LAPENNA Program
LECTURE 2 - Introduction

Kwai Wong, Stan Tomov
Julian Halloy, Stephen Qiu, Eric Zhao

University of Tennessee, Knoxville

February 10, 2022

Acknowledgements:

- Support from NSF, UTK, JICS, ICL, NICS
- LAPENNA, www.jics.utk.edu/lapenna, NSF award #202409
- www.icl.utk.edu, cfdlab.utk.edu, www.xsede.org, www.jics.utk.edu/recsem-reu,
- MagmaDNN is a project grown out from the RECSEM REU Summer program supported under NSF award #1659502
- Source code: www.bitbucket.org/icl/magmadnn
- www.bitbucket.org/cfdl/opendnnwheel

INNOVATIVE COMPUTING LABORATORY

Overview of LAPENNA

The major goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem. This program aims to prepare college faculty, researchers, and industrial practitioners to design, enable and direct their own course curricula, collaborative projects, and training programs for in-house data-driven sciences programs. The LAPENNA program focuses on delivering computational techniques, numerical algorithms and libraries, and implementation of AI software on emergent CPU and GPU platforms.

Ecosystem Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Modeling, Numerical Linear Algebra, Data Analytics, Machine Learning, DNN, GPU, HPC

Session 1	Session 2	Session 3	Session 4
7/2020 - 12/2020	1/2021- 6/2021	7/2021 - 1/2022	1/2022 - 6/2022
16 participants	16 participants	16 participants	16 participants
Faculty/Students	Faculty/Students	Faculty/Students	Faculty/Students
10 webinars	10 webinars	10 webinars	10 webinars
4 Q & A	4 Q & A	4 Q & A	4 Q & A

Colleges courses, continuous integration, online courses, projects, software support

Web-based resources, tutorials, webinars, training, outreach

Teams and Participants of LAPENNA

- ✓ **PIs : Kwai Wong (JICS), Stan Tomov (ICL), University of Tennessee, Knoxville**
  - **Stephen Qiu, Julian Halloy, Eric Zhao (Students)**

- ✓ **Team : Clemson University**
- ✓ **Teams : University of Arkansas**
- ✓ **Team : University of Houston, Clear Lake**
- ✓ **Team : Miami University, Ohio**
- ✓ **Team : Boston University**
- ✓ **Team : West Virginia University**
- ✓ **Team : Louisiana State University, Alexandria**
- ✓ **Teams : Jackson Laboratory**
- ✓ **Team : Georgia State University**
- ✓ **Teams : University of Tennessee, Knoxville**
- ✓ **Teams : Morehouse College, Atlanta**
- ✓ **Team : North Carolina A & T University**
- ✓ **Team : Clark Atlanta University, Atlanta**
- ✓ **Team : Alabama A & M University**
- ✓ **Team : Slippery Rock University**
- ✓ **Team : University of Maryland, Baltimore County**

Meeting Time

- ✓ **Webinar Meeting time. Thursday 8:00 – 10:00 pm ET,**
- ✓ **Tentative schedule, www.jics.utk.edu/lapenna --> Spring 2022**

Topic: LAPENNA Spring 2022 Webinar
Time: Feb 3, 2022 08:00 PM Eastern Time (US and Canada)
Every week on Thu, 12 occurrence(s)
Feb 3, 2022 07:30 PM
Feb 10, 2022 07:30 PM
Feb 17, 2022 07:30 PM
Feb 24, 2022 07:30 PM
Mar 3, 2022 07:30 PM
Mar 10, 2022 07:30 PM
Mar 17, 2022 07:30 PM
Mar 24, 2022 07:30 PM
Mar 31, 2022 07:30 PM
Apr 7, 2022 07:30 PM
Apr 14, 2022 07:30 PM
Apr 21, 2022 07:30 PM

Join from PC, Mac, Linux, iOS or Android: https://tennessee.zoom.us/j/94140469394
Password: 708069

Schedule of LAPENNA Spring 2022
Thursday 8:00pm -10:00pm Eastern Time

The goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem for data-driven applications.

Month	Week	Date	Topics
February	Week 01	3	Logistics, High Performance Computing
	Week 02	10	Computational Ecosystem, Linear Algebra
	Week 03	17	Introduction to DNN, Forward Path (MLP)
	Week 04	24	Backward Path (MLP), Math, Example
March	Week 05	3	CNN Computation
	Week 06	10	CNN Backpropagation, Example
	Week 07	17	CNN Network, Linear Algebra
	Week 08	24	Segmentation, Unet,
	Week 09	31	Object Detection, RC vehicle
April	Week 10	7	RNN, LSTEM, Transformers
	Week 11	14	DNN Computing on GPU
	Week 12	21	Overview, Closing
June or July	Workshop	To be arranged	4 Days In Person at UTK

Topics of Webinar Two

- ✓ **Introduction to LAPENNA Program**
- ✓ **Big Science and Big Data Computational Ecosystem**
  - ➢ **Hardware System, Hardware, Performance**
  - ➢ **Computational Models - Mathematics, Solvers, Numerical Linear Algebra**
  - ➢ **Software Implementation - Computer Sciences, Implementations**
  - ➢ **Data Computing – Machin Learning Models, Frameworks**

- ✓ **Unit 2 : Numerical Linear Algebra**
  - ➢ **Linear Algebra Operations**
  - ➢ **Basic Linear Algebra Subprogram (BLAS)**
  - ➢ **MM code in Google Colab and more**
  - ➢ **Numerical Linear Algebra and Neural Network**
  - ➢ **Tensorflow 2.0, Tensor operations**

Modeling Cycle and Emergent Technology

Big Data → Deep Learning → Computing capacity
Combining the Learned the pattern and behavior hidden in sensor or computed data to predict complicated phenomena

A growing field with evolving technology
Statistical Machine Learning →
Deep Neural Network → Numerical Linear Algebra Supercomputing (HPC) → Real Time Applications

Data- + model-based simulation --> Big Iron (supercomputer)

Fast enough to get approximate results (time scale)
Enough memory to process the huge amount of data (length scale)

Computational Ecosystem

Computational Ecosystem

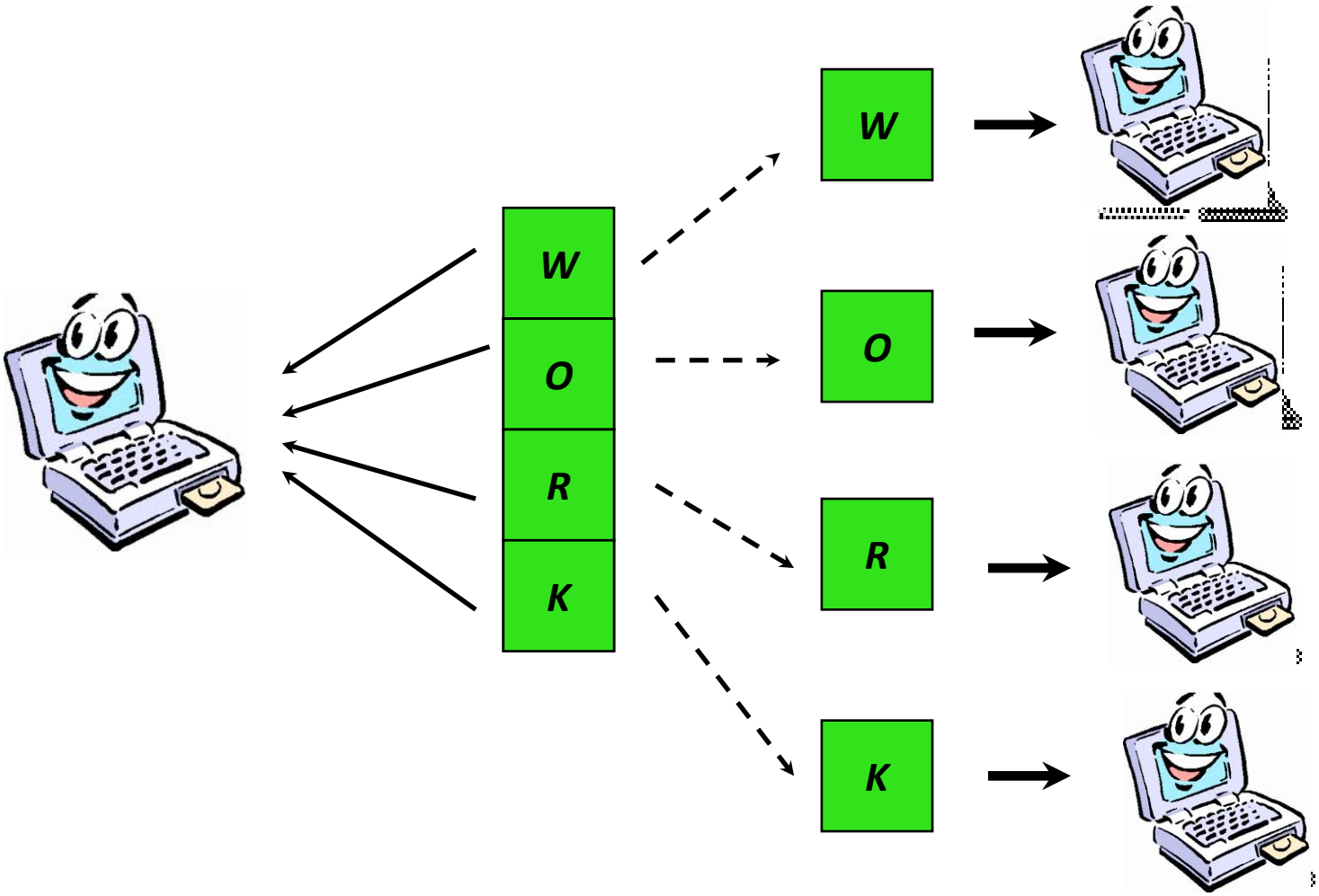
- **Advance Computing system**
  - **Top500, FLOPS and Performance**
  - **DOE Exascale Road Map**
  - **NSF infrastructure**
  - **Desktop, Accelerators, GPUs**
  - **Google COLAB**
- **Predictive Simulation Science**
  - **Equation Based Simulation Sciences,**
  - **Mathematics Essentials, Calculus**
  - **Solvers, Linear Algebra, BLAS**
- **Computer Sciences and Software tools**
  - **Linux OS**
  - **Computational thinking**
  - **Workflow**
  - **Languages**
- **Data Intensive Sciences**
  - **Statistical Learning**
  - **Data mining**
  - **Deep Learning**
  - **Framework**

Simple Story of Parallel Computing

Division of work into smaller tasks

Multiple computers work on smaller tasks simultaneously

>> Reduce Wall Clock Time <<

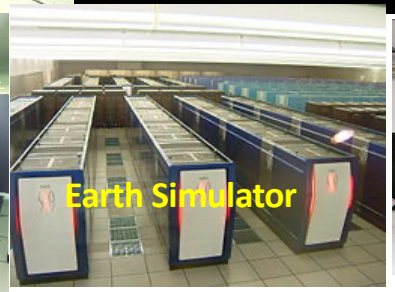


Simple Parallel Computer

Many commodity units connected by a COS interconnect

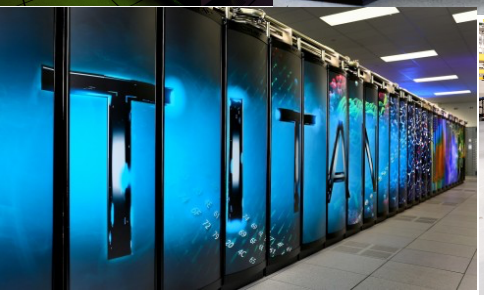
LANDSCAPE OF SUPERCOMPUERS

70s-80s
Long vector
Few processor units
Shared memory
Fortran



90s
Medium size vector, AP
Many processor units, SM,
Distributed memory, MPI,
Fortran, C, openMP, MPI

90-2000s
Small single processor, PC chip (RISC)
Lots more processor units
Distributed memory,
Fortran, C, C++, openMP, MPI



2010s-2020s
Powerful multicore processor,
Accelerators, GPU,
Lots more processor units
Distributed memory,
Fortran, C, C++, openMP, MPI, cuda,

EXASCALE SUPERCOMPUERS

Essential of Parallel Computing

INNOVATIVE COMPUTING LABORATORY

- ✓ Single instruction, single data, old time serial
- ✓ Single instruction, pipeline of data, vectorization
- ✓ Multiple (single instruction. Pipeline data)
- ✓ Single instruction, block of data, threads (CPU, multicore)
- ✓ Multiple instruction, multiple block of data (accelerator, GPU)
- ✓ Single CPU (multicore, Multiple GPU, (shared memory node)
- ✓ Multiple nodes (blade)
- ✓ Multiple blade (rack) ------ -> distributed computer

Increasing number of instruction

Parallel Execution

Increasing number of data access per instruction

Memory Access

Exascale computer

Units of Measure for HPC

- **High Performance Computing (HPC) units are:**
  - **Flop: floating point operation, usually double precision unless noted**
  - **Flop/s: floating point operations per second**
  - **Bytes: size of data (a double precision floating point number is 8 bytes)**

- **Typical sizes are millions, billions, trillions…**

Kilo	**Kflop/s = 10^3 flop/sec**	**Kbyte = 10^3 ~ 2^{10} = 1024 bytes (KiB)**
Mega	**Mflop/s = 10^6 flop/sec**	**Mbyte = 10^6 ~ 2^{20} bytes (MiB)**
Giga	**Gflop/s = 10^9 flop/sec**	**Gbyte = 10^9 ~ 2^{30} bytes (GiB)**
Tera	**Tflop/s = 10^{12} flop/sec**	**Tbyte = 10^{12} ~ 2^{40} bytes (TiB)**
Peta	**Pflop/s = 10^{15} flop/sec**	**Pbyte = 10^{15} ~ 2^{50} bytes (PiB)**
Exa	**Eflop/s = 10^{18} flop/sec**	**Ebyte = 10^{18} ~ 2^{60} bytes (EiB)**
Zetta	**Zflop/s = 10^{21} flop/sec**	**Zbyte = 10^{21} ~ 2^{70} bytes (ZiB)**
Yotta	**Yflop/s = 10^{24} flop/sec**	**Ybyte = 10^{24} ~ 2^{80} bytes (YiB)**

Goal (points to Exa)

Goal (points to Pbyte)

- **Current fastest (public) machines are petaflop systems**
  - **Up-to-date list at www.top500.org**

Big Science, Big Data, Big Iron

www.top500.org

- 500 most powerful computers in the world
- Updated twice a year:
  - ISC'xy in June in Germany
  - SCxy in November in the U.S.

Yardstick: Floating Point Operations per Second (FLOP/s) Rmax of Linpack

- Solve Ax=b, Matrix A is dense with random entries
- Dominated by dense matrix-matrix multiply

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096

2021

Fugaku (#1 Worldwide) System Overview

RIKEN Center for Computational Science (R-CCS)

System Performance

- Peak performance of 442 petaflops (per TOP500 Rmax),
- 2.0 EFLOPS on a different mixed-precision benchmark

Each node has

- Fujitsu A64FX CPU (48+4 cores) per node
- HBM2 32 GiB

The system includes

- 158,976 nodes
- Custom Tofu Interconnect D
- 1.6 TB NVMe SSD/16 nodes (L1)
- 150 PB Lustre Filesystem (L2)
- Cloud storage (L3)

TOP500® SUPERCOMPUTER SITES

Summit (#1 in US) System Overview

OAK RIDGE National Laboratory

summit

System Performance

- Peak performance of 200 petaflops for modeling & simulation
- Peak of 3.3 ExaOps for data analytics and artificial intelligence

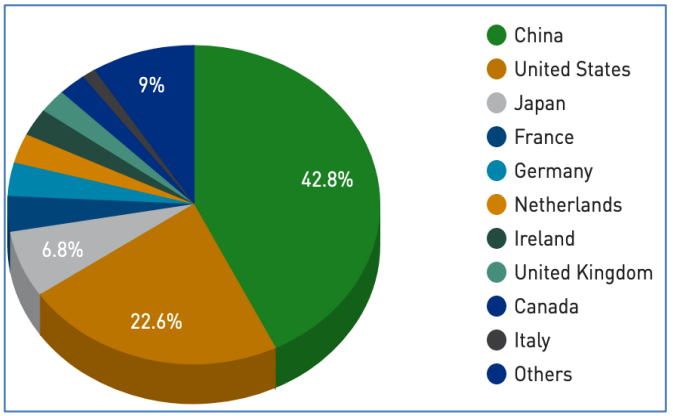
Each node has

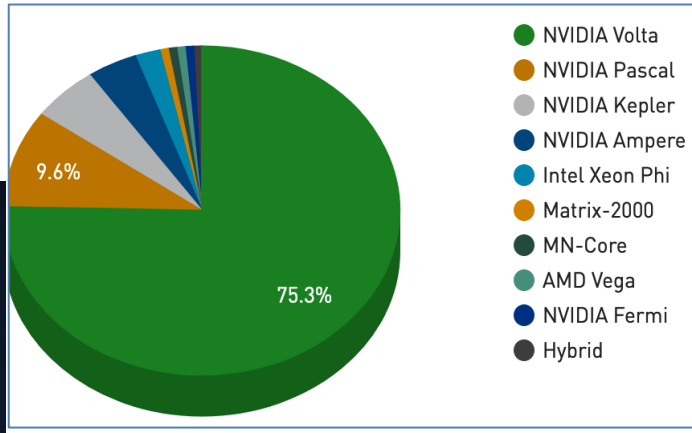
- 2 IBM POWER9 processors
- 6 NVIDIA Tesla V100 GPUs
- 608 GB of fast memory
- 1.6 TB of NVMe memory

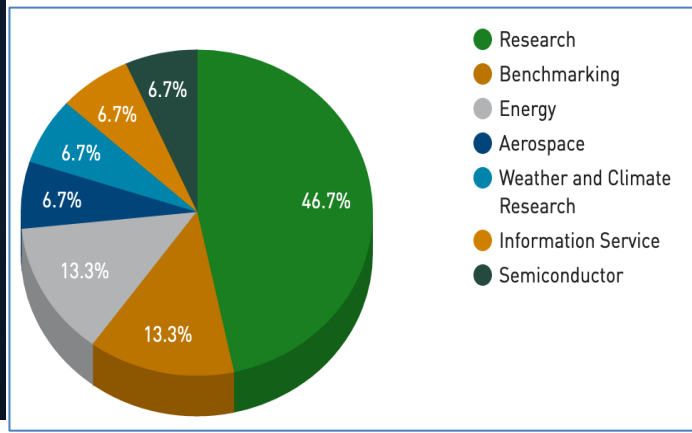
The system includes

- 4608 nodes
- Dual-rail Mellanox EDR InfiniBand network
- 250 PB IBM Spectrum Scale file system transferring data at 2.5 TB/s

