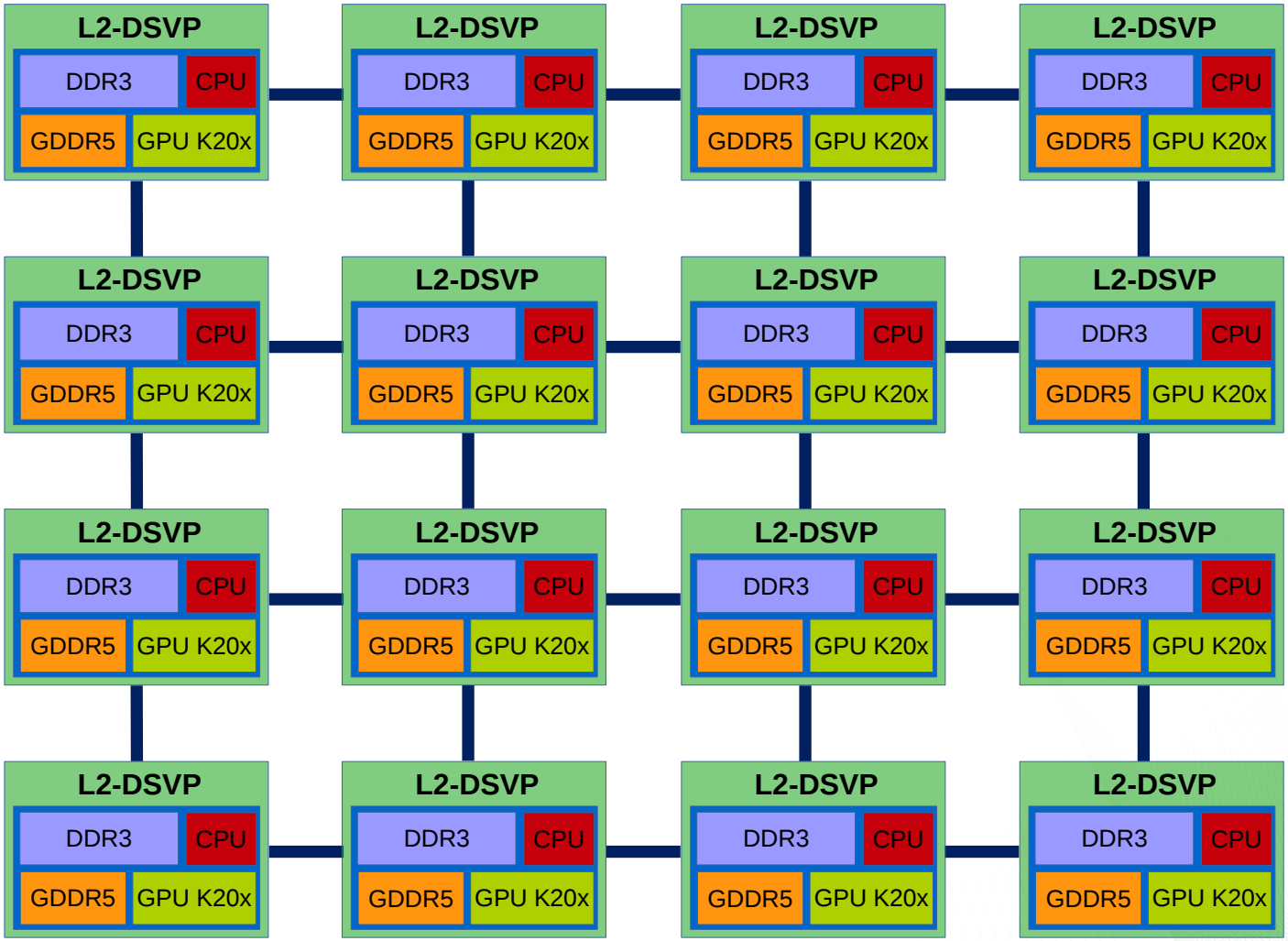
Domain-Specific Virtual Processor Architecture