Advanced Computing Systems

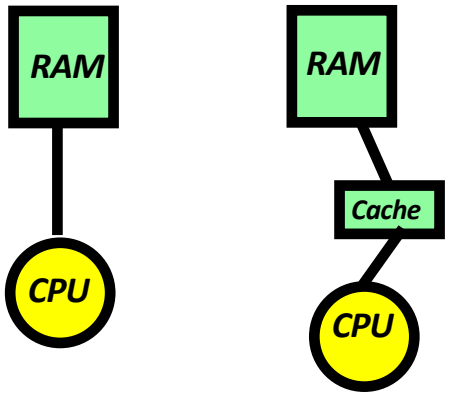


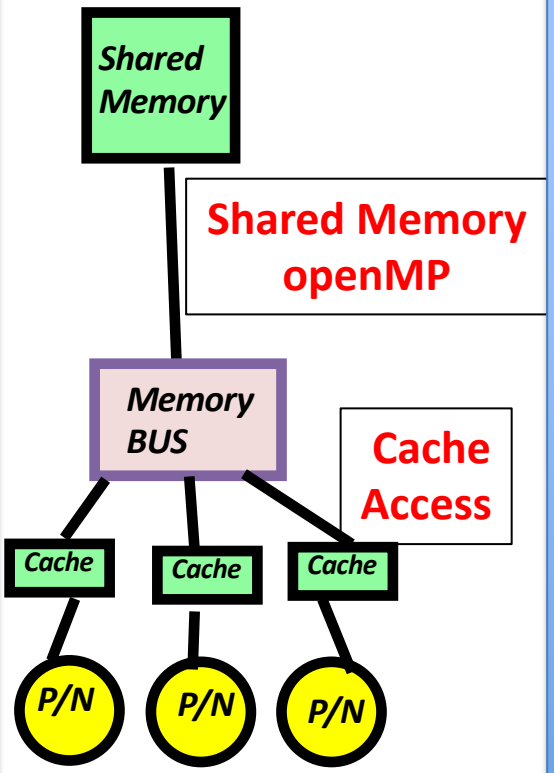


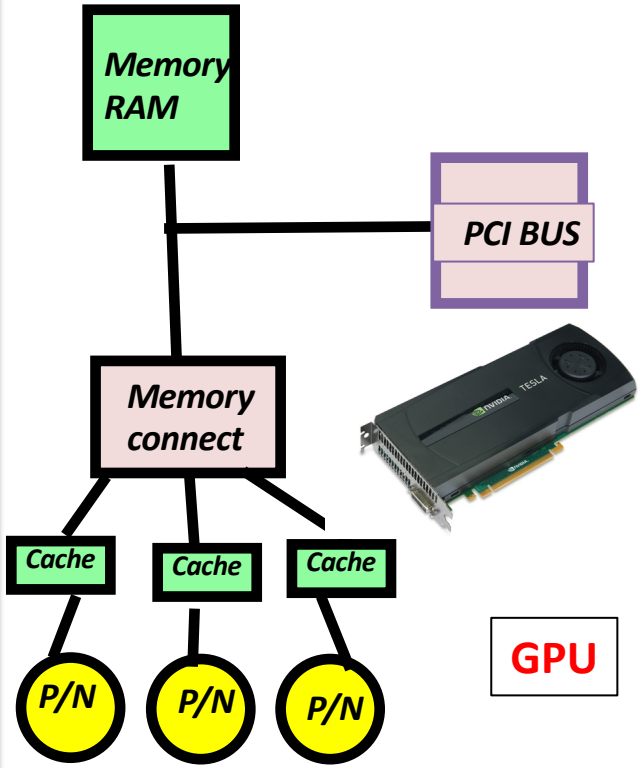


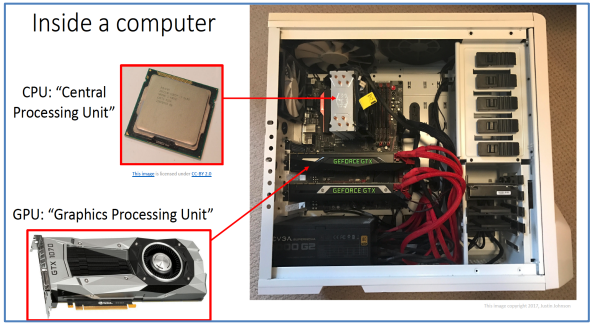
Simple story of Computers

Major Bottleneck - Communication, Memory Access





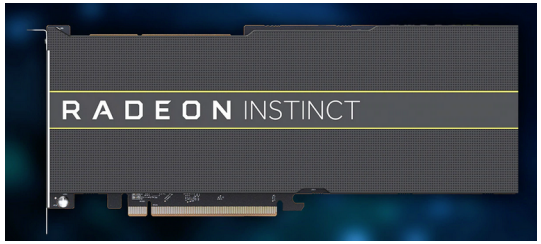


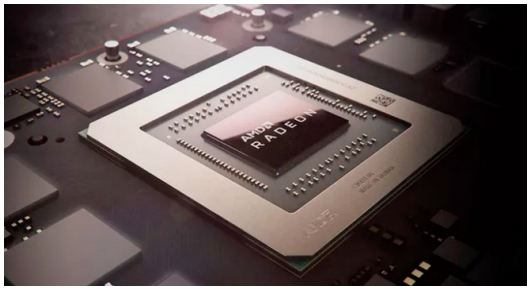


GPU Processors in supercomputers

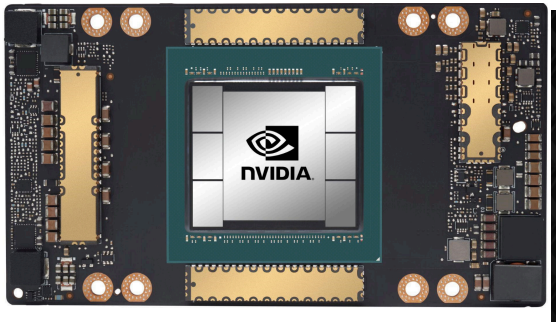
Joint Institute for Computational Sciences

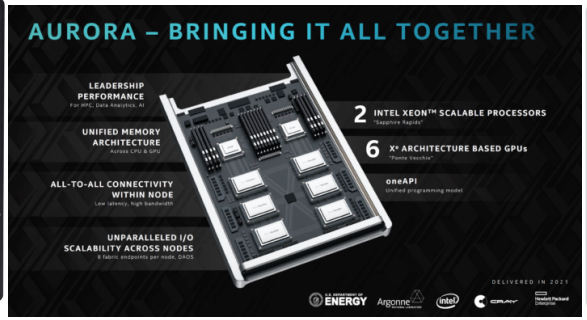
INNOVATIVE COMPUTING LABORATORY

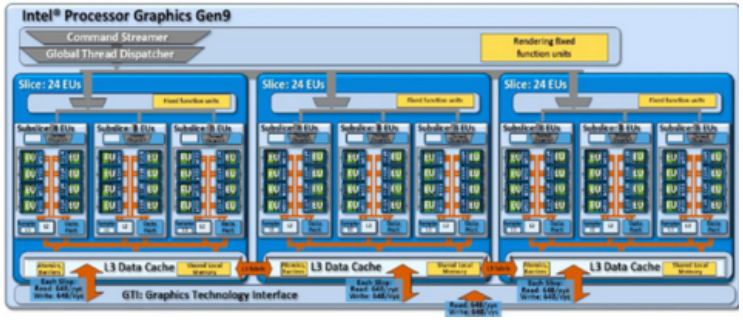




Based on "Vega 7nm" Technology with 60 supercharged Compute Units (3840 Stream Processors)
Up to 26.5 TFLOPS FP16 and 13.3 TFLOPS FP32
Performance 6.6 TFLOPS Double Precision



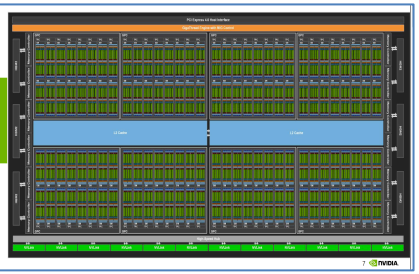




54B transistors
826 mm²

108 SM
6912 CUDA Cores
432 Tensor Cores

40 GB HBM2
1555 GB/s HBM2
600 GB/s NVLink



Intel Xe HPC 'Ponte Vecchio' GPU & Xeon Sapphire Rapids CPU Powered Aurora Exascale Supercomputer Further Detailed – Deploys in 2021 (Gen 12, 7nm??)

Peak FP64[1]	9.7 TFLOPS
Peak FP64 Tensor Core[1]	19.5 TFLOPS
Peak FP32[1]	19.5 TFLOPS
Peak FP16[1]	78 TFLOPS
Peak BF16[1]	39 TFLOPS
Peak TF32 Tensor Core[1]	156 TFLOPS | 312 TFLOPS[2]
Peak FP16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak BF16 Tensor Core[1]	312 TFLOPS | 624 TFLOPS[2]
Peak INT8 Tensor Core[1]	624 TOPS | 1,248 TOPS[2]
Peak INT4 Tensor Core[1]	1,248 TOPS | 2,496 TOPS[2]

Nvidia/CUDA Terminology	AMD Terminology	Intel Terminology	Description (pulled almost word-for-word from AMD slides)
Streaming Multiprocessor (SM)	Compute Unit (CU)	SubSlice (SS)	One of many independent parallel vector processors in a GPU that contain multiple SIMD ALUs.
Kernel	Kernel	Kernel	Functions launched to the GPU that are executed by multiple parallel workers on the GPU.
Warp	Wavefront	Vector thread, issuing SIMD instruction	Collection of operations that execute in lockstep, run the same instructions, and follow the same control-flow path. Individual lanes can be masked off.
Thread block	Workgroup	Workgroup	Group of warps/wavefronts/vector threads that are on the GPU at the same time. Can synchronize together and communicate through local memory.
Thread	Work item/Thread	Work item/Vector lane	Individual lane in a warp/wavefront/vector thread
Global Memory	Global Memory	GPU Memory	DRAM memory accessible by the GPU that goes through some layers of cache
Shared memory	Local memory	Shared local memory	Scratchpad that allows communication between warps/wavefronts/vector threads in a threadblock/workgroup
Local memory	Private memory	GPRF	Per-thread private memory, often mapped to registers.

GPU capability and performance

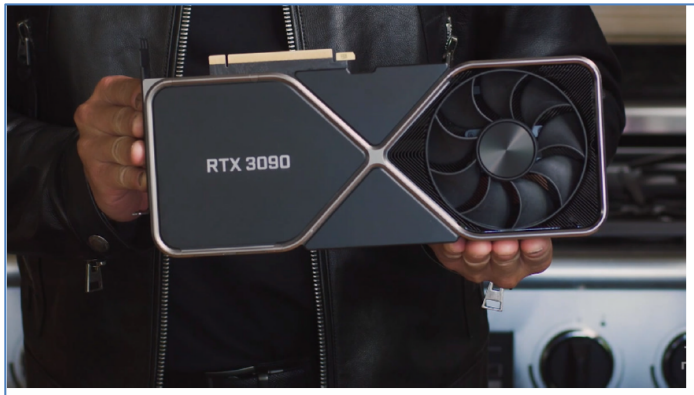
- Two important features that describe GPU capability:
  - Number of CUDA cores
  - Memory size
- Two corresponding metrics for describing GPU performance:
  - Peak computational performance: how many single-precision or double-precision floating point calculations can be processed per second (e.g. trillion floating-point operations per second, tflops)
  - Memory bandwidth: the rate at which data can be read from or stored to memory (e.g. gigabytes per second, GB/s)

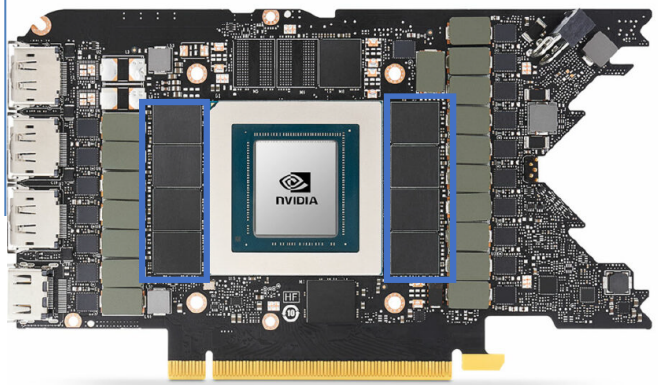
Product	#of Cuda Cores	DP TFLOPS	SP (TFLOPS)	Memory Size(GB)	memory Bandwidth(GB/s)
Jetson nano	128	n/a	0.472	4	25.6
GTX1650	892	n/a	3.00	4	8.0
K80	4992	2.91	8.74	24	480.0
Volta 100	5120	7.00	14.00	32	900.0

Inside a computer

CPU: "Central Processing Unit"

GPU: "Graphics Processing Unit"





CPU vs GPU

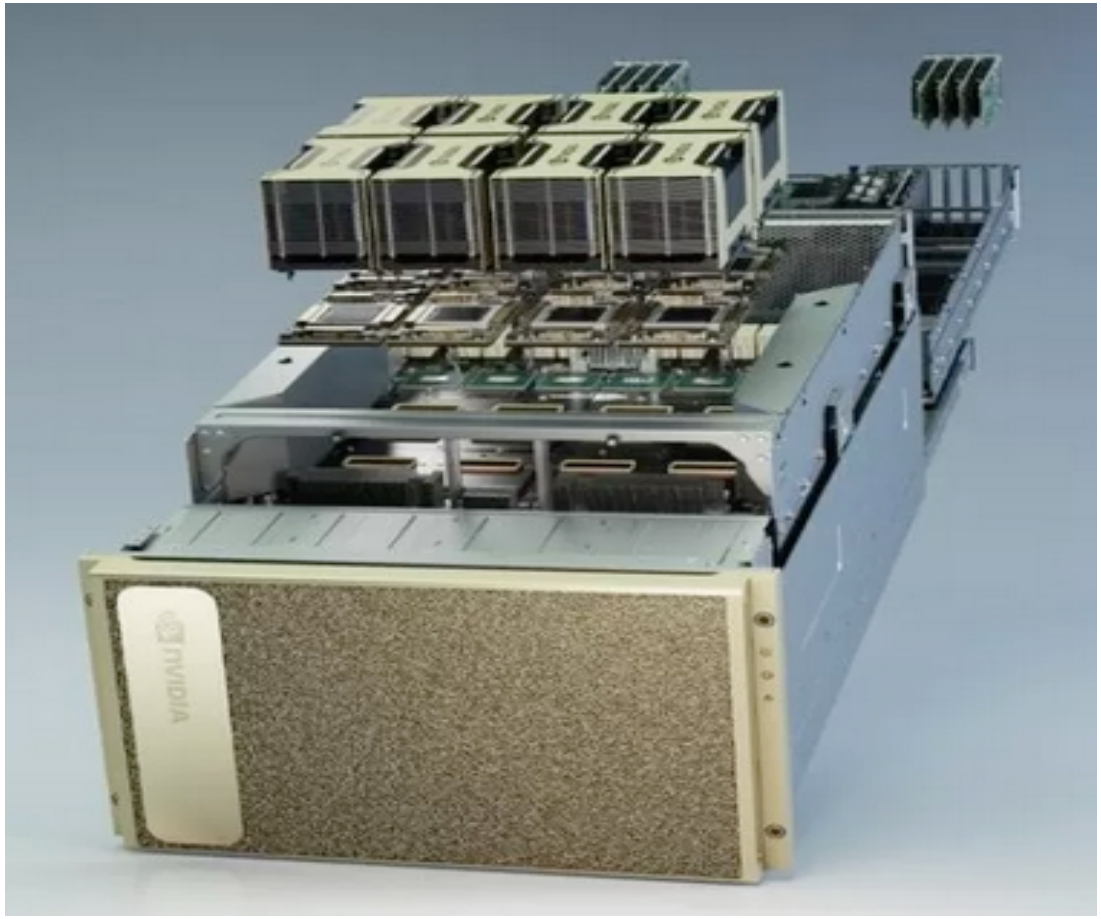
	Cores	Clock Speed (GHz)	Memory	Price	TFLOP/sec
CPU Ryzen Threadripper 3970X	64 (128 threads with hyperthreading)	3.7 (4.5 boost)	System RAM	$1999	~6.9 FP32
GPU NVIDIA RTX 3090	10496	1.4 (1.7 boost)	24 GB GDDR6X	$1499	~35.6 FP32 ~142 TFLOP with Tensor core

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html

DGX A100 Box



SYSTEM SPECIFICATIONS	
GPUs	8x NVIDIA A100 Tensor Core GPUs
GPU Memory	320 GB total
Performance	5 petaFLOPS AI 10 petaOPS INT8
NVIDIA NVSwitches	6
System Power Usage	6.5kW max
CPU	Dual AMD Rome 7742, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost)
System Memory	1TB
Networking	8x Single-Port Mellanox ConnectX-6 VPI 200Gb/s HDR InfiniBand 1x Dual-Port Mellanox ConnectX-6 VPI 10/25/50/100/200Gb/s Ethernet
Storage	OS: 2x 1.92TB M.2 NVME drives Internal Storage: 15TB (4x 3.84TB) U.2 NVME drives
Software	Ubuntu Linux OS
System Weight	271 lbs (123 kgs)
Packaged System Weight	315 lbs (143kgs)
System Dimensions	Height: 10.4 in (264.0 mm) Width: 19.0 in (482.3 mm) MAX Length: 35.3 in (897.1 mm) MAX
Operating Temperature Range	5°C to 30°C (41°F to 86°F)

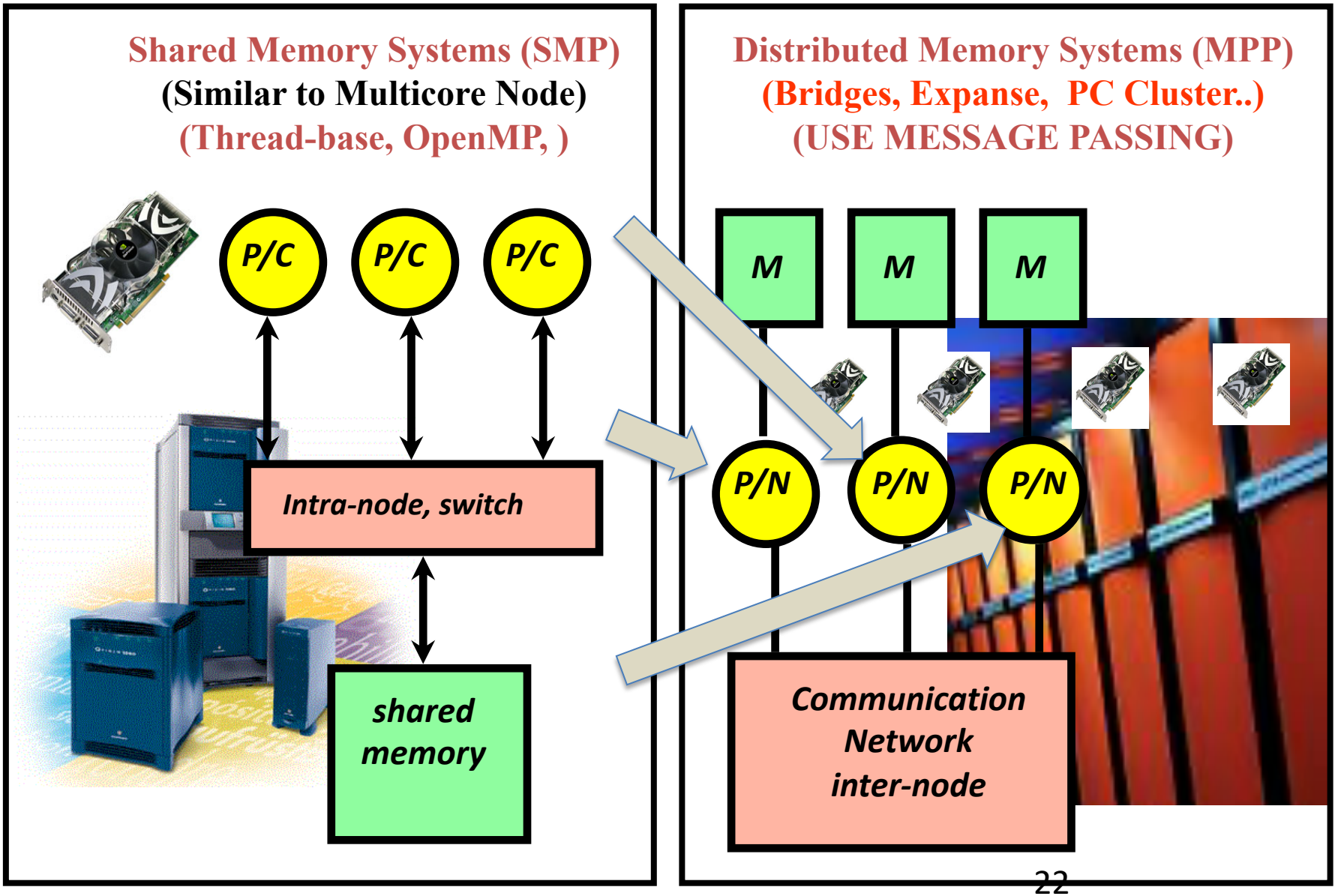
Single Box = DP 9.7x8=77.6 TF (155.3 TF TC),
TF32 TC: 312x8=2.5PFLOPS; FP16 TC : 2.5 PF x2 – 5 PF AI

Cost : assume $100,000 x 6000 = 600 M, Frontier @ORNL = 600M
Aggregate performance DP 9.7 x 8 x 6000 ~= 465 PF, (931PF TC)
AI Performance : FP16 TC : 5 PF x 8 x 6000 = 240 ExaFLOPS !!!!

GPU : 32bit ML : Best performance / cost

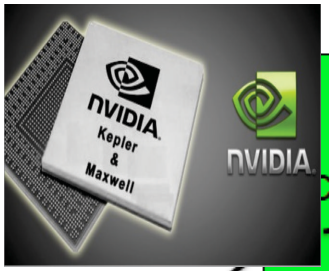
Simple Story of High Performance Parallel Computers

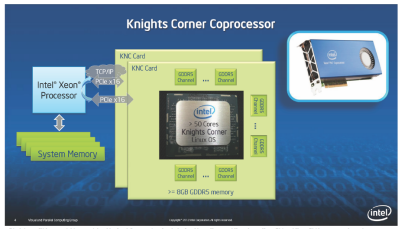
Computing in Parallel, Time Scale



Example of typical parallel machine

Shared memory programming between processes on a board and a combination of shared memory and distributed memory programming between nodes and cabinets





Source: George Chrysos, Hot Chips, August 28, 2012

Cabinet

Node/Board ... Node/Board Node/Board

Chip/Socket GPU ... Chip/Socket GPU Chip/Socket GPU

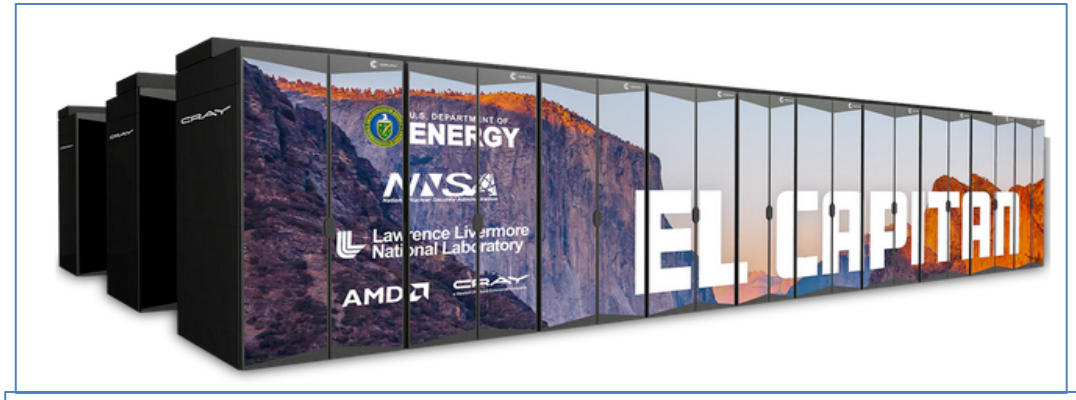
Core Core Core Core Core

From ICL Dr. Jack Dongarra : icl.cs.utk.edu

DOE Exascale Systems

Frontier

- ✓ **AMD in collaboration with the U.S. Department of Energy, Oak Ridge National Laboratory, and Cray Inc., designed the Frontier supercomputer expected to deliver more than 1.5 exaflops of peak processing power.**
- ✓ **Custom AMD EPYC™ CPUs optimized for HPC and AI**
- ✓ **Purpose-built HBM-enabled Radeon Instinct™ GPUs**
- ✓ **~$500+ millions , coming soon!!**



El Capitan

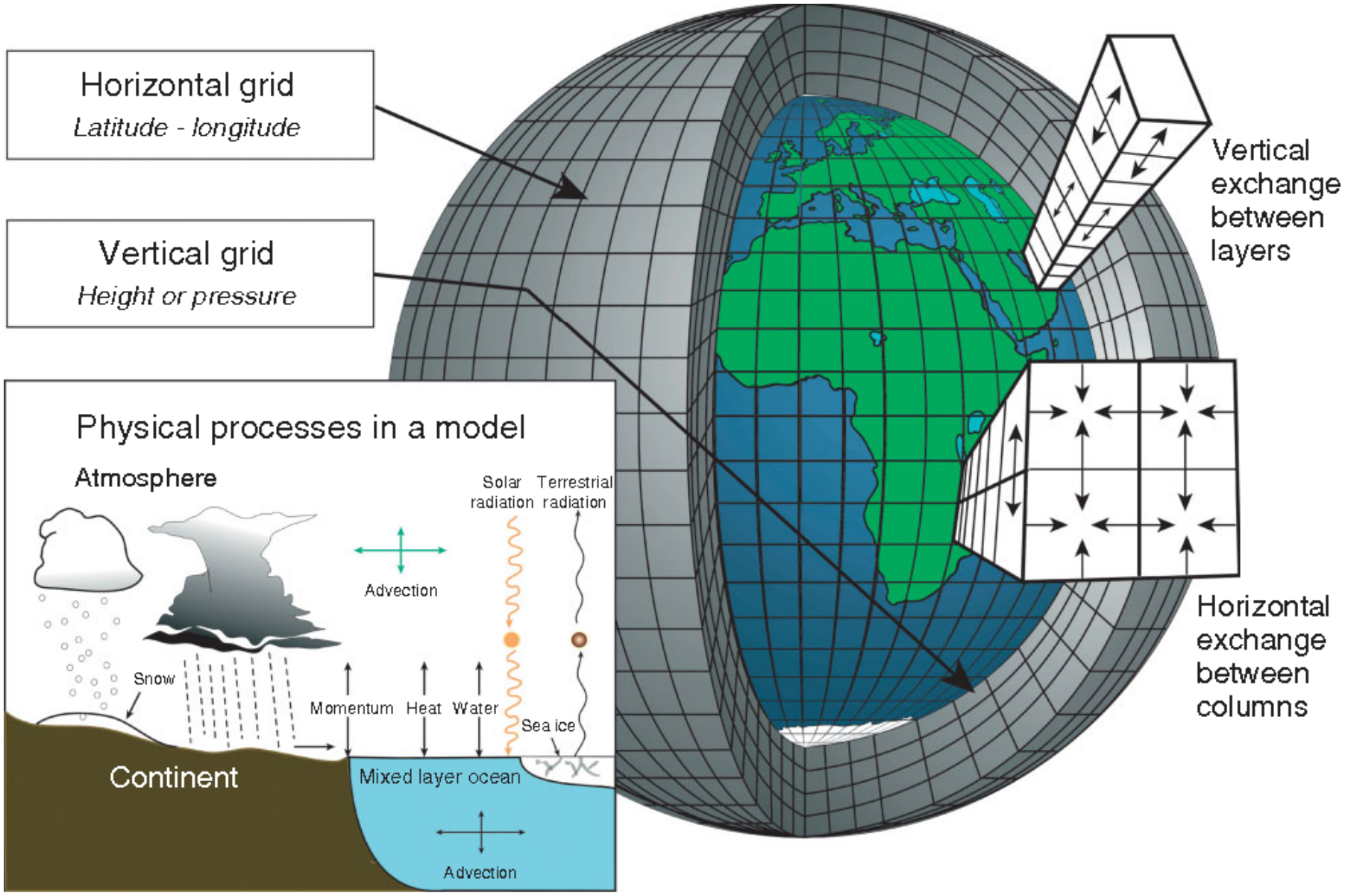
- ✓ **To enable greater than 2 exaflops of double precision processing power, the U.S. Department of Energy, Lawrence Livermore National Laboratory, and HPE have teamed up with AMD to design El Capitan, expected to be the world's fastest supercomputer with delivery anticipated in early 2023**
- ✓ **Next generation AMD EPYC processors, codenamed "Genoa", will feature the "Zen 4" processor core to support next generation memory and I/O sub systems for AI and HPC workloads**
- ✓ **Next generation Radeon Instinct GPUs based on new compute-optimized architecture for HPC and AI workloads will use next generation high bandwidth memory for optimum deep learning performance**
- ✓ **This design will excel at AI and machine-learning data analysis to create models that are faster, more accurate, and capable of quantifying the uncertainty of their predictions.**

Computational Ecosystem

- **Advance Computing system**
  - **Top500, FLOPS and Performance**
  - **DOE Exascale Road Map**
  - **NSF infrastructure**
  - **Desktop, Accelerators, GPUs**
  - **Google COLAB**
- **Predictive Simulation Science**
  - **Equation Based Simulation Sciences,**
  - **Mathematics Essentials, Calculus**
  - **Solvers, Linear Algebra, BLAS**

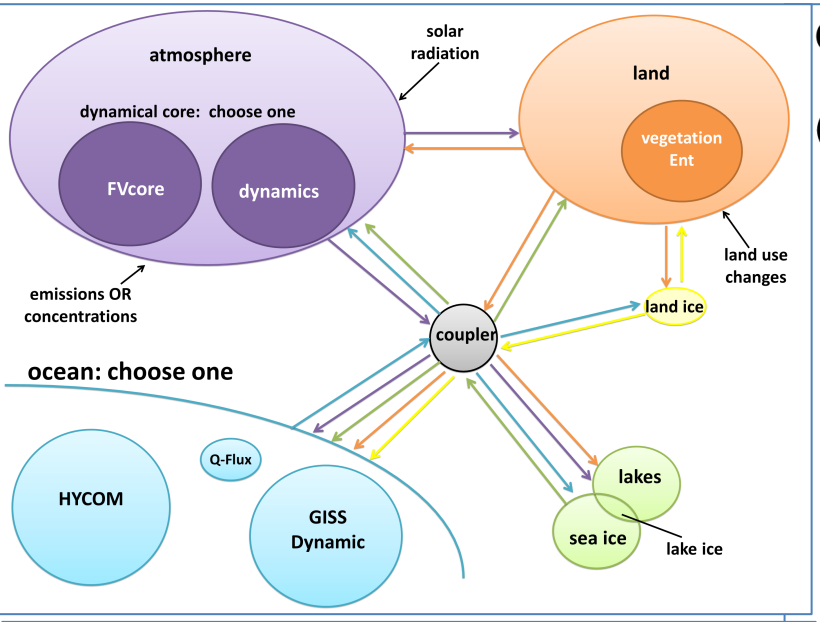
- **Computer Sciences and Software tools**
  - **Linux OS**
  - **Computational thinking**
  - **Workflow**
  - **Languages**
- **Data Intensive Sciences**
  - **Statistical Learning**
  - **Data mining**
  - **Deep Learning**
  - **Framework**

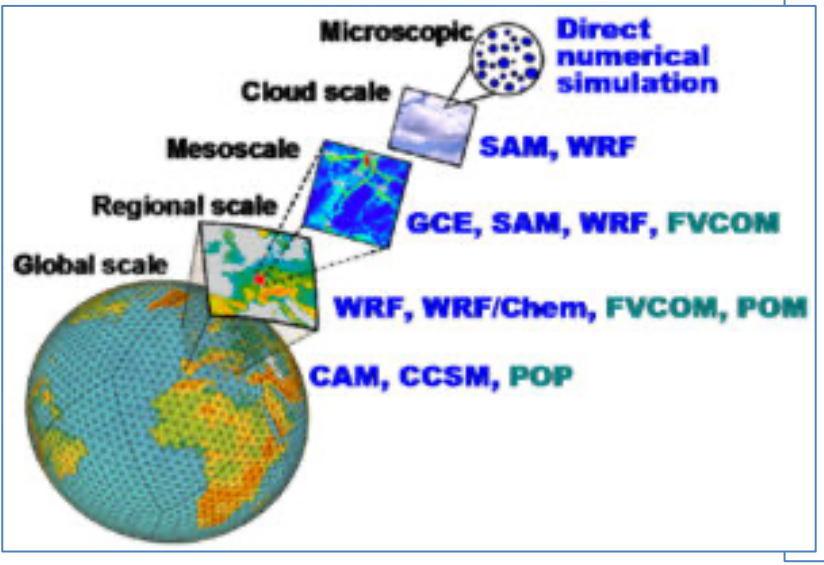
Big Science : Climate Modeling, Numerical Weather Forecast

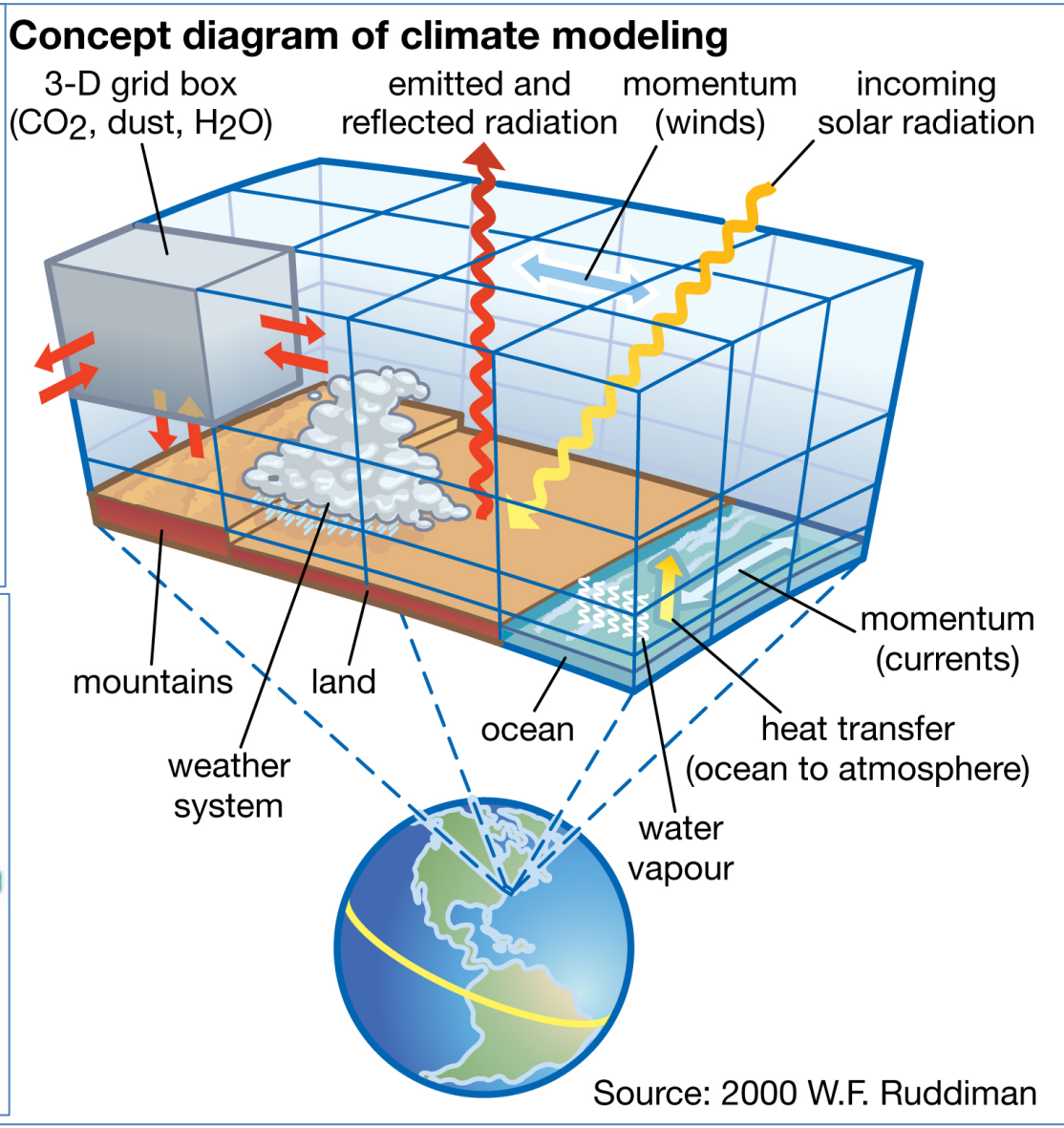


Diameter of Earth = ~12700km ; circumference of earth = ~40000km, surface area= 500,000,000 sq km