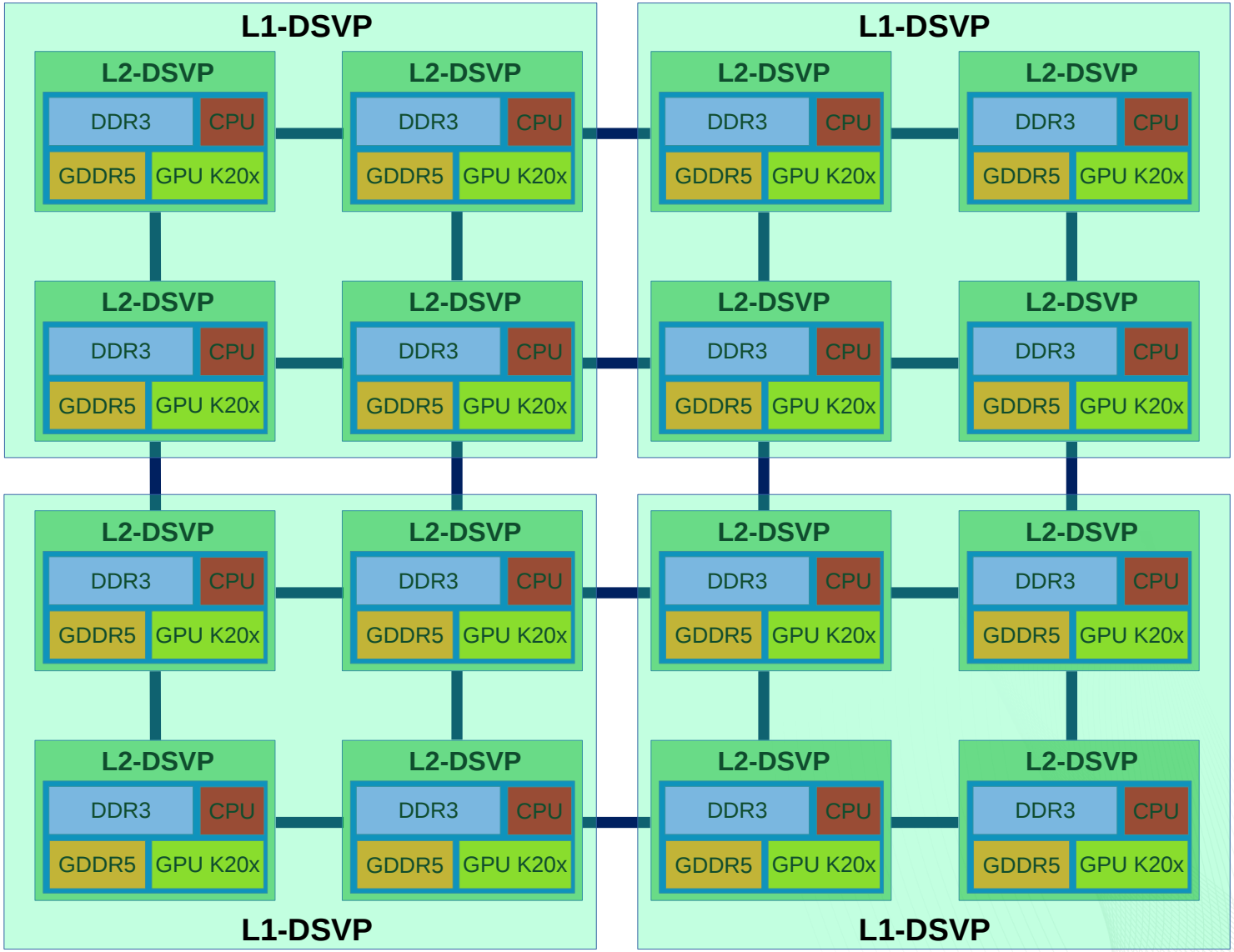
Global Virtualization: Hiding HPC Scale



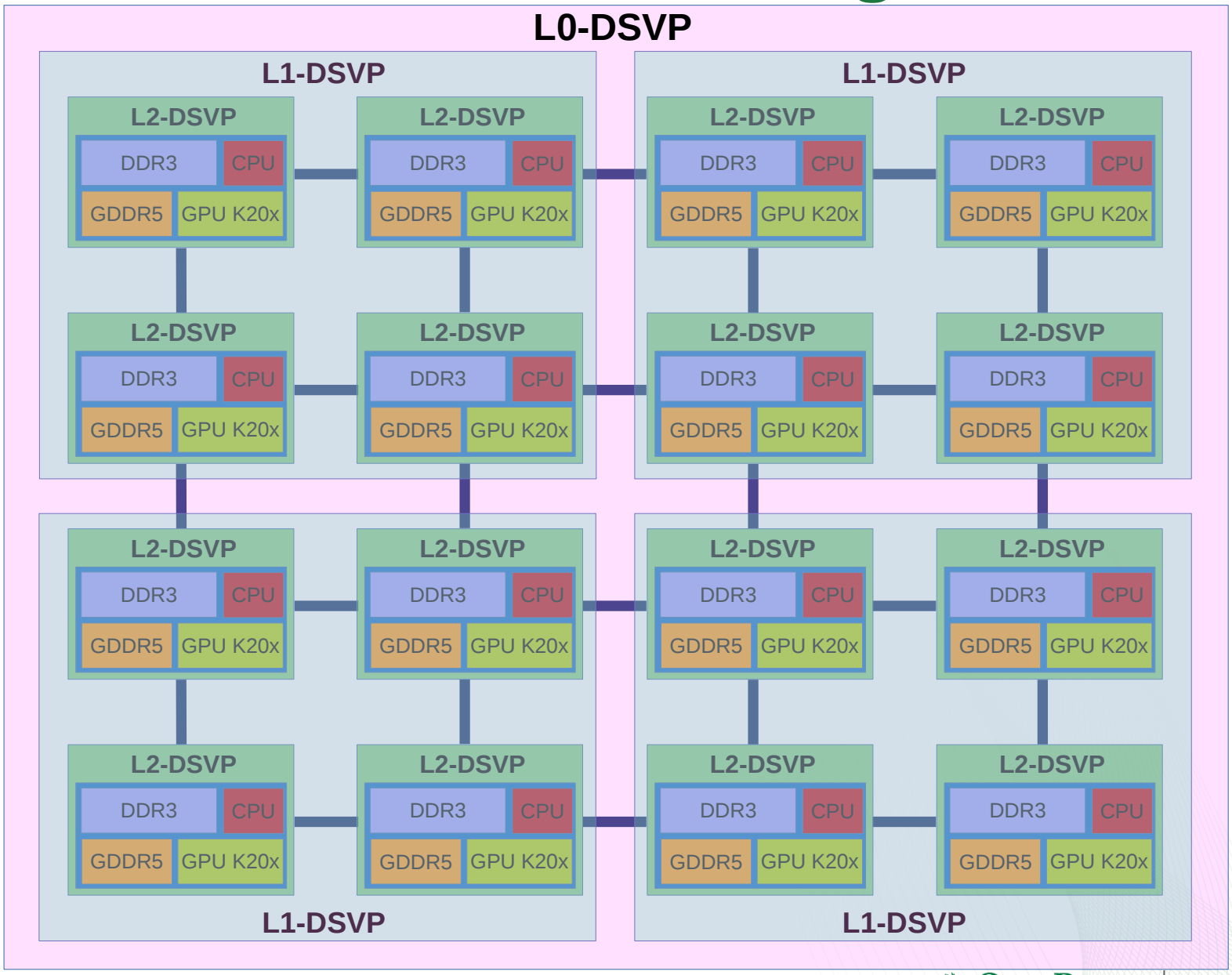
Global Virtualization: Hiding HPC Scale



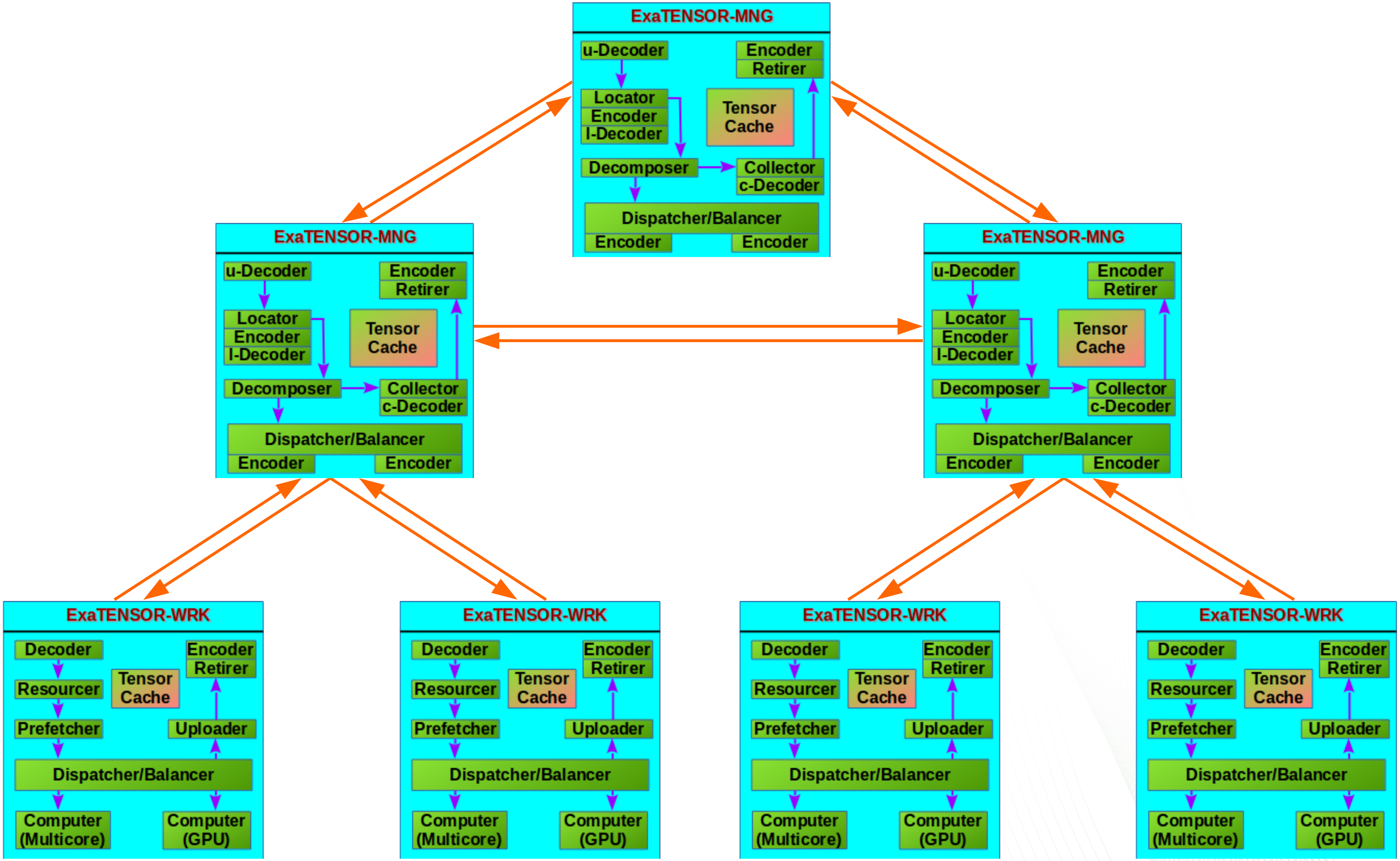
Global Virtualization: Hiding HPC Scale



Global Virtualization: Hiding HPC Scale

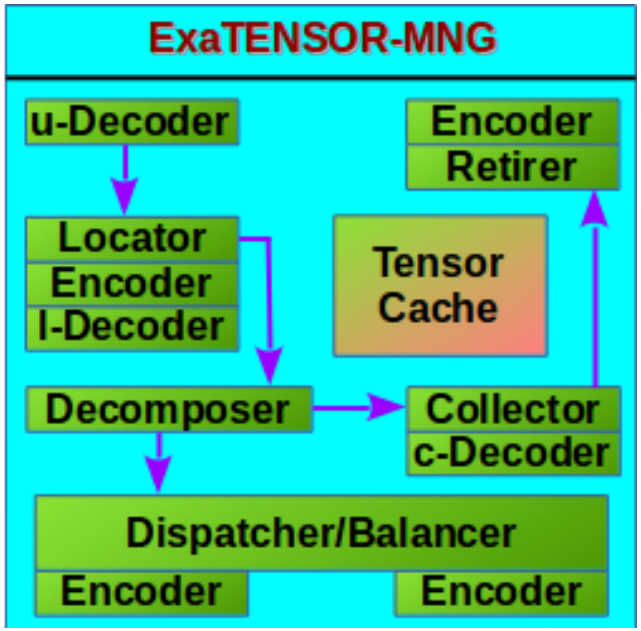