Climate Modeling : Complexity, Scales, Real World Application







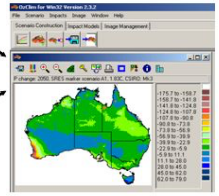
Earth surface = 500,000,000 sq km, surface grid = 100kmx100km (one patch for Hong Kong, 950 patches for CONUS), 10 vertical layers, total number of grid box = 500,000 boxes, 20 variables per grid box, => 10,000,000 DOF

Model hierarchy

Complex → Simple

Global climate model
(grid: 180 km by 180 km)

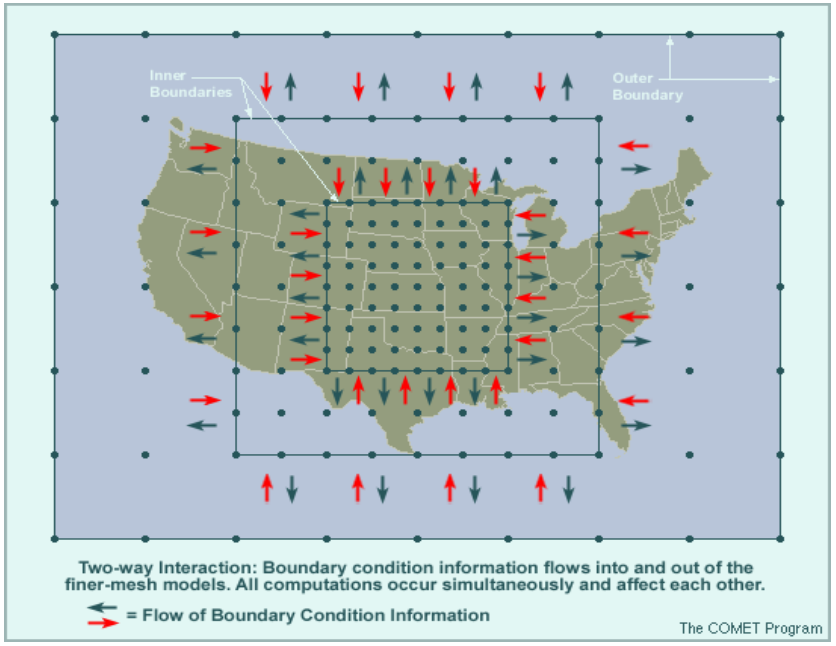
PC software, e.g. MAGICC, OzClim



Regional climate model
(grid: e.g. 70 km by 70 km)

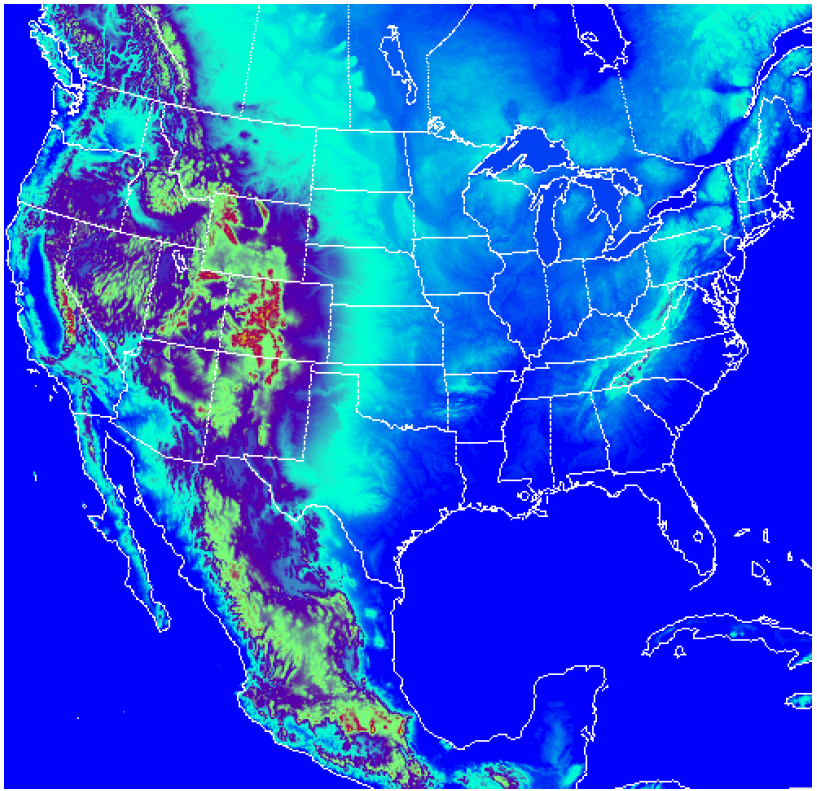
Regional climate model
(grid: e.g. 14 km by 14 km)

Statistical downscaling
(local sites: e.g. Perth)



WRF v4, CONUS Suite
1500x1500x50, 3 km, => 675 0000 grid box

- 50 vertical levels
- No cumulus
- Hybrid vertical activated
- Moist theta
- 18 s dt
- 6 minute simulation
- 10 hour spin-up, then restart
- No I/O included in timing
- Single radiation time step
- 18 non-radiation time steps



Science Driven Computing
Time (Speed) and Length (Memory) Scales

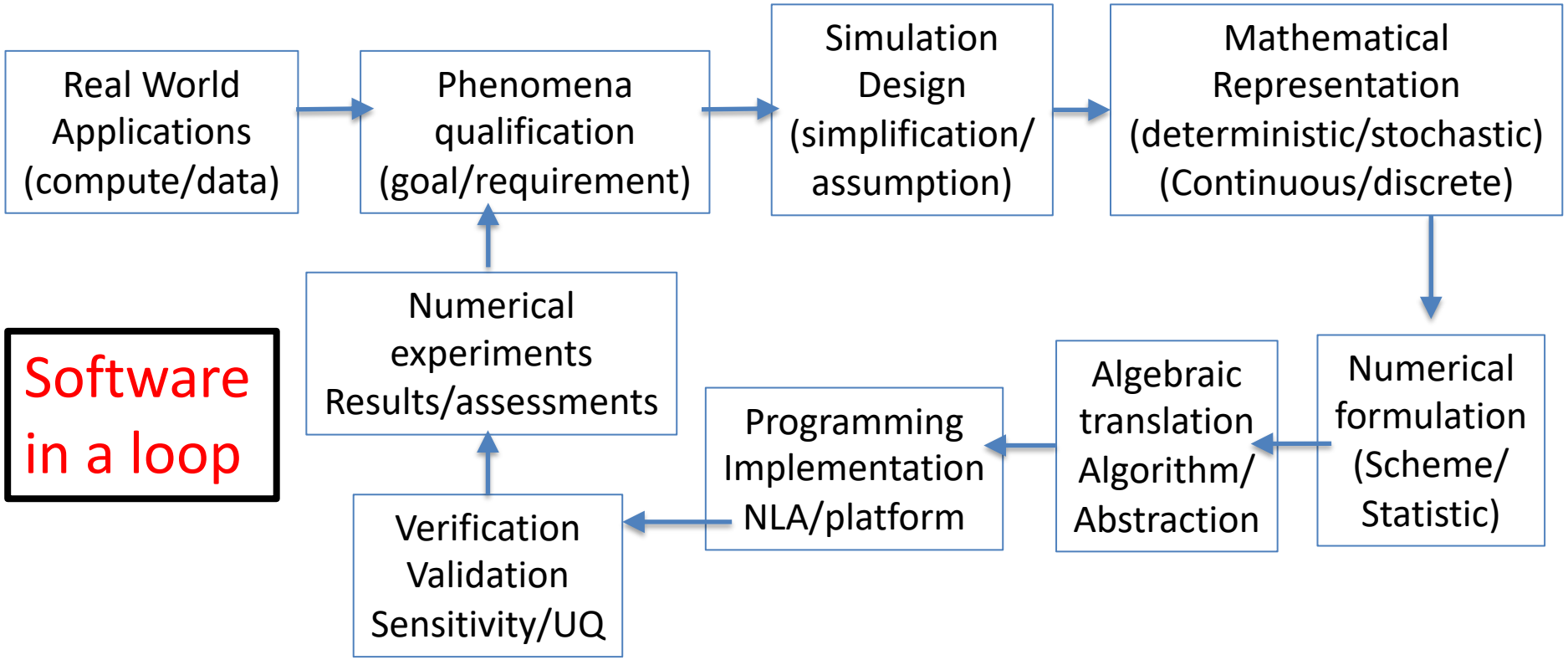
Large Scale Complex model → Computing capacity
Combining physics models and data from sensors to predict complicated phenomena

Fast enough to get acceptable results (time scale)
Enough memory to process the huge amount of data (length scale)

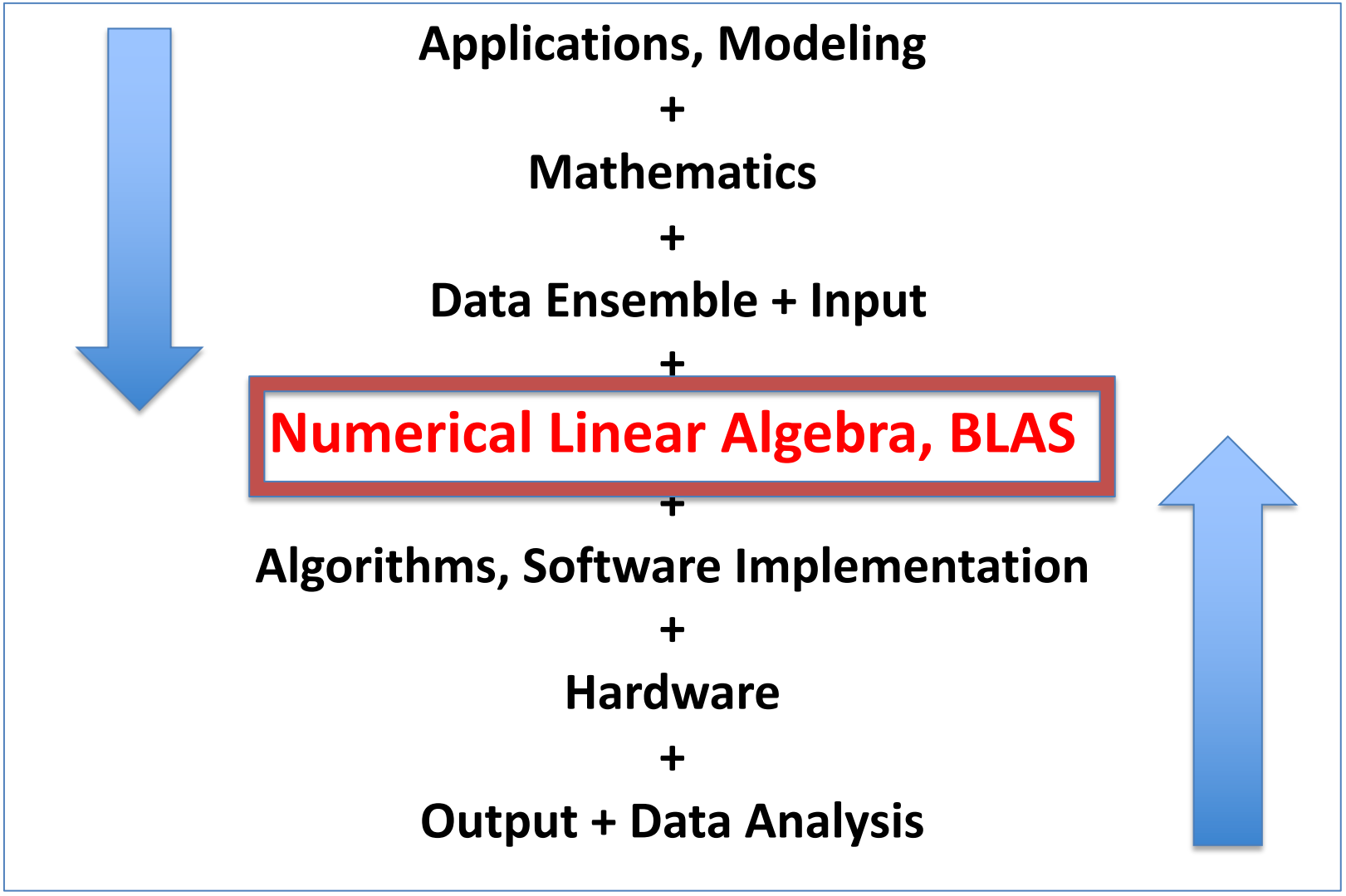
An established field with evolving technology
Parallel + model-based simulation --> Big Iron (supercomputer, clusters of computers)

Real World Applications Modeling

- ✓ **Goals and Objectives**
- ✓ **Cost, Resources and time**
- ✓ **Top down, bottom up, project design, agile, tools. UML, data management,**
- ✓ **Small test to large scale, Abstraction, Object oriented**
- ✓ **Length scale and time scale of model**
- ✓ **Open source , home grown, libraries**



Workflow of Modeling



JICS Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

Linear Algebra

David Cherney, Tom Denton,
Rohit Thomas and Andrew Waldron

Introduction to Linear Algebra

Mark Goldman

Emily Mackevicius

Lecture slides for

Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares

Stephen Boyd Lieven Vandenberghe

Caltech Division of the Humanities and Social Sciences

Quick Review of Matrix and Real Linear Algebra

KC Border
Subject to constant revision
Last major revision: December 12, 2016
v. 2019.10.23::14.52

Linear Algebra Review and Reference

Zico Kolter (updated by Chuong Do)

September 30, 2015

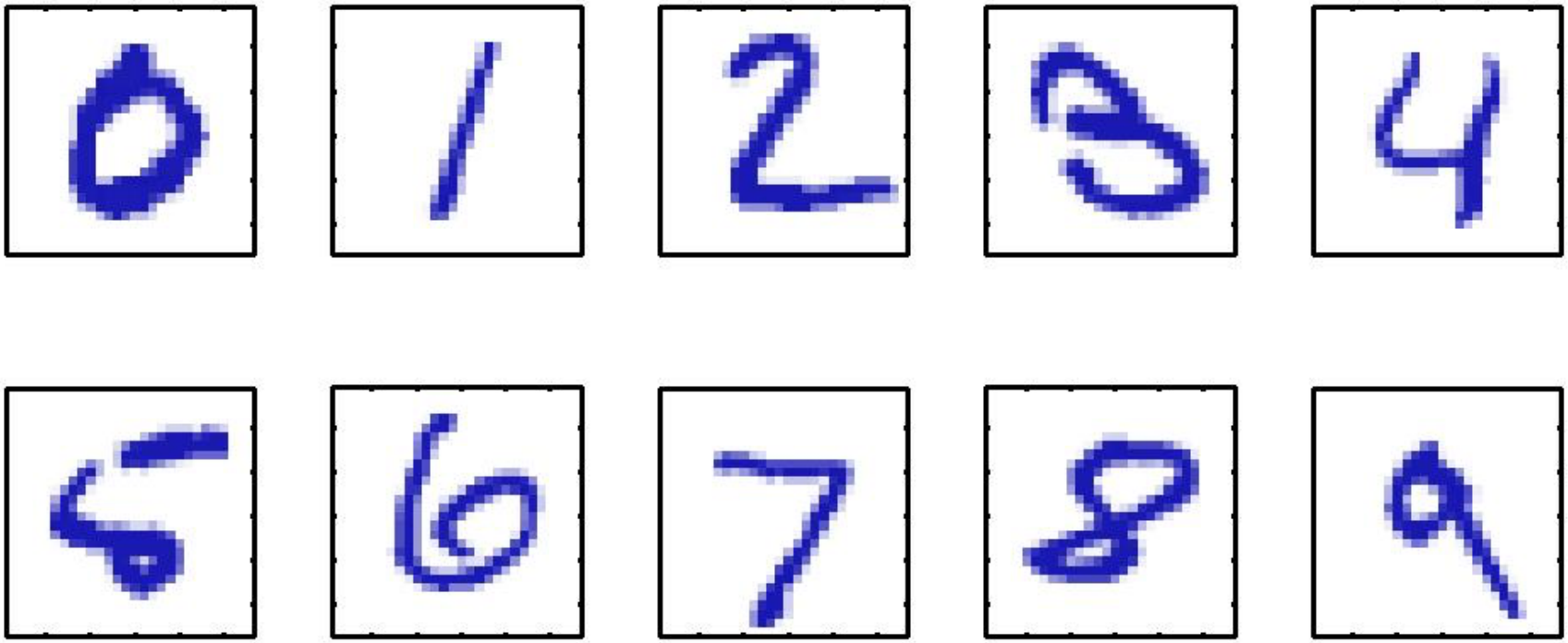
List of references: Linear Algebra

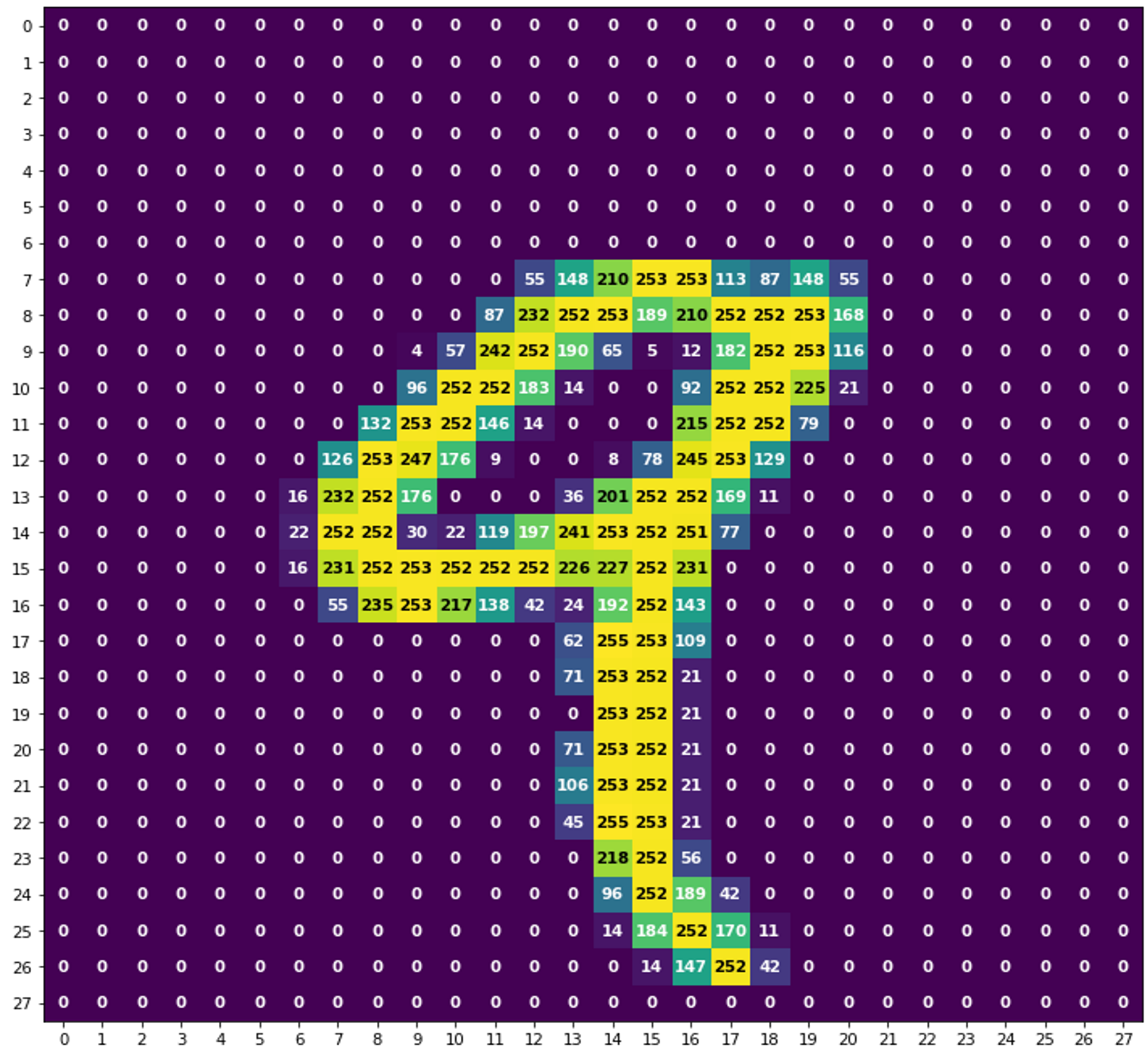
- ✓ https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/
- ✓ Deep Learning UC Berkeley STAT-157 2019, https://www.youtube.com/watch?v=Va8WWRfw7Og&list=PLZSO_6-bSqHQHBCoGaObUljoXAyyqhpFW
- ✓ **Linear Algebra – Gilbert Strang:** https://www.youtube.com/watch?v=dtX8iQGQQkI&list=PLZSO_6-bSqHQHBCoGaObUljoXAyyqhpFW&index=4
- ✓ http://www-math.mit.edu/~gs/ , **video course**
- ✓ **Matrix Applied Linear Algebra-Jim Demmel** https://people.eecs.berkeley.edu/~demmel/ma221_Spr20/Lectures/index.html
- ✓ https://www.3blue1brown.com/
- ✓ https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
- ✓ https://www.math.ucdavis.edu/~linear/linear-guest.pdf
- ✓ https://comp-think.github.io/
- ✓ https://edu.google.com/resources/programs/exploring-computational-thinking/
- ✓ **https://www.youtube.com/watch?v=Nc-V948dXWI**
- ✓ https://medium.com/swlh/linear-algebra-in-python-b967061e342a
- ✓ https://www.youtube.com/watch?v=JSjWltL9-7M
- ✓ https://pabloinsente.github.io/intro-linear-algebra
- ✓ https://rlhick.people.wm.edu/stories/linear-algebra-python-basics.html
- ✓ https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html **(excellent)**

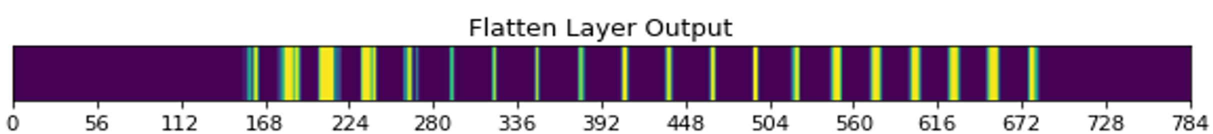
Example: how do computer see images?

Handwritten Digit Recognition (MNIST data set)

The **MNIST database** (*Modified National Institute of Standards and Technology database*) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.[4][5] It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels. The MNIST database contains 60,000 training images and 10,000 testing images. – from Wikipedia







Grayscale image of 28 x 28 pixels

same as a 28 x 28 matrix

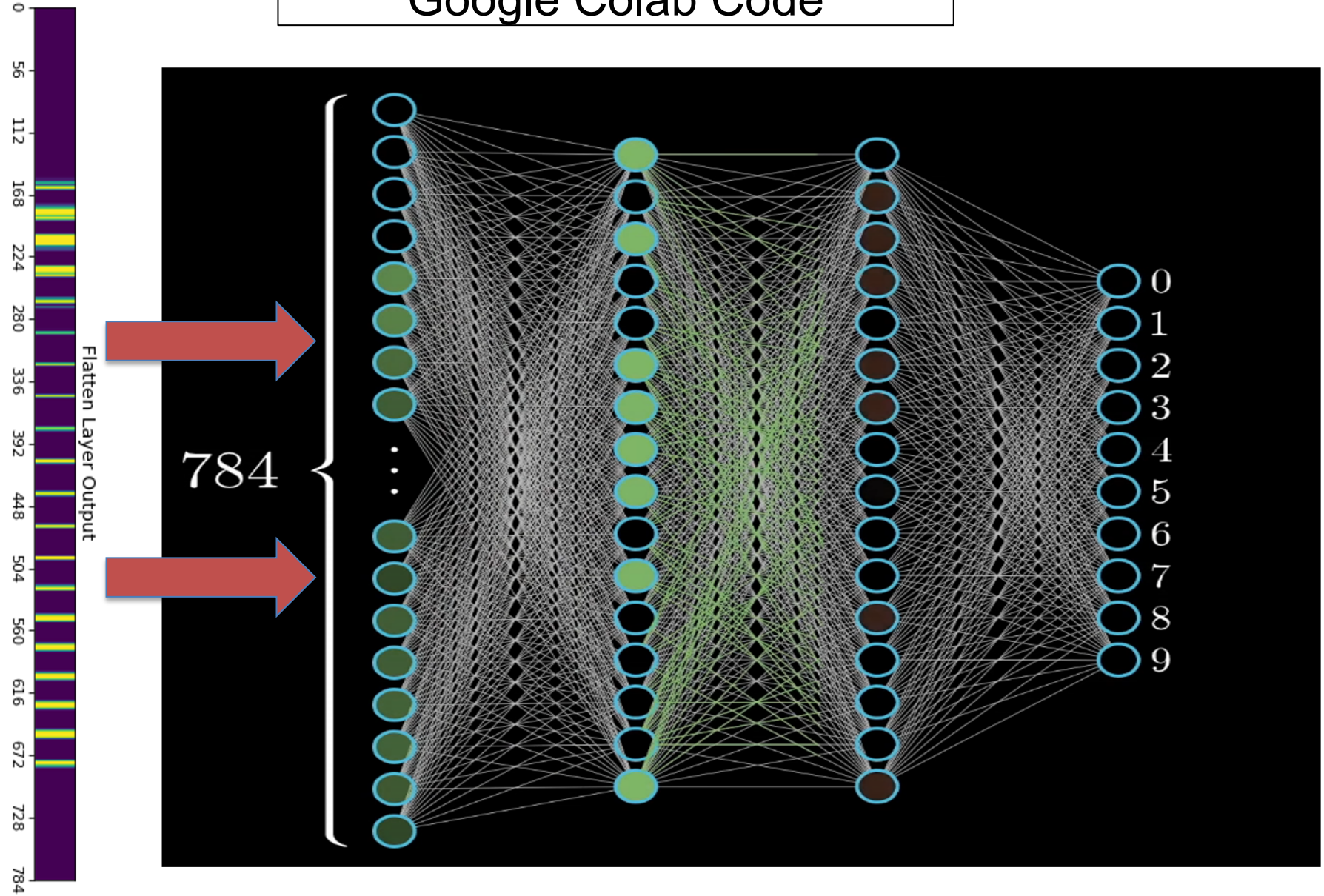
values of 0 - 255

Flatten the 28 x 28 matrix to a vector of 28 x 28 elements

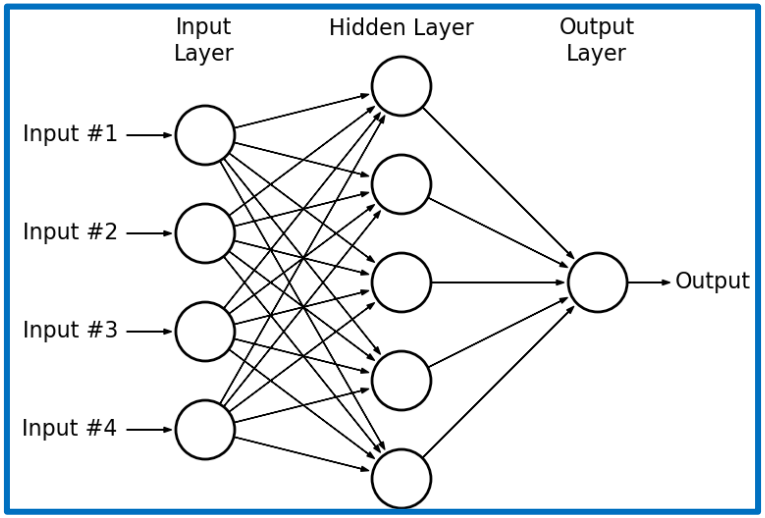
784 elements

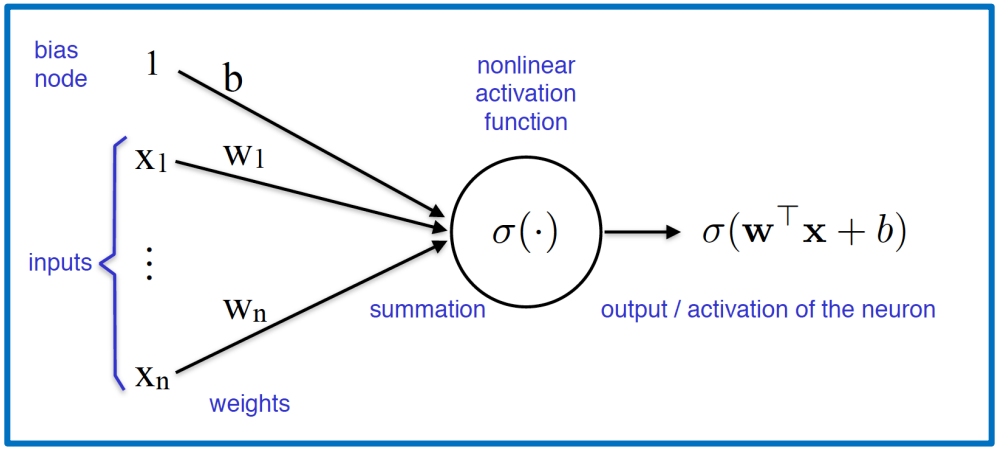
Simple MNIST MLP Network

Google Colab Code



NN Model – Vectors, Matrices, Tensors Computation





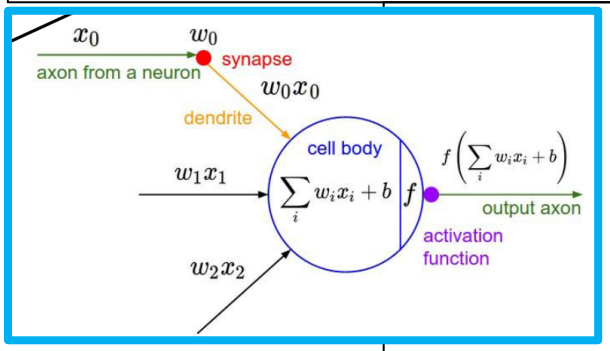
- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Inputs represent a set of vectors containing weights (w) and bias (b). They are the sets of parameters to be determined.

- ✓ A collection of data are organized in arrays of vectors or matrices.
- ✓ In a computer, the unknown parameters within the layers of a neural network are also represented as vectors and matrices
- ✓ Under the hood of the feed forward neural network (MLP) is just a composite function, that multiplies some matrices and vectors together.

LA Kernels in NN

https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494

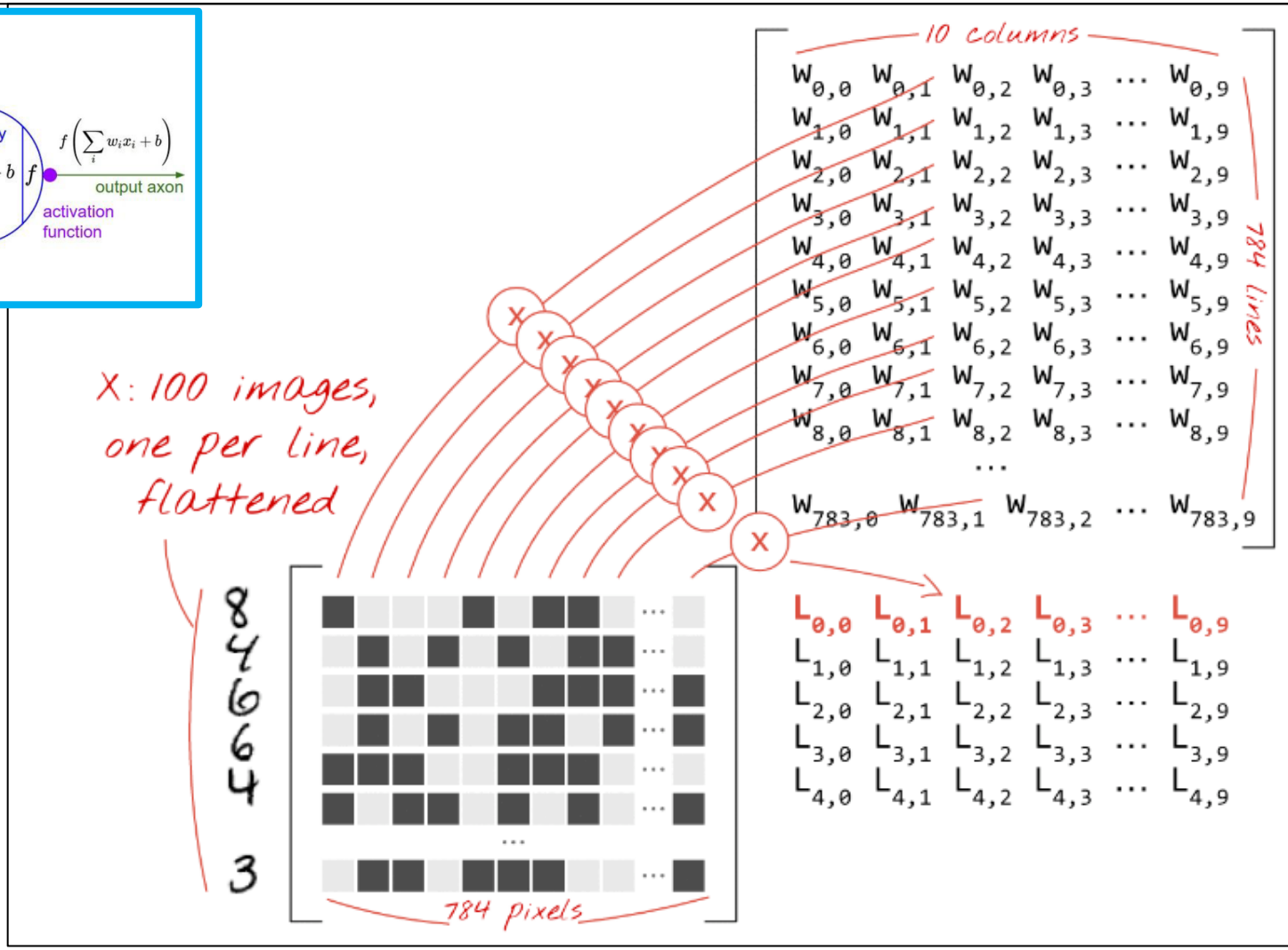
- ✓ In DNN calculations, data are organized in arrays of vectors or matrices.
- ✓ In a computer, the unknown parameters within the layers of a neural network as vectors and matrices
- ✓ Under the hood of the neural network simply involves many matrices and vectors multiplications.