Hierarchical Virtualized HPC Platform

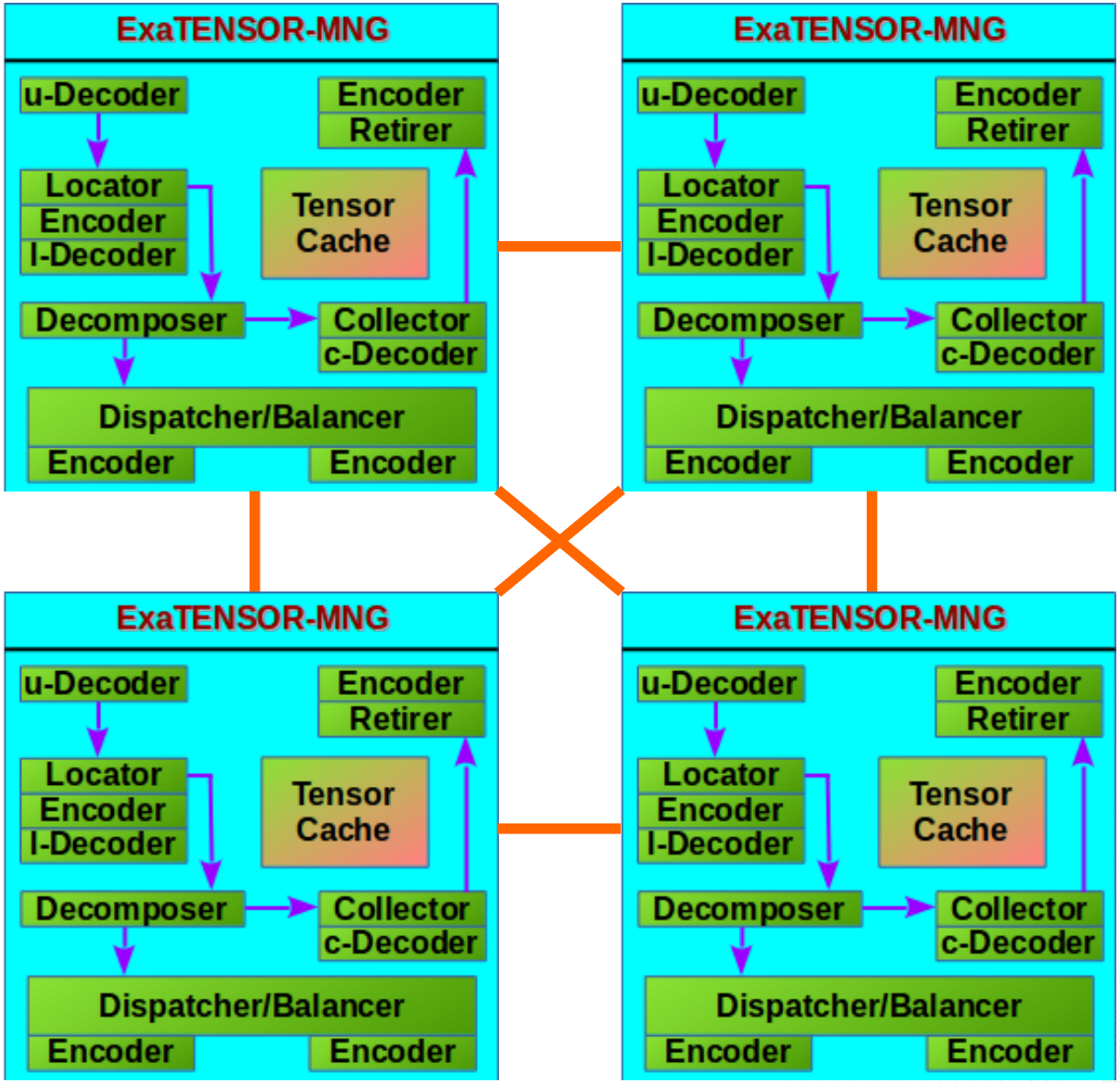


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

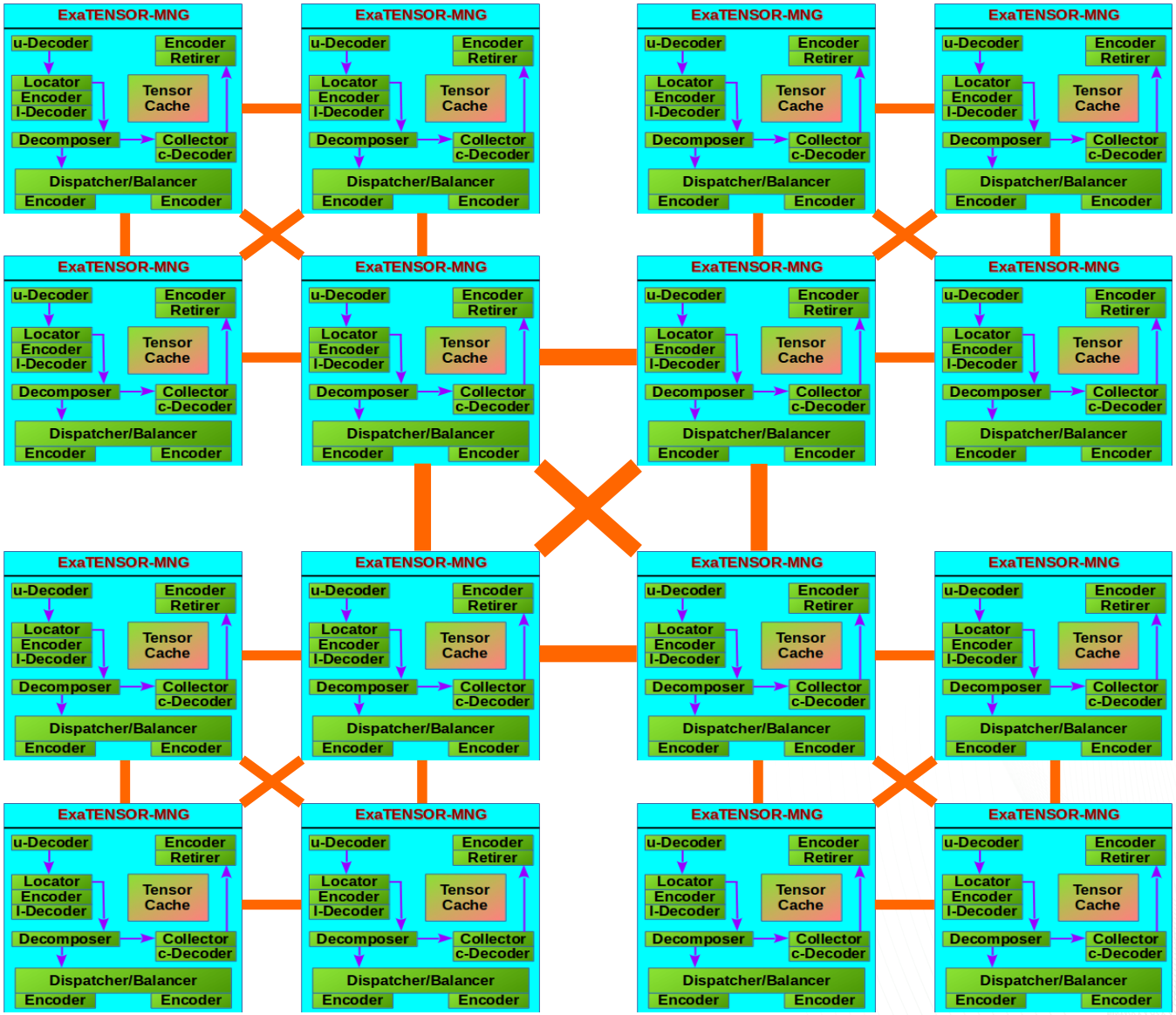
Recursive HPC System Virtualization



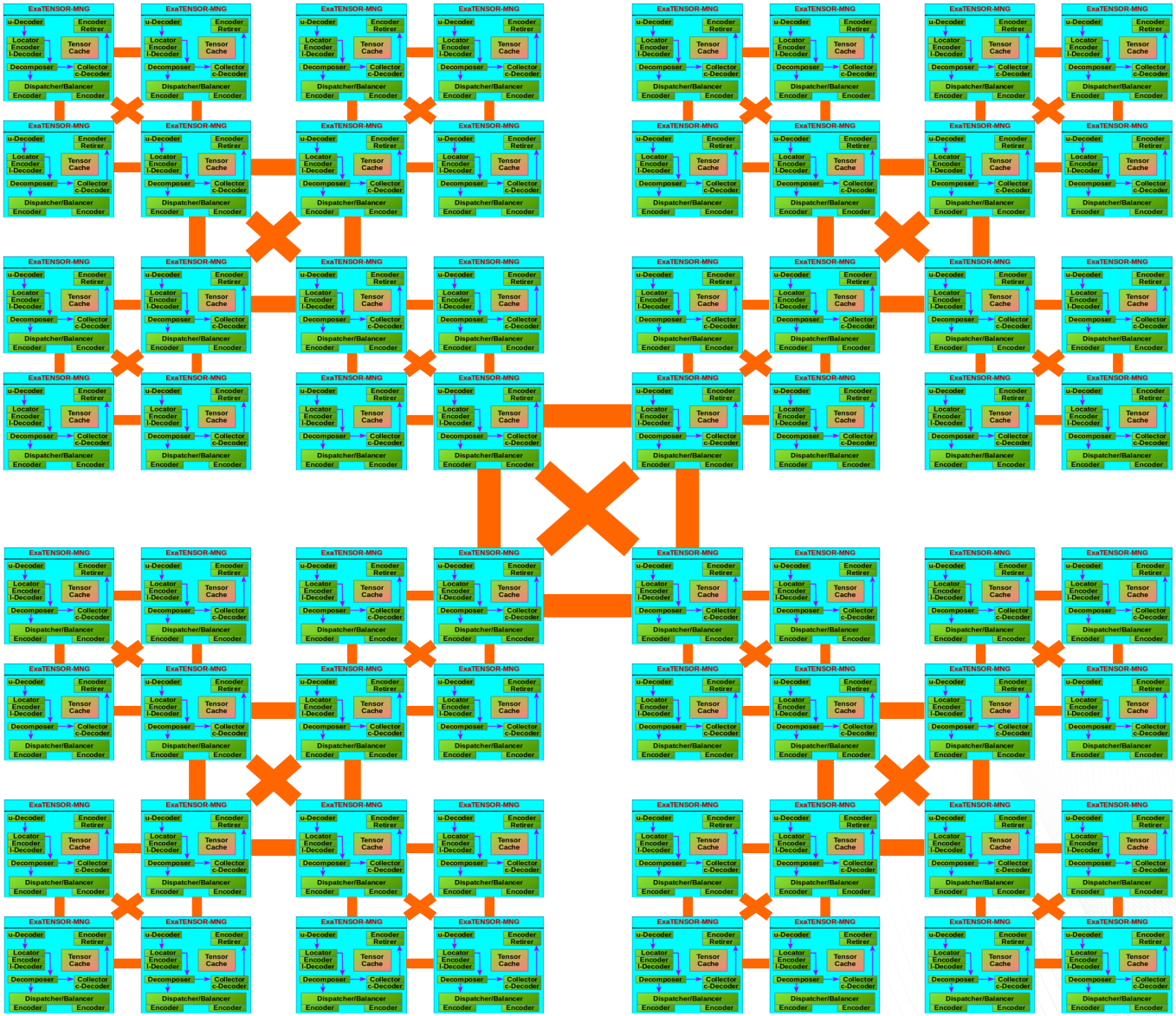
Recursive HPC System Virtualization



Recursive HPC System Virtualization