Vector Vector operations

Matrix Vector operations

Matrix Matrix operations

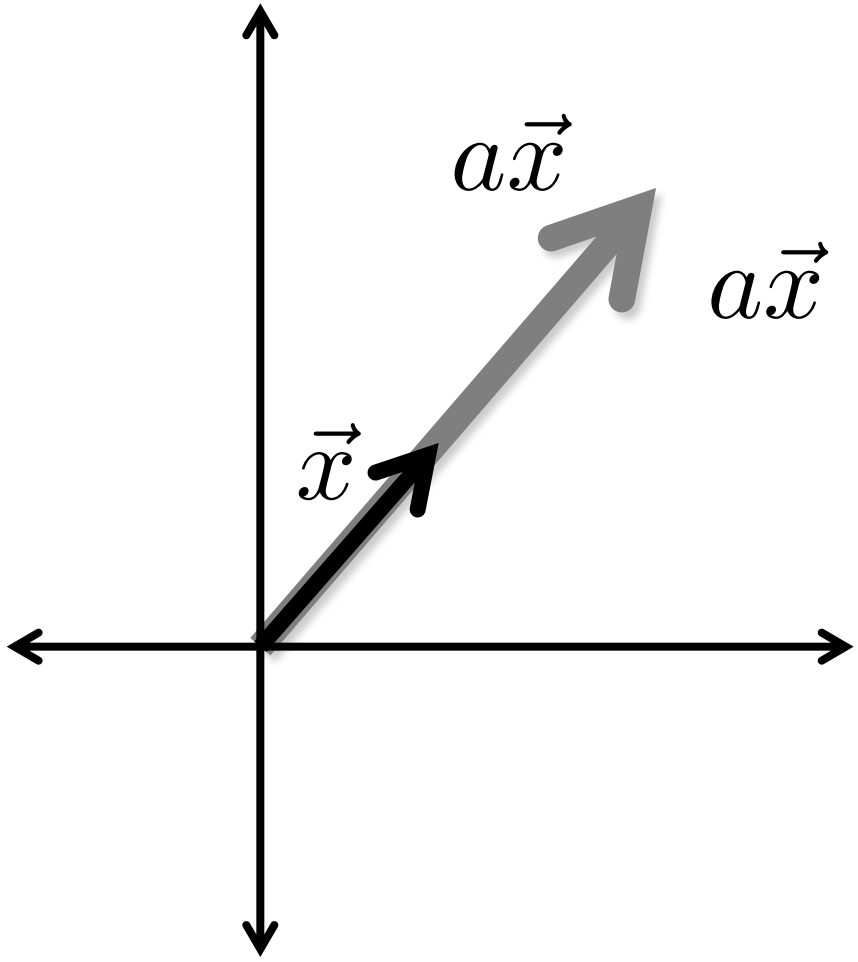


Introduction to Linear Algebra

Mark Goldman
Emily Mackevicius

Begin Here

Scalar times vector



$$a\vec{x} = a \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} ax_1 \\ ax_2 \\ \vdots \\ ax_N \end{pmatrix}$$

Product of 2 Vectors

Three ways to multiply

- Element-by-element
- Inner product
- Outer product

Element-by-element product (Hadamard product)

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} .* \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \end{pmatrix}$$

- Element-wise multiplication (.* in MATLAB)

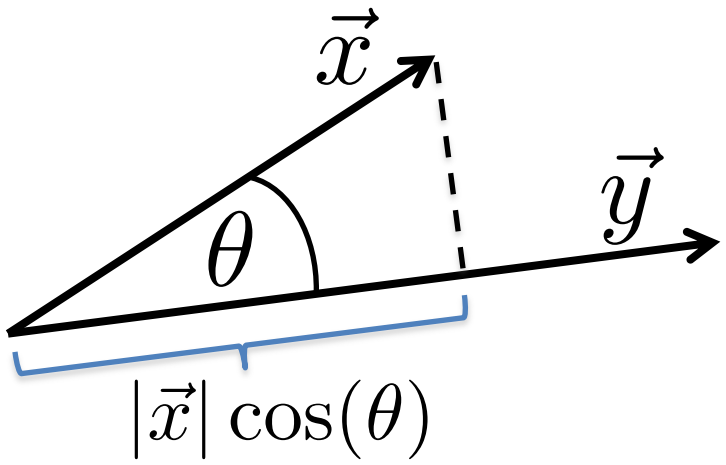
Multiplication:
Dot product (inner product)

$$\vec{x} \cdot \vec{y} =$$

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_N \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = x_1 y_1 + x_2 y_2 + \cdots + x_N y_N$$

$$= \sum_{i=1}^{N} x_i y_i$$

Dot product geometric intuition: "Overlap" of 2 vectors



$$\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}|\cos(\theta)$$

Matrix times a vector

$$\overrightarrow{y} = \overleftrightarrow{W}\,\overrightarrow{x}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

Matrix times a vector

$$\overrightarrow{y} = \overleftrightarrow{W}\overrightarrow{x}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

M X 1 M X N N X 1

Matrix times a vector: **inner product interpretation**

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{iN} \\ \vdots & \vdots & \ddots & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- Rule: the i^{th} element of **y** is the dot product of the i^{th} row of W with **x**

Matrix times a vector: **inner product interpretation**

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{iN} \\ \vdots & \vdots & \ddots & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- Rule: the i^{th} element of **y** is the dot product of the i^{th} row of W with **x**

Matrix times a vector: **inner product interpretation**

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{iN} \\ \vdots & \vdots & \ddots & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- Rule: the i^{th} element of **y** is the dot product of the i^{th} row of W with **x**

Matrix times a vector: **inner product interpretation**

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{iN} \\ \vdots & \vdots & \ddots & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- Rule: the i^{th} element of **y** is the dot product of the i^{th} row of W with **x**

Matrix times a vector: **inner product interpretation**

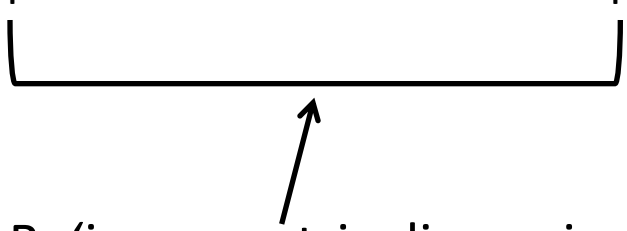
$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{iN} \\ \vdots & \vdots & \ddots & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- Rule: the i^{th} element of **y** is the dot product of the i^{th} row of W with **x**

Product of 2 Matrices

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & \vdots & & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NP} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots \\ B_{P1} & B_{P2} & \cdots & B_{PM} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NM} \end{pmatrix}$$

N X P P X M N X M



- MATLAB: ‘inner matrix dimensions must agree’
- **Note:** Matrix multiplication doesn’t (generally) commute, **AB** ≠ **BA**

Matrix times Matrix:
by inner products

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & \vdots & & \vdots \\ A_{i1} & A_{i2} & \cdots & A_{iP} \\ \vdots & \vdots & & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NP} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1j} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2j} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ B_{P1} & B_{P2} & \cdots & B_{Pj} & \cdots & B_{PM} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & C_{ij} & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NM} \end{pmatrix}$$

- C_{ij} is the inner product of the i^{th} row with the j^{th} column

Matrix times Matrix:
by inner products

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & \vdots & & \vdots \\ A_{i1} & A_{i2} & \cdots & A_{iP} \\ \vdots & \vdots & & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NP} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1j} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2j} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ B_{P1} & B_{P2} & \cdots & B_{Pj} & \cdots & B_{PM} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & C_{ij} & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NM} \end{pmatrix}$$

- C_{ij} is the inner product of the i^{th} row with the j^{th} column

Matrix times Matrix:
by inner products

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & \vdots & & \vdots \\ A_{i1} & A_{i2} & \cdots & A_{iP} \\ \vdots & \vdots & & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NP} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1j} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2j} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ B_{P1} & B_{P2} & \cdots & B_{Pj} & \cdots & B_{PM} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & C_{ij} & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NM} \end{pmatrix}$$

- C_{ij} is the inner product of the i^{th} row with the j^{th} column

Matrix times Matrix:
by inner products

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & \vdots & & \vdots \\ A_{i1} & A_{i2} & \cdots & A_{iP} \\ \vdots & \vdots & & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NP} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1j} & \cdots & B_{1M} \\ B_{21} & B_{22} & \cdots & B_{2j} & \cdots & B_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ B_{P1} & B_{P2} & \cdots & B_{Pj} & \cdots & B_{PM} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1M} \\ C_{21} & C_{22} & \cdots & C_{2M} \\ \vdots & \vdots & C_{ij} & \vdots \\ C_{N1} & C_{N2} & \cdots & C_{NM} \end{pmatrix}$$

$$C_{ij} = \sum_{k=1}^{P} A_{ik} B_{kj}$$

- C_{ij} is the inner product of the i^{th} row of **A** with the j^{th} column of **B**

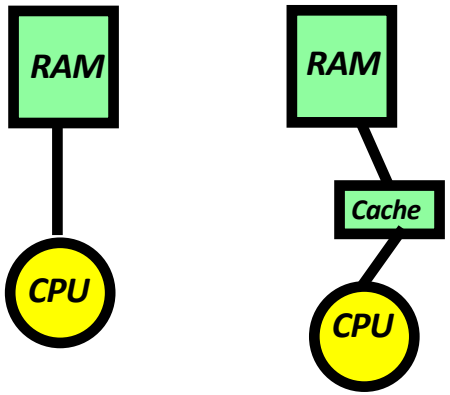
Introduction to Linear Algebra

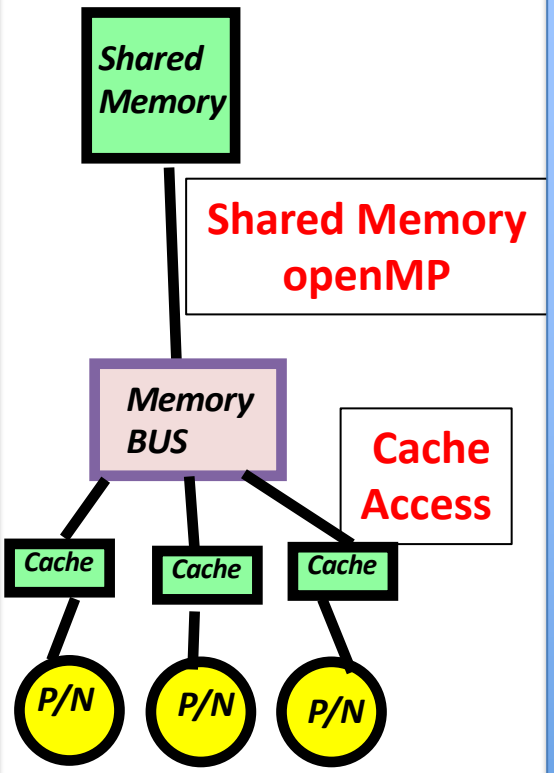
Mark Goldman
Emily Mackevicius

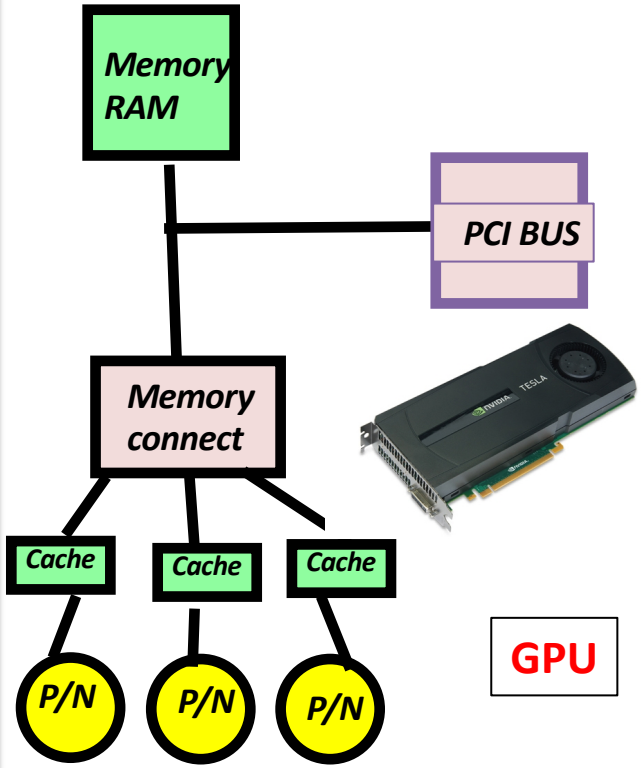
End Here

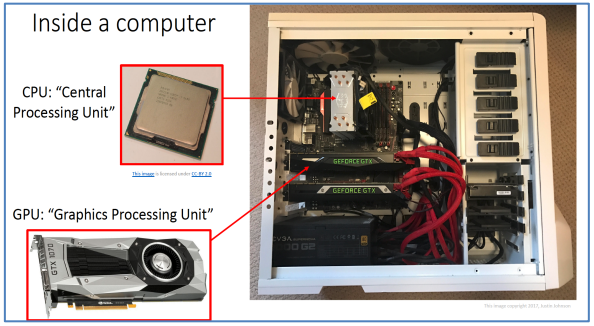
Simple story of Computers

Length Scale, Major Bottleneck, Memory Bound!!







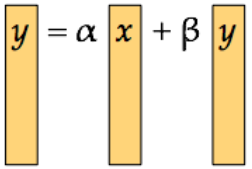


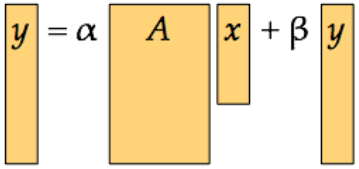
Basic Linear Algebra Subprograms (BLAS)

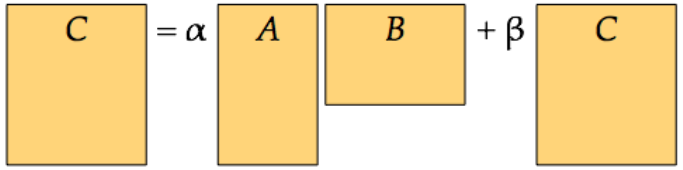
- **BLAS is a library of standardized basic linear algebra computational kernels created to perform efficiently on serial computers taking into account the memory hierarchy of modern processors.**
- **BLAS1 does vectors-vectors operations.**
  - **Saxpy = y(i) = a* x(i) + y(i), ddot= $\sum$x(i) *y(i)**
- **BLAS2 does matrices - vectors operations.**
  - **MV : y = A x + b (dgemv)**
- **BLAS3 operates on pairs or triples of matrices.**
  - **MM : C = αAB + βC, Triangular Solve : X = $\alpha T^{-1}X$**
- **Level3 BLAS is created to take full advantage of the fast cache memory. Matrix computations are arranged to operate in block fashion. Data residing in cache are reused by small blocks of matrices. dgemm**
- **Atlas, openBLAS, MKL, ESSL, libsci, ACML, cuBLAS, BLIS**

Computational Intensity = FLOPS/ Memory Access

- ✓ Level 1 BLAS — vector operations
  - ✓ $O(n)$ data and flops (floating point operations)
  - ✓ Memory bound:
    $O(1)$ flops per memory access
- ✓ Level 2 BLAS — matrix-vector operations
  - ✓ $O(n^2)$ data and flops
  - ✓ Memory bound:
    $O(1)$ flops per memory access
- ✓ Level 3 BLAS — matrix-matrix operations
  - ✓ $O(n^2)$ data, $O(n^3)$ flops
  - ✓ Surface-to-volume effect
  - ✓ Compute bound:
    $O(n)$ flops per memory access



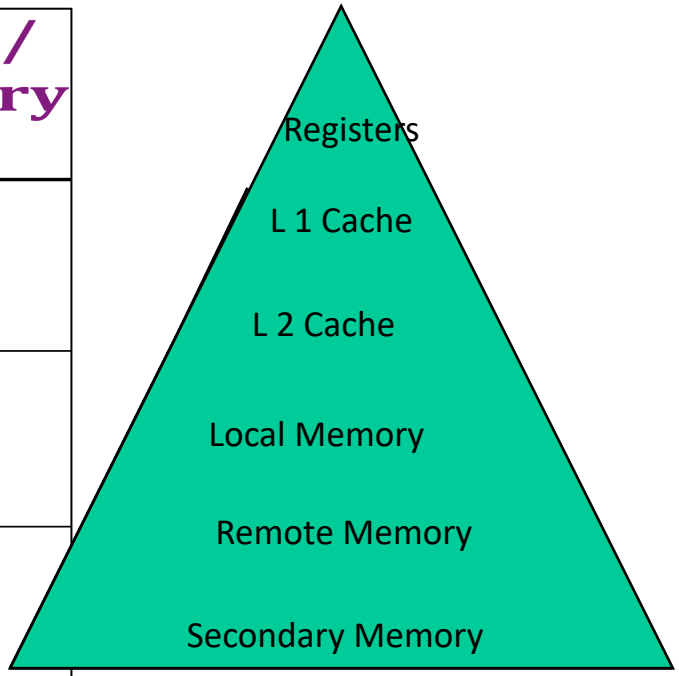




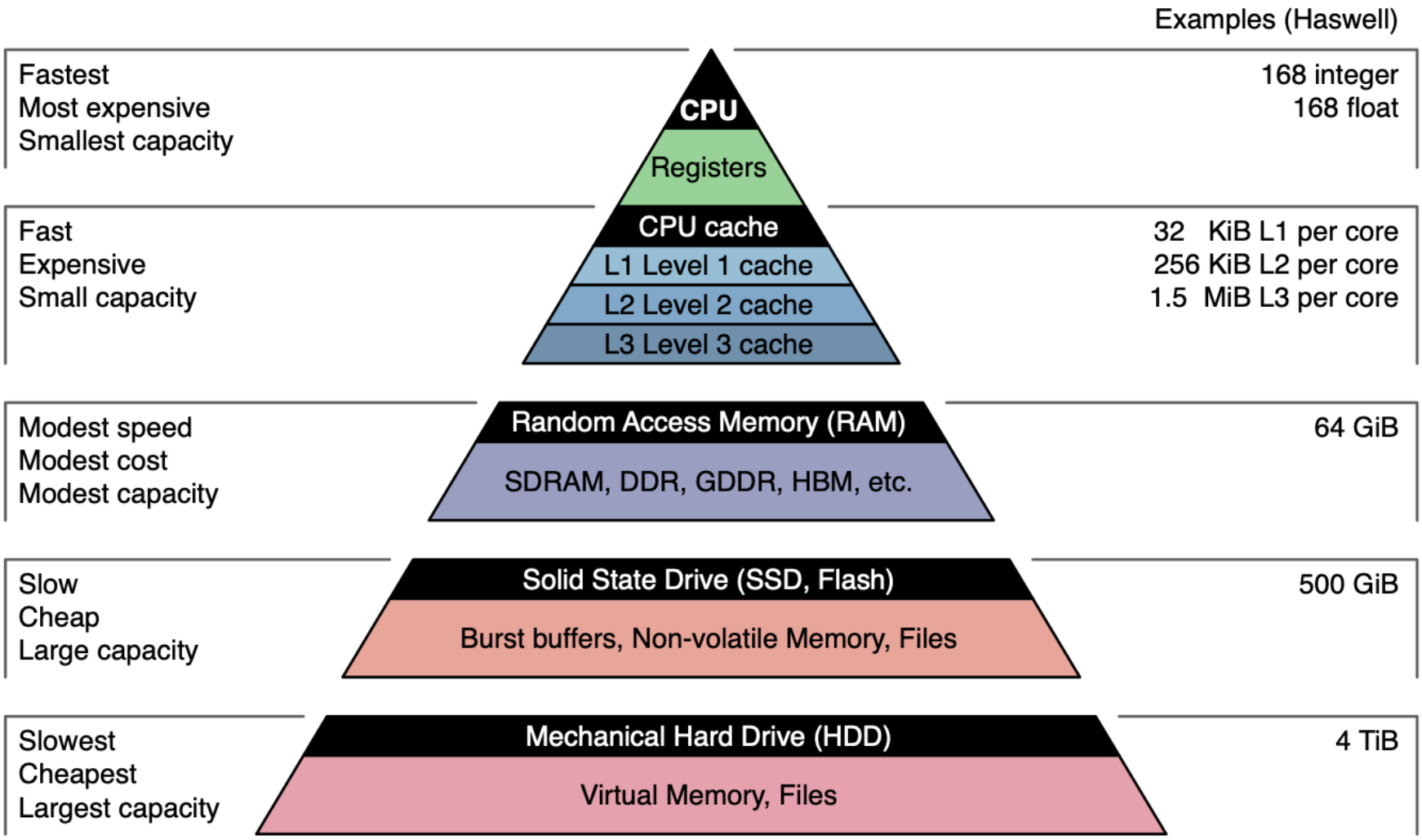
Why Higher Level BLAS?