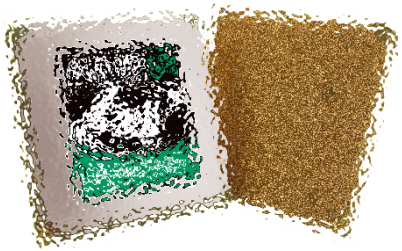


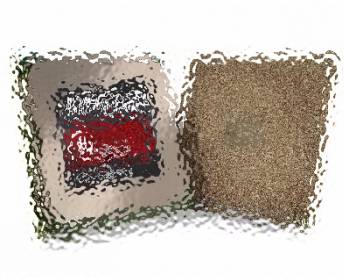
Recursive HPC System Virtualization

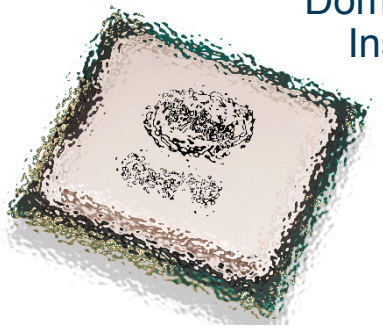


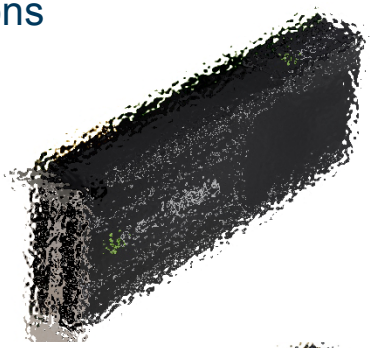
Node-Level Virtualization: Hiding Hardware

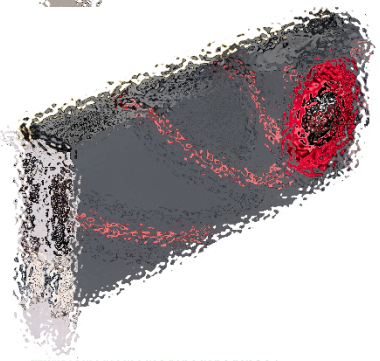
Domain-Specific Virtual Processor (DSVP)

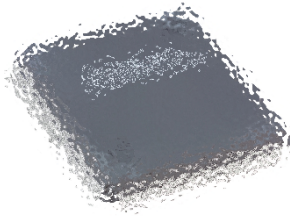


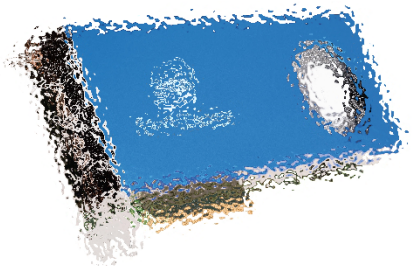


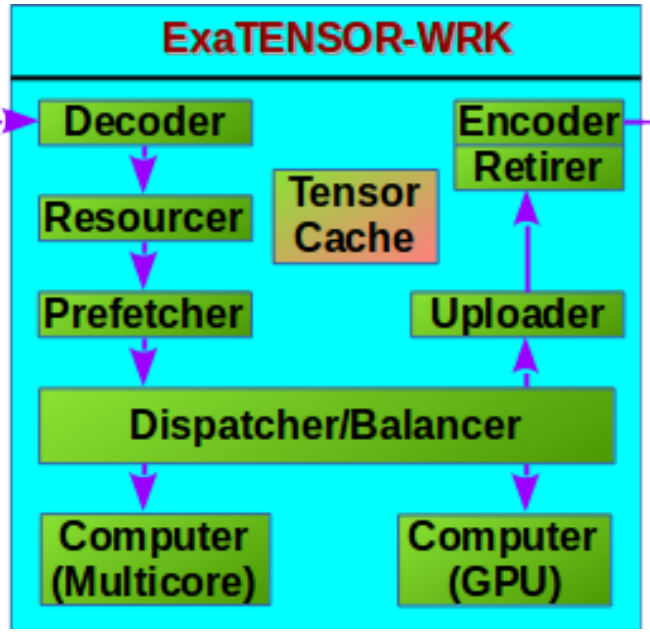








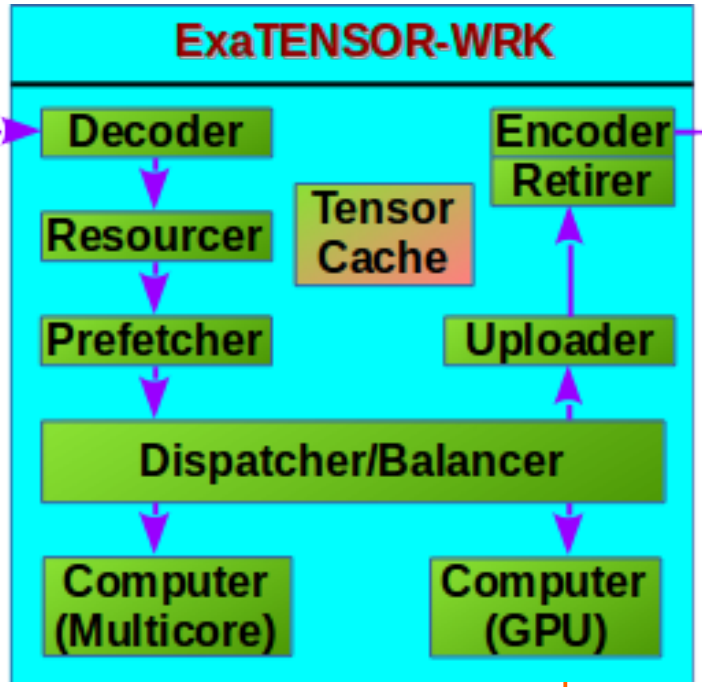




Domain-Specific Instructions

Domain-Specific Instructions

Node-Level Virtualization: Hiding Hardware



Domain-Specific Instructions

Domain-Specific Instructions

	Tesla V100 for NVLink	Tesla V100 for PCIe
PERFORMANCE with NVIDIA GPU Boost™	DOUBLE-PRECISION 7.8 TeraFLOPS SINGLE-PRECISION 15.7 TeraFLOPS DEEP LEARNING 125 TeraFLOPS	DOUBLE-PRECISION 7 TeraFLOPS SINGLE-PRECISION 14 TeraFLOPS DEEP LEARNING 112 TeraFLOPS
INTERCONNECT BANDWIDTH Bi-Directional	NVLINK 300 GB/s	PCIE 32 GB/s
MEMORY CoWoS Stacked HBM2	CAPACITY 16 GB HBM2 BANDWIDTH 900 GB/s	

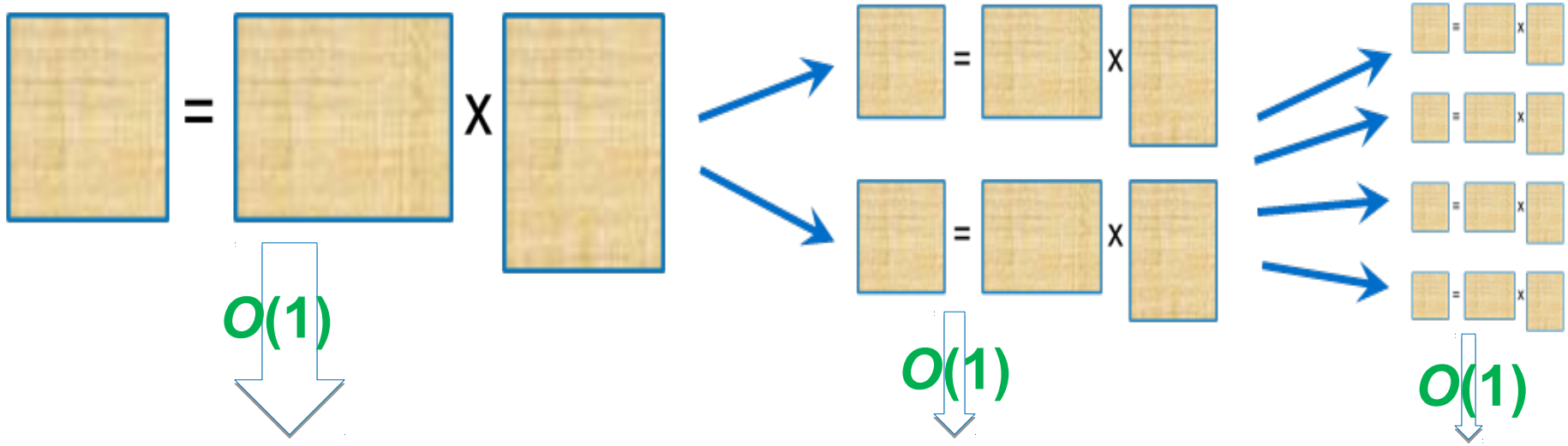
TENSOR ALGEBRA DRIVER for Multicore CPU and NVIDIA GPU: TAL-SH library: (tensor algebra primitives = domain-specific microcode)

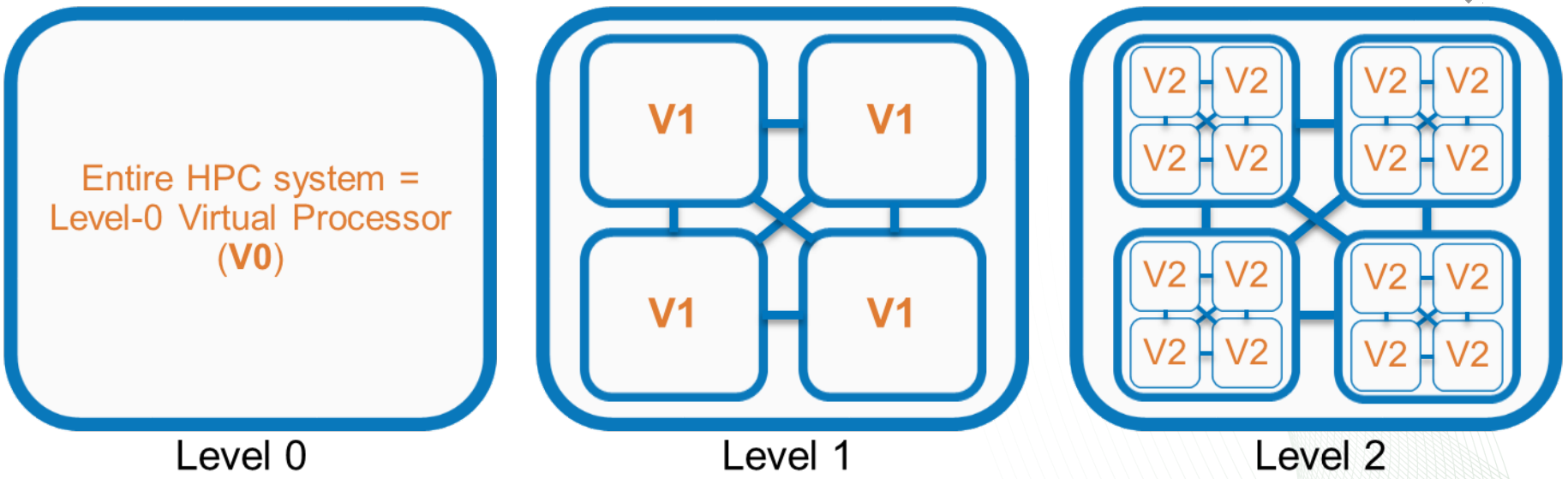
https://github.com/DmitryLyakh/TAL_SH.git