- ✓ 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.
- ✓ 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 $C=C+AB$	$4n^2$	$2n^3$	$n/2$



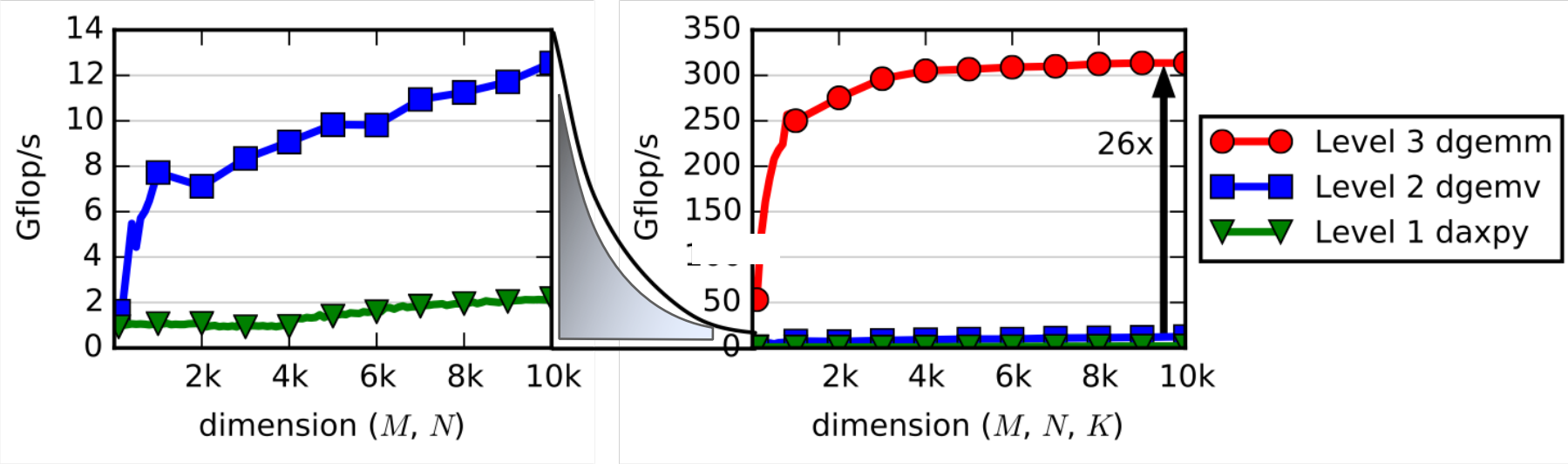
Memory hierarchy



Adapted from illustration by Ryan Leng

BLAS: Basic Linear Algebra Subroutines

- Intel Sandy Bridge (2 socket E5-2670)
  - Peak 333 Gflop/s = 2.6 GHz * 16 cores * 8 double-precision flops/cycle †
  - Max memory bandwidth 51 GB/s ‡

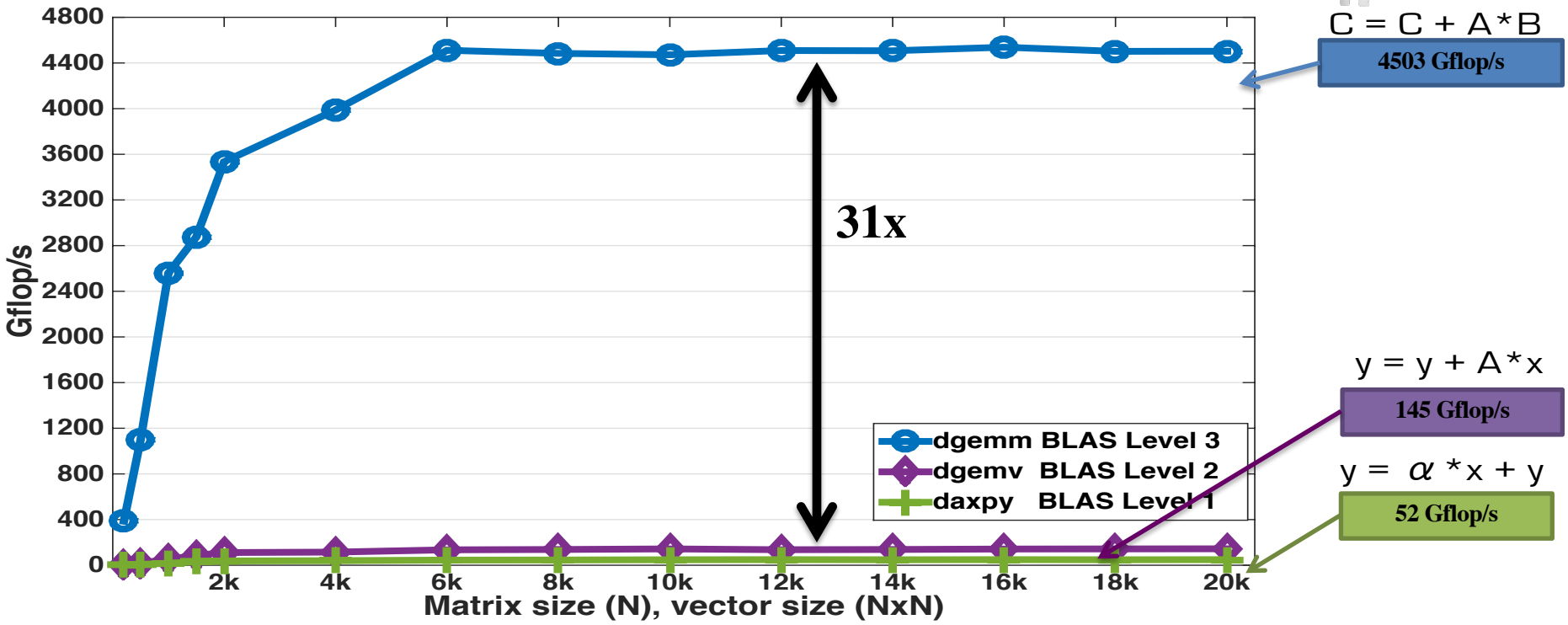


† http://stackoverflow.com/questions/15655835/flops-per-cycle-for-sandy-bridge-and-haswell-sse2-avx-avx2

‡ http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI

Level 1, 2 and 3 BLAS

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 Gflop/s



C = C + A*B

4503 Gflop/s

y = y + A*x

145 Gflop/s

y = α *x + y

52 Gflop/s

Nvidia P100
The theoretical peak double precision is 4700 Gflop/s
CUDA version 8.0

BLAS Naming scheme

- One or two letter **data type** (precision)
  - –`i` = integer (e.g., index)
  - –`s` = single (float)
  - –`d` = double
  - –`c` = single-complex
  - –`z` = double-complex
- Two letter **matrix type** (BLAS 2, 3)
  - –`ge` = general nonsymmetric
  - –`sy` = symmetric $(A = A^T)$
  - –`he` = complex Hermitian $(A = A^H)$
  - –`tr` = triangular $(L$ or $U)$
  - –Also banded and packed formats
- Two or more letter **function**, e.g.
  - –`mv` = matrix-vector product
  - –`mm` = matrix-matrix product
  - –`etc.`

Example

`dgemm` $C = AB + C$

BLAS naming scheme

- One or two letter **data type**
  - `i` = integer (e.g., index)
  - `s` = single (float)
  - `d` = double
  - `c` = single-complex
  - `z` = double-complex
- Two letter **matrix type** (BLAS 2, 3)
  - `ge` = general nonsymmetric
  - `sy` = symmetric $(A = A^T)$
  - `he` = complex Hermitian $(A = A^H)$
  - `tr` = triangular $(L$ or $U)$
  - Also banded and packed formats
- Two or more letter **function**, e.g.
  - `mv` = matrix-vector product
  - `mm` = matrix-matrix product
  - `etc.`

BLAS 1 examples

`sdot`	$result = x^T y$ (single)
`ddot`	$result = x^T y$ (double)
`cdotc`	$result = x^H y$ (single-complex)
`cdotu`	$result = x^T y$ (single-complex)
`zdotc`	$result = x^H y$ (double-complex)
`zdotu`	$result = x^T y$ (double-complex)
`_axpy`	$y = \alpha x + y$
`_scal`	$y = \alpha y$
`_copy`	$y = x$
`_swap`	$x \leftrightarrow y$
`_nrm2`	2-norm: $result = \|x\|_2$
`_asum`	1-norm: $result = \sum_i \|x_i\|$
`i_amax`	max/inf norm: $index = \text{argmax}_i \|x_i\|$
`_rotg`	generate Given's rotation
`_rot`	apply Given's rotation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

BLAS naming scheme

- One or two letter **data type**
  - –i = integer (e.g., index)
  - –s = single (float)
  - –d = double
  - –c = single-complex
  - –z = double-complex
- Two letter **matrix type** (BLAS 2, 3)
  - –ge = general nonsymmetric
  - –sy = symmetric $(A = A^T)$
  - –he = complex Hermitian $(A = A^H)$
  - –tr = triangular $(L$ or $U)$
  - –Also banded and packed formats
- Two or more letter **function**, e.g.
  - –mv = matrix-vector product
  - –mm = matrix-matrix product
  - –etc.

BLAS 2 examples

_gemv	$y = Ax + y,$	A general
_symv	$y = Ax + y,$	A symmetric
_hemv	$y = Ax + y,$	A Hermitian
_ger	$C = xy^T + C,$	C general
_syr	$C = xx^T + C,$	C symmetric
_her	$C = xx^H + C,$	C Hermitian
_trmv	$x = Ax,$	A triangular
_trsv	*solve* $Ax = b,$	A triangular

BLAS 3 examples

_gemm	$C = AB + C,$	all general
_symm	$C = AB + C,$	A symmetric
_hemm	$C = AB + C,$	A Hermitian
_syrk	$C = AA^T + C,$	C symmetric
_herk	$C = AA^H + C,$	C Hermitian
_trmm	$X = AX$ *or* $XA,$	A triangular
_trsm	*solve* $AX = B,$	A triangular
	or $XA = B$	

A Simple Model of Memory Algorithmic Analysis -

- Assume just 2 levels in the hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - t_m = time per slow memory operation (inverse bandwidth in best case)
  - f = number of arithmetic operations
  - t_f = time per arithmetic operation << t_m
  - $CI = f / m$ average number of flops per slow memory access
- Minimum possible time = $f * t_f$ when all data in fast memory
- Actual time
  - $f * t_f + m * t_m = f * t_f * (1 + t_m/t_f * 1/CI)$
- Larger CI means time closer to minimum $f * t_f$

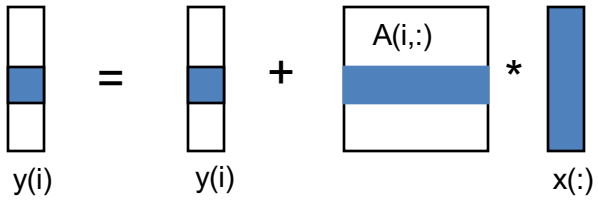
Computational Intensity (CI): **Key to algorithm efficiency**

Machine Balance: **Key to machine efficiency**

Matrix-vector multiplication
major operation for iterative methods

{implements y = y + A*x}

```
for i = 1:n
      for j = 1:n
            y(i) = y(i) + A(i,j)*x(j)
```



f (FLOP) = number of arithmetic operations = $2n^2$

Matrix-vector multiplication

```
{read x(1:n) into fast memory}.  --- 1 n
{read y(1:n) into fast memory}.  --- 1 n
for i = 1:n
        {read row i of A into fast memory}. --- n^2
        for j = 1:n
         y(i) = y(i) + A(i,j)*x(j)
{write y(1:n) back to slow memory}  --- 1 n
```

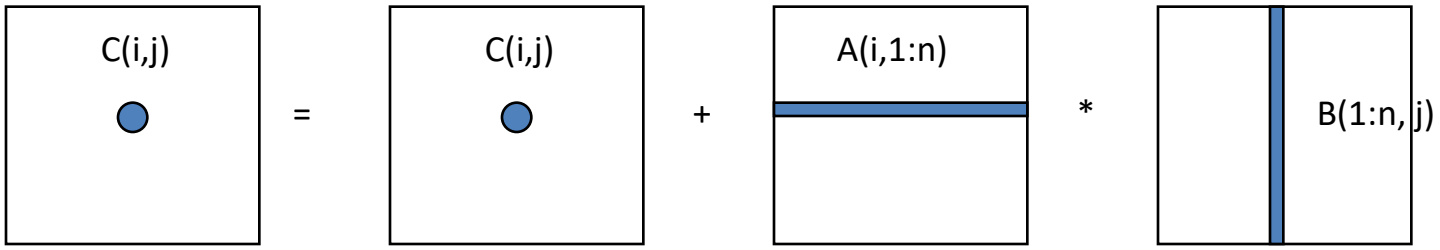
- m = number of slow memory refs = $3n + n^2$
- f = number of arithmetic operations = $2n^2$
- q = $f / m \approx 2$ (Low Computational Intensity)

- Matrix-vector multiplication limited by slow memory speed

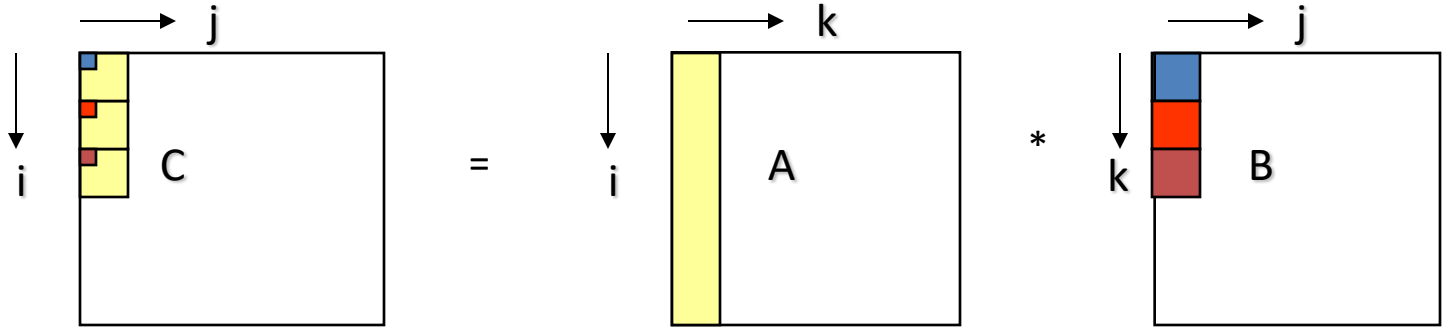
Note: Shown with 1-based indexing for simplicity

MM Multiplication

- **Simple MM - q = average number of flops per memory reference ~ 2**



- **Performance of MM can be improved by rearranging the order of multiplication indices in column fashion in Fortran or in row fashion in C.**



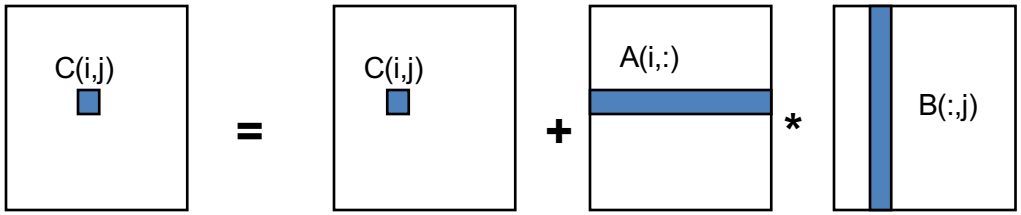
k - j - i ordering for FORTRAN

Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
        for j = 1 to n
        for k = 1 to n
                C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

Algorithm has $2*n^3 = O(n^3)$ Flops and operates on $3*n^2$ words of memory

Computational intensity (q) *potentially* as large as $2*n^3 / 3*n^2 = O(n)$



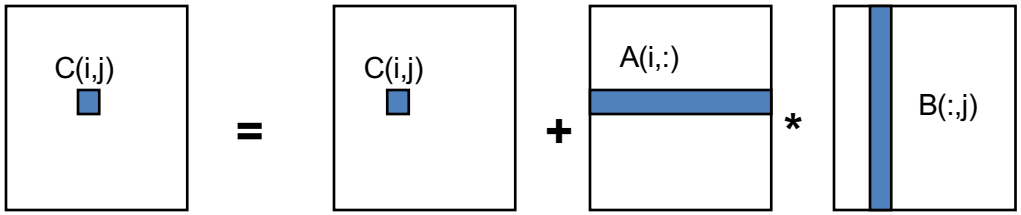
Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
      for j = 1 to n
      for k = 1 to n
                  C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

Algorithm has $2*n^3 = O(n^3)$ Flops

Need only $3*n^2$ words of data in memory

What is the actual number of access to slow memory here



Naïve Matrix Multiply

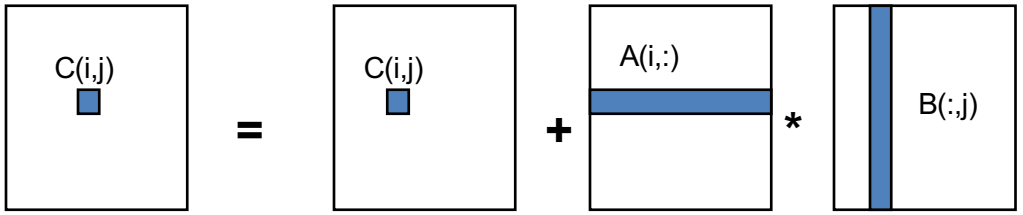
```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}
   for j = 1 to n
      {read C(i,j) into fast memory}
      {read column j of B into fast memory}
      for k = 1 to n
          C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write C(i,j) back to slow memory}
```