$$\forall p,q,r,s: \quad T^{pq}_{rs} = L^{pai}_{bcd} R^{qbcd}_{rsai}$$

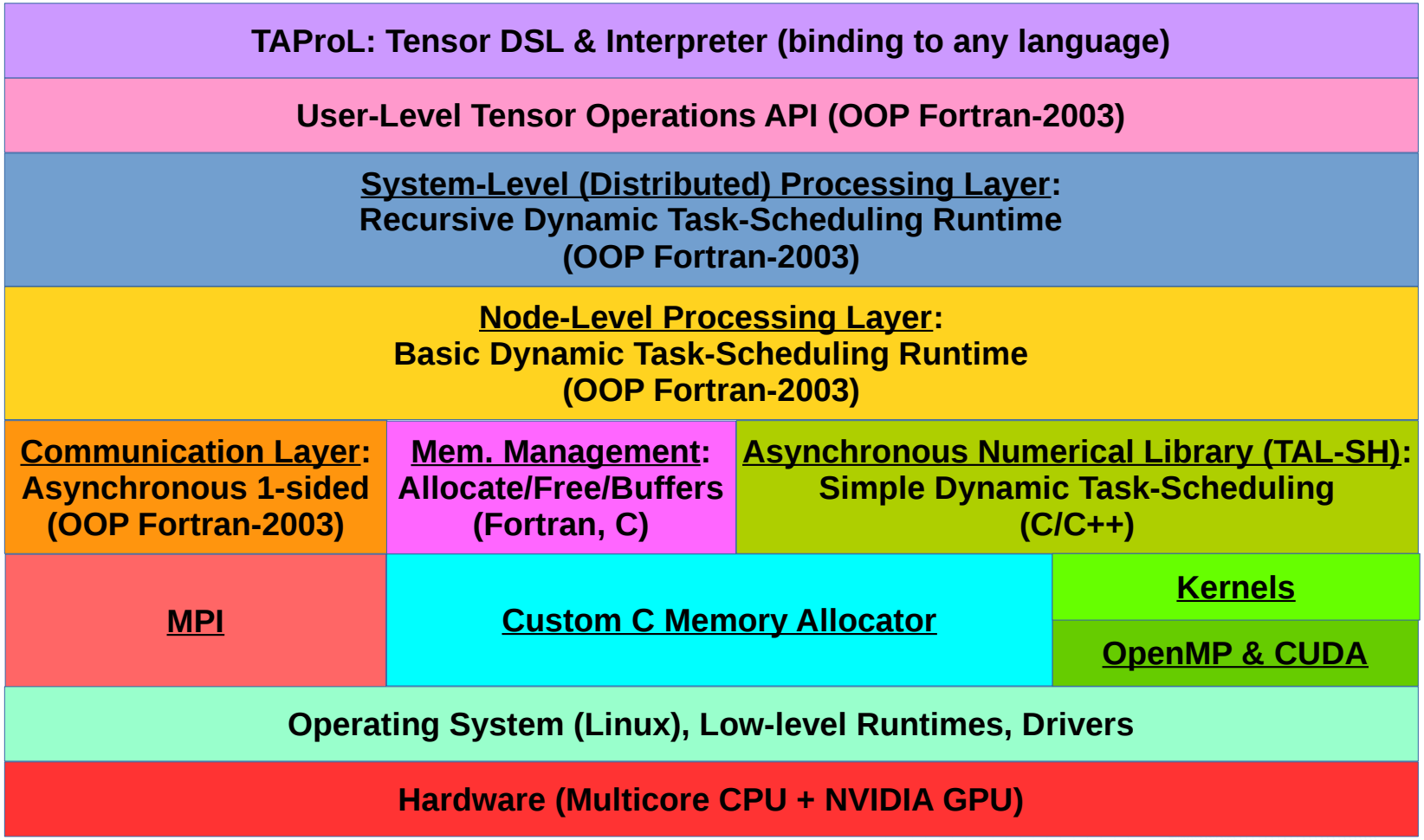
Hierarchical Dynamic Task Scheduling

Data storage granularity is decoupled from the task granularity





ExaTENSOR: Software Layers



Raw ExaTENSOR performance on Summit w/o optimized communication:

(O^2)(V^4) tensor contraction: O=280, V=980

Nodes	Procs	Time, s
259	512	742.79
517	1024	434.66
1033	2048	200.90
1549	3072	126.08

TAL-SH: Software Layers

TAL-SH FORTRAN-95 INTERFACE

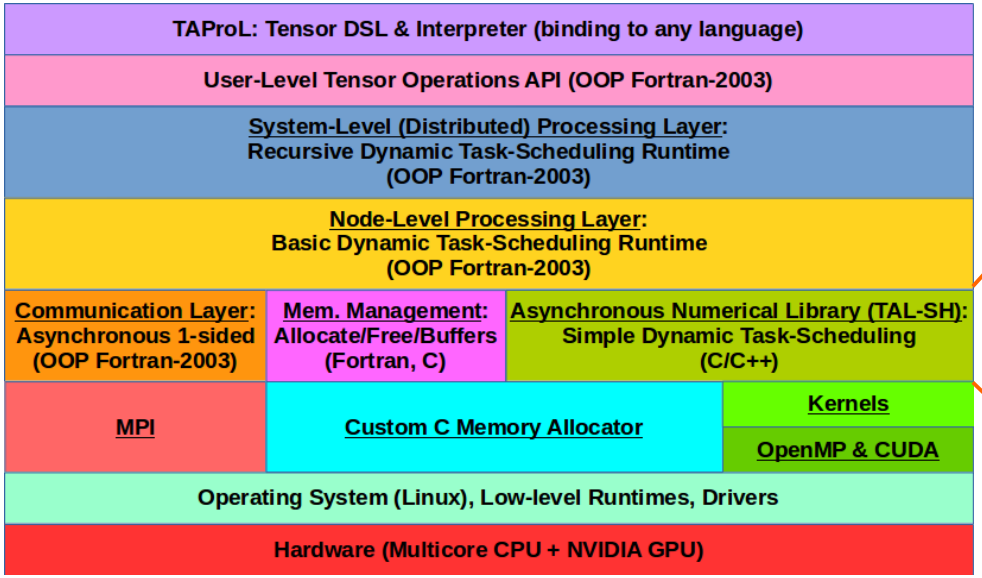
```
ierr=talsh_tensor_construct(dtens,R8,(/33,33,33,33/),init_val=(0d0,0d0))
ierr=talsh_tensor_construct(ltens,R8,(/420,33,33/),init_val=(1d-3,0d0))
ierr=talsh_tensor_construct(rtens,R8,(/420,33,33/),init_val=(1d-2,0d0)
ierr=talsh_tensor_contract_xl(
     'D(a,b,c,d)+=L(p,c,a)*R(p,b,d)',
     dtens,ltens,rtens,
     scale=(5d-1,0d0),
     dev_id=0,dev_kind=DEV_NVIDIA_GPU)
```

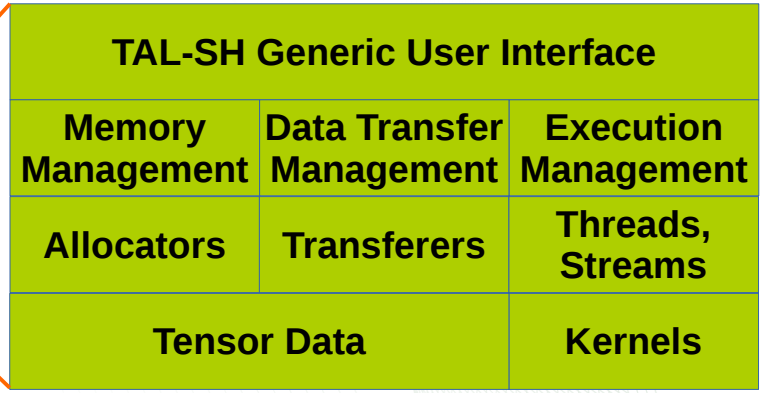
TAL-SH C++ INTERFACE

```
talsh::Tensor dtens({74,156,74,156},std::complex<float>{0.0f,0.0f});
talsh::Tensor ltens({74,156,74,156},std::complex<float>{0.1f,0.2f});
talsh::Tensor rtens({74,156,74,156},std::complex<float>{0.3f,0.4f});
int ierr = dtens.contractAccumulateXL(
           nullptr,
           std::string("D(i,a,j,b)+=L(j,a,k,c)*R(k,b,i,c)"),
           ltens, rtens,
           DEV_NVIDIA_GPU, DEV_DEFAULT,
           std::complex<float>{0.5f,0.0f});
bool done = dtens.sync();
```

TAL-SH: Tensor Algebra library

- Basic tensor algebra operations: Tensor contraction, addition, etc.
- Real/Complex, Single/Double precision
- C, C++, Fortran-95 interfaces
- Multicore CPU, NVIDIA GPU, self-hosted Intel Xeon Phi (KNL)
- Support of multi-GPU nodes
- Asynchronous execution of tensor operations on GPU (TAL-SH tasks)
- Automatic management of Host/Device memory allocations and asynchronous Host-Device and Device-Device data transfers
- Single- and multi-GPU execution of tensor contractions with large tensors which do not fit in GPU memory
- Fast GPU tensor transpose via cuTT library
- Support of NVIDIA cuTensor backend

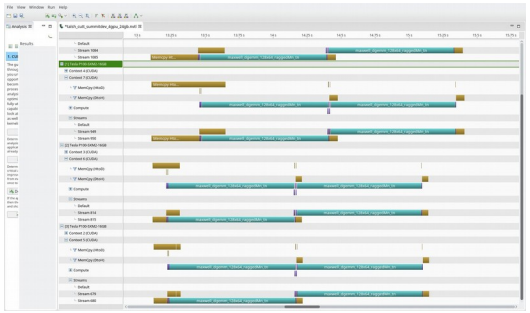




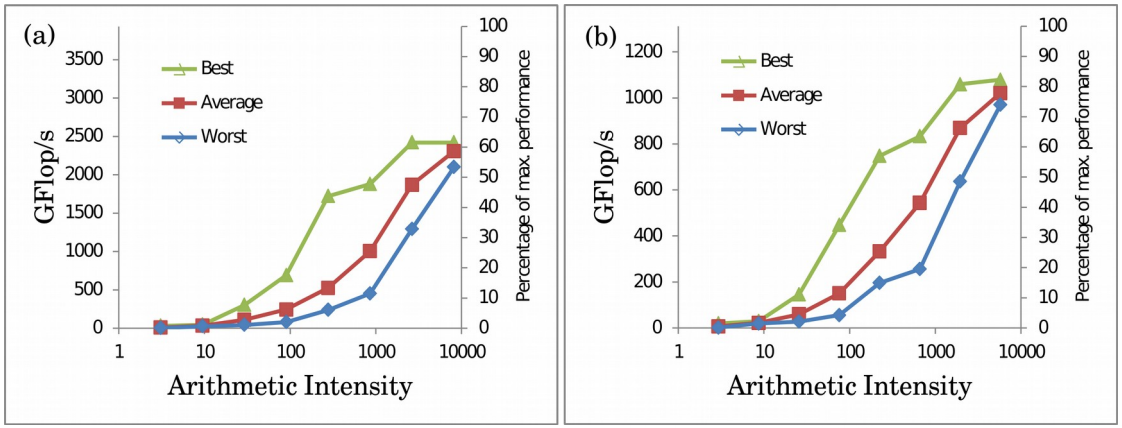
TAL-SH: Intra-Node Tensor Algebra Library

Performance on CPU and GPU in Gflop/s

Device	Complex Single Prec.	Complex Double Prec.
Sandy Bridge	679.9	326.9
1 GPU K20x	1872.8	659.9
1 GPU V100	14041.7	7119.6
2 GPU V100	27759.9	14018.5
3 GPU V100	40430	20175.9

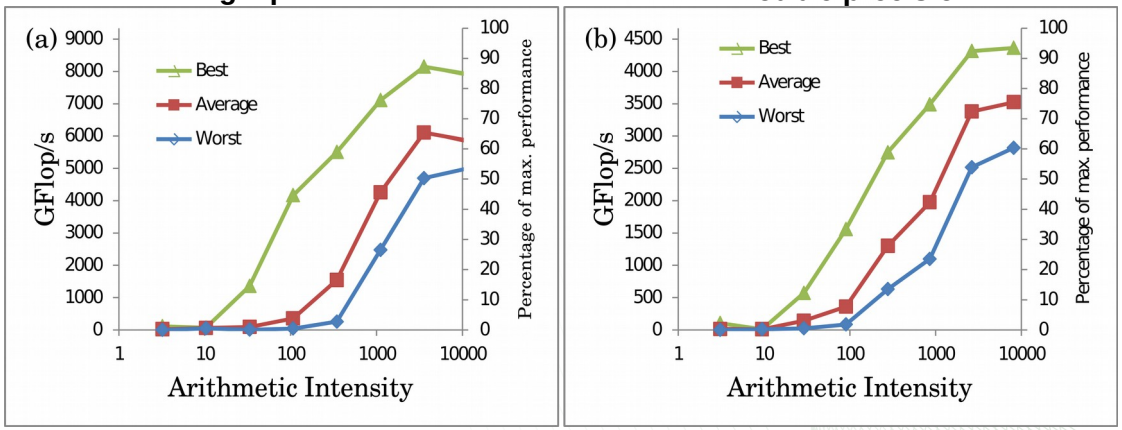


Kepler K20x



Single precision **Double precision**

Pascal P100



TAL-SH FORTRAN-95 INTERFACE

```
ierr=talsh_tensor_construct(dtens,R8,(/33,33,33,33/),init_val=(0d0,0d0))
ierr=talsh_tensor_construct(ltens,R8,(/420,33,33/),init_val=(1d-3,0d0))
ierr=talsh_tensor_construct(rtens,R8,(/420,33,33/),init_val=(1d-2,0d0)
ierr=talsh_tensor_contract_xl(
     'D(a,b,c,d)+=L(p,c,a)*R(p,b,d)',
     dtens,ltens,rtens,
     scale=(5d-1,0d0),
     dev_id=0,dev_kind=DEV_NVIDIA_GPU)
```

TAL-SH C++ INTERFACE

```
talsh::Tensor dtens({74,156,74,156},std::complex<float>{0.0f,0.0f});
talsh::Tensor ltens({74,156,74,156},std::complex<float>{0.1f,0.2f});
talsh::Tensor rtens({74,156,74,156},std::complex<float>{0.3f,0.4f});
int ierr = dtens.contractAccumulateXL(
           nullptr,
           std::string("D(i,a,j,b)+=L(j,a,k,c)*R(k,b,i,c)"),
           ltens, rtens,
           DEV_NVIDIA_GPU, DEV_DEFAULT,
           std::complex<float>{0.5f,0.0f});
bool done = dtens.sync();
```

https://github.com/DmitryLyakh/TAL_SH.git

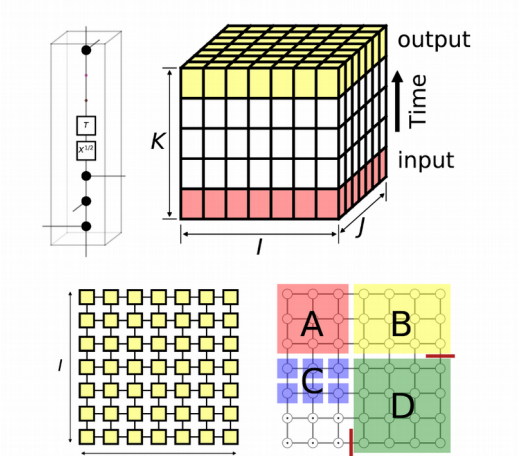
TAL-SH in Quantum Circuit Simulations

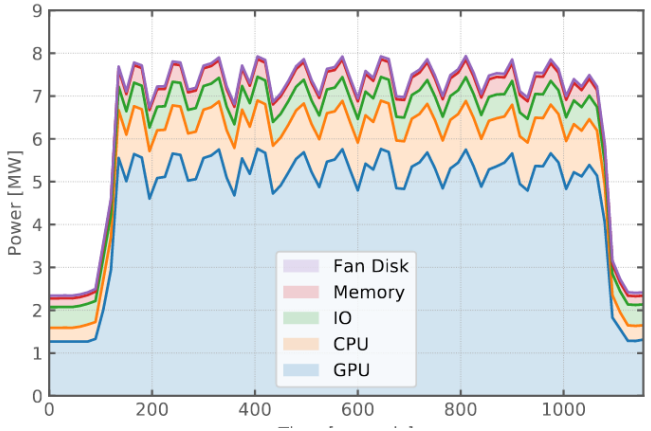
Arxiv 1905.00444 (2019)

Establishing the Quantum Supremacy Frontier with a 281 Pflop/s Simulation

Benjamin Villalonga[1,2,3,*], Dmitry Lyakh[4,5†], Sergio Boixo[6,‡], Hartmut Neven[6,+], Travis S. Humble[4,§], Rupak Biswas[1,×], Eleanor G. Rieffel[1,¶], Alan Ho[6,‖], and Salvatore Mandrà[1,7,⊢]

[1] Quantum Artificial Intelligence Lab. (QuAIL), NASA Ames Research Center, Moffett Field, CA 94035, USA
[2] USRA Research Institute for Advanced Computer Science (RIACS), 615 National, Mountain View, California 94043, USA
[3] Institute for Condensed Matter Theory and Department of Physics, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[4] Quantum Computing Institute, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[5] Scientific Computing, Oak Ridge Leadership Computing, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[6] Google Inc., Venice, CA 90291, USA
[7] Stinger Ghaffarian Technologies Inc., 7701 Greenbelt Rd., Suite 400, Greenbelt, MD 20770

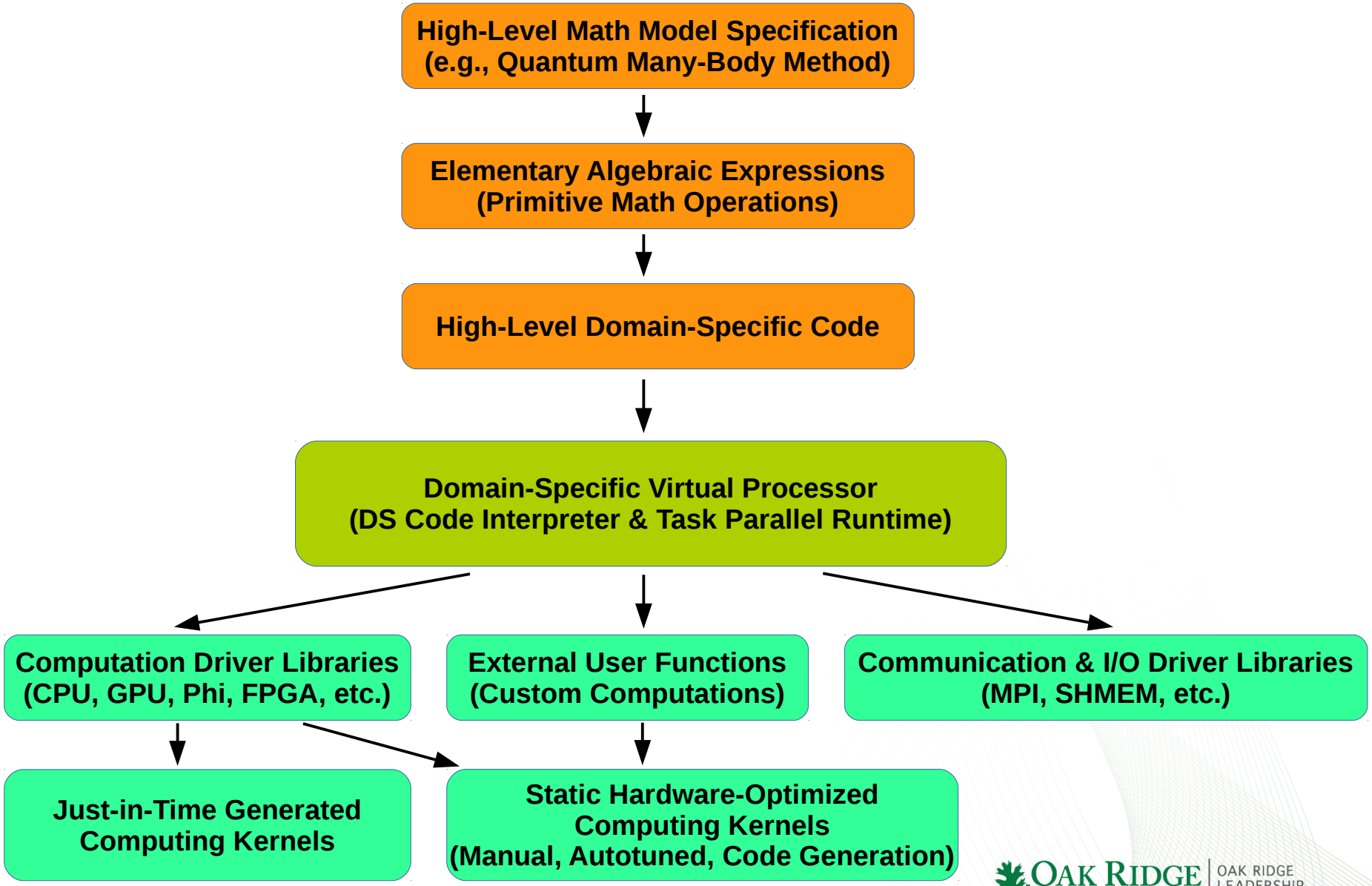




Circuit Size	Nodes Used	Runtime (h)	PFlop/s Peak	PFlop/s Sust.	Efficiency (%) Peak	Efficiency (%) Sust.	Power (MW)	Energy Cost (MWh)	PFlop/s/MW
$7 \times 7 \times (1 + 40 + 1)$	2300	4.84	191	142	92.0	68.5	-	-	-
$7 \times 7 \times (1 + 40 + 1)$	4600	2.44	381	281	92.1	68.0	8.075	21.1	34.8
$11 \times 11 \times (1 + 24 + 1)$	4550	0.278	368	261	89.8	63.7	7.3	2.32	35.8

https://github.com/DmitryLyakh/TAL_SH.git

Portable Scalable Scientific Computing



How to Build a DSVP

- **Domain-Specific Microcode**: Library-based implementation of domain-specific primitives, plus auxiliary operations: Manual or generated code
- **Resource Allocation Primitives**: Building blocks for hierarchical memory management: Target-agnostic
- **Data Transfer Primitives**: Building blocks for data transfer between devices in the same node as well as between nodes: Target-agnostic
- **Data Decomposition/Aggregation methods**: User-provided
- **Virtual Architecture Specification**: Composition of the domain-specific virtual processor in terms of domain-specific virtual units with well-defined functionality: Domain-provided