# of slow memory ops:

$m = n^3$ to read each column of B n^2 times
$+ n^2$ to read each row of A once
$+ 2n^2$ to read and write each element of C once
$= n^3 + 3n^2$

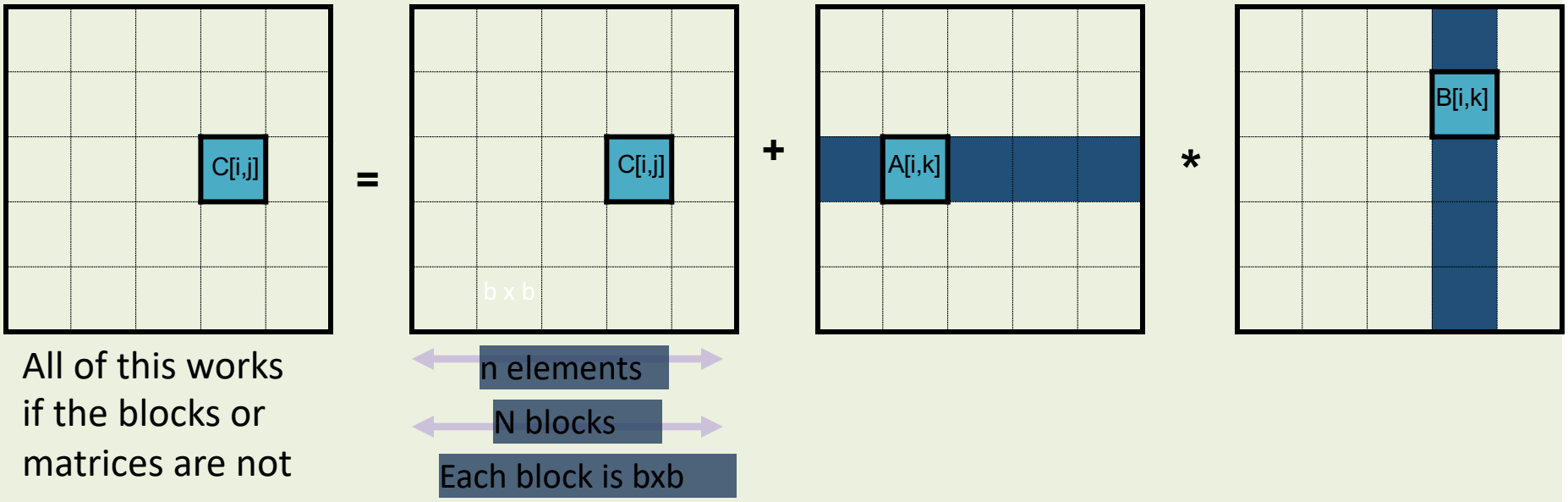
So $q = f / m = 2n^3 / (n^3 + 3n^2)$ computational intensity
≈ 2 for large n, no improvement over matrix-vector multiply

Computational Intensity q = 2 !!



Blocked [Tiled] Matrix Multiply

Consider A,B,C to be N-by-N matrices of b-by-b subblocks where b=n / N is called the block size



All of this works if the blocks or matrices are not square

Summary – Blocked MM

of slow memory ops: (Naïve MM)

$m = n^3$ to read each column of B n times (for each row of A)

$+ n^2$ to read each row of A once (n row of A)

$+ 2n^2$ to read and write each element of C once

$= n^3 + 3n^2$

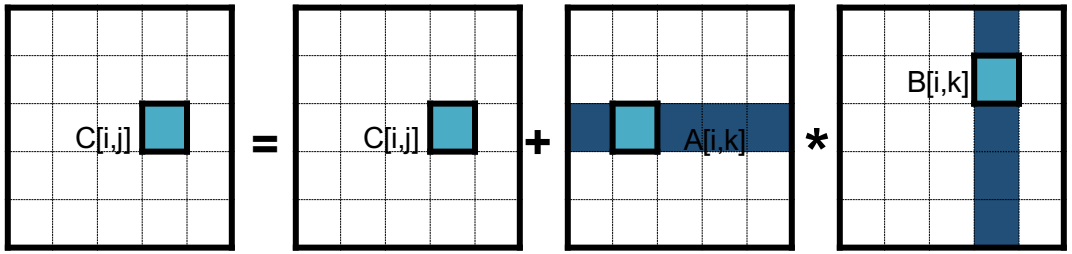
So $q = f / m = 2n^3 / (n^3 + 3n^2)$ computational intensity

≈ 2 for large n, no improvement over matrix-vector multiply

A, B, C = nxn elements

NxN blocks

Each block is bxb



Consider A,B,C to be N-by-N matrices of b-by-b subblocks

b = n / N

n = N * b

If N=n, n^3

to read each block of B = (b)^2, N^3 times

Access of B : $N*n^2$

to read each block of A = (b)^2, N^3 times

Access of A : ($N^3 * b^2 = N^3 * (n/N)^2$) = $N*n^2$

$2n^2$ to read and write each block of C once to slow memory, Access of C : ($2N^2 * b^2 = 2n^2$)

Blocked (Tiled) Matrix Multiply

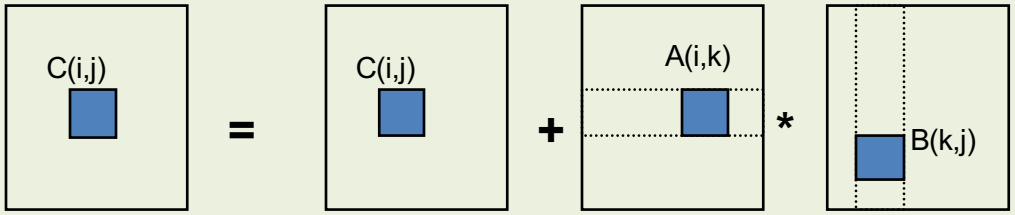
Consider A,B,C to be N-by-N matrices of b-by-b subblocks where
b=n / N is called the block size

```
for i = 1 to N
  for j = 1 to N
    {read block C(i,j) into fast memory}
    for k = 1 to N
      {read block A(i,k) into fast memory}
      {read block B(k,j) into fast memory}
      C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory}
```

3 nested loops inside

block size
b = loop bounds

choose b to fit in cache



Tiling for registers or caches

Blocked [Tiled] Matrix Multiply

Consider A,B,C to be N-by-N matrices of b-by-b subblocks
where b=n / N is called the block size

```
for i = 1 to N
   for j = 1 to N
    {read block C[i,j] into fast memory}
      for k = 1 to N
       {read block A[i,k] into fast memory}
       {read block B[k,j] into fast memory}
           C[i,j] = C[i,j] + A[i,k] * B[k,j] {do a matrix multiply on blocks}

    {write C[i,j] back to slow memory}
```

$2n^2$ to read and write each block of C once

$N*n^2$ to read each block of A N^3 times

$N*n^2$ to read each block of B N^3 times

Memory words moved: $m = 2n^2 + N*n^2 + N*n^2 = 2n^2 (1+N)$

Blocked [Tiled] Matrix Multiply

Consider A,B,C to be N-by-N matrices of b-by-b subblocks

where b=n / N is called the block size

```
for i = 1 to N
   for j = 1 to N
   {read block C[i,j] into fast memory}
   for k = 1 to N
      {read block A[i,k] into fast memory}
      {read block B[k,j] into fast memory}
         C[i,j] = C[i,j] + A[i,k] * B[k,j] {do a matrix multiply on blocks}
      {write C[i,j] back to slow memory}
```

$2n^2$ to read and write each block of C once

$N*n^2$ to read each block of A N^3 times

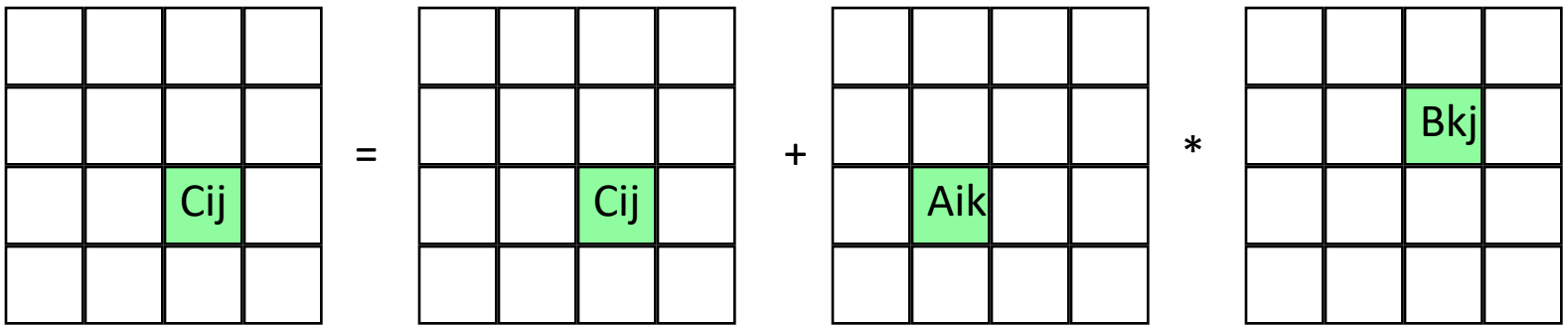
$N*n^2$ to read each block of B N^3 times

Computational Intensity, CI = $f / m = 2n^3 / ((2N + 2) * n^2)$

$\approx n / N = b$ for large n

Block MM

- Computational intensity = q = f/m = (2*n^3) / ((2*N + 2) * n^2) ~ n / N
- If N is equal to 1 (b = n), all A, B, C fit in the cache, the algorithm is ideal. However, N is bounded by the amount of fast cache memory. However, N can be taken independently to the size of matrix, n.
- The optimal value of b = sqrt (size of fast memory / 3)

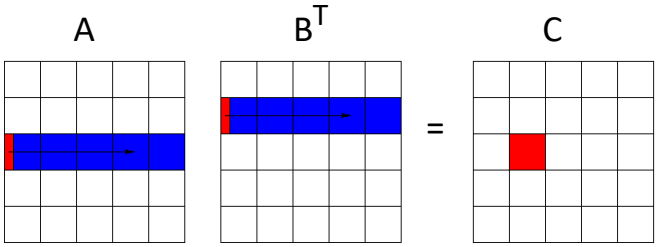


$$C_{ij} = C_{ij} + \sum_{k=1}^{n} A_{ik} * B_{kj}$$

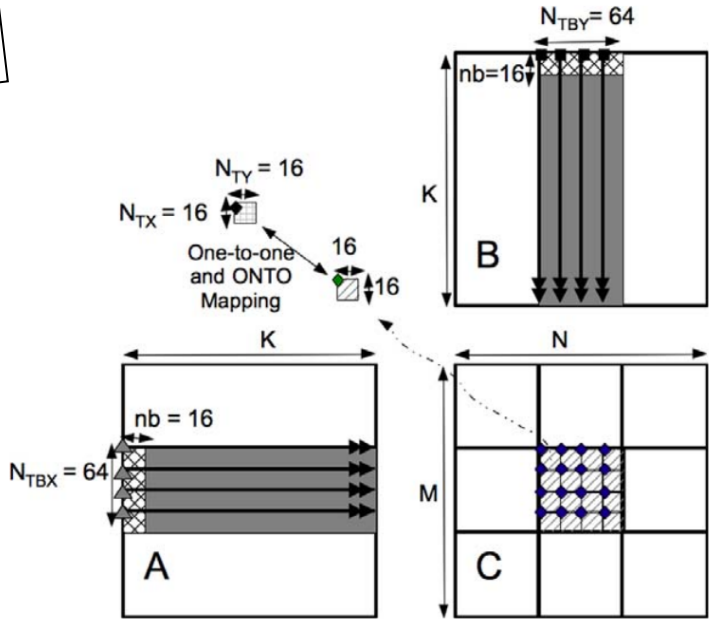
GEMM in MAGMA

2010. R. Nath, S. Tomov and J. Dongarra. ***An improved MAGMA GEMM for Fermi Graphics Processing Units***, The International Journal of High Performance Computing Applications.
http://www.netlib.org/utk/people/JackDongarra/journals/208_2010_an-improved-magma-gemm-for-fermi-gpus.pdf

- **Add register blocking**
- **Parameterized for autotuning for particular size GEMMs and portability across GPUs**



* **Small** red rectangles (to overlap communication & computation) are of size 32 x 4 and are red by 32 x 2 threads



A thread computes part of a row (16 values) of the C block

A thread computes a block of C (4 x 4 values in this case)

Matrix Multiplication, CPU, GPU

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

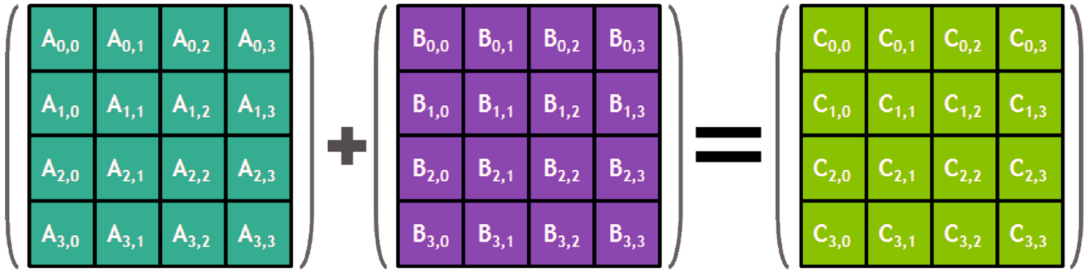
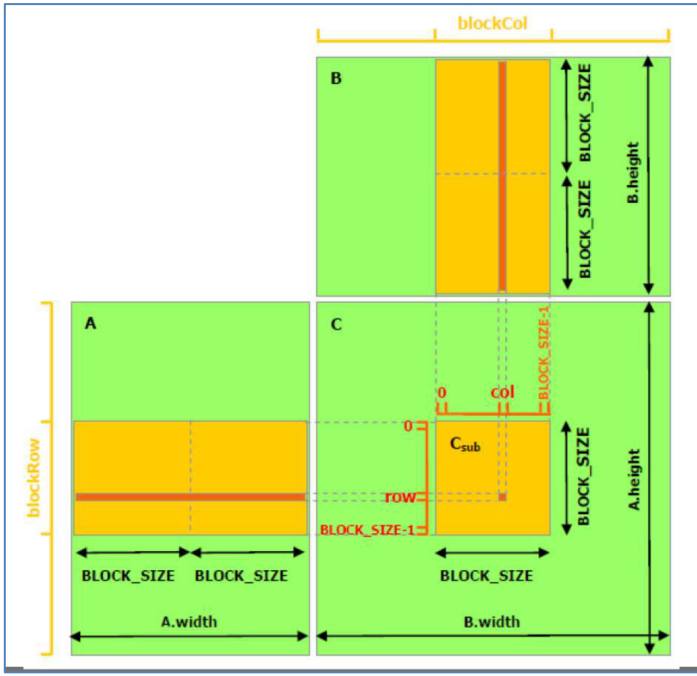
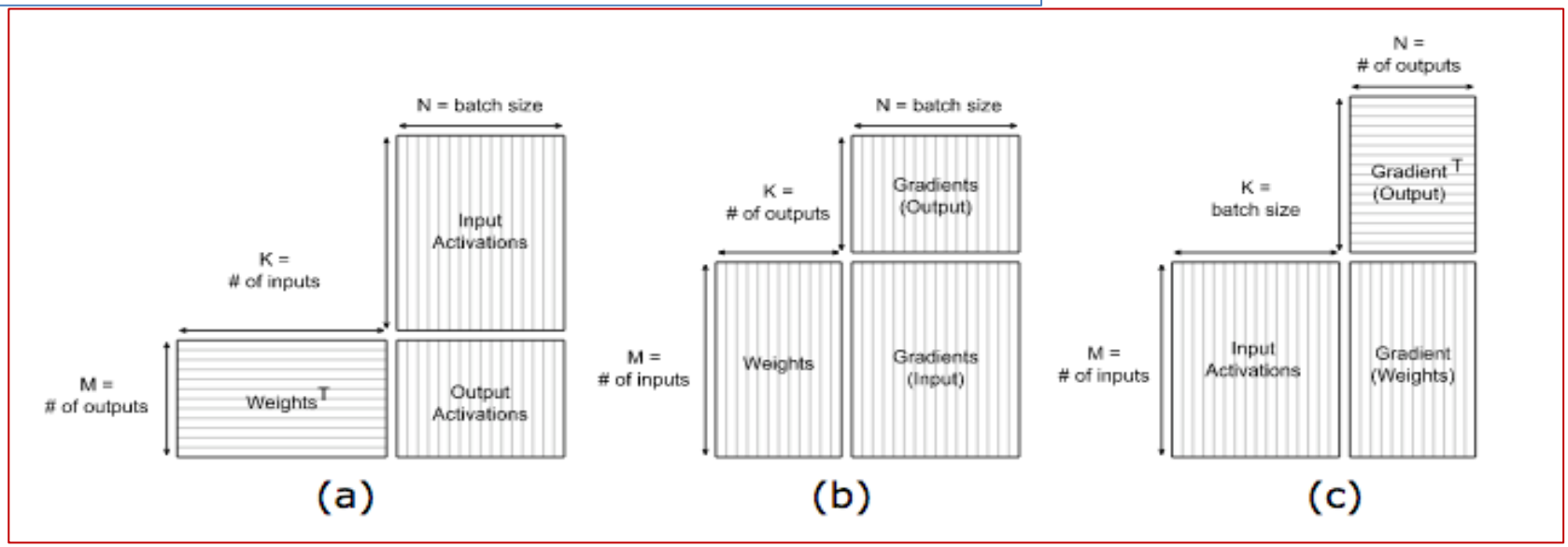


Table 1. Mapping of inputs, outputs, and batch size to GEMM parameters M, N, K.

Computation Phase	M	N	K
Forward Propagation	Number of outputs	Batch size	Number of inputs
Activation Gradient	Number of inputs	Batch size	Number of outputs
Weight Gradient	Number of inputs	Number of outputs	Batch size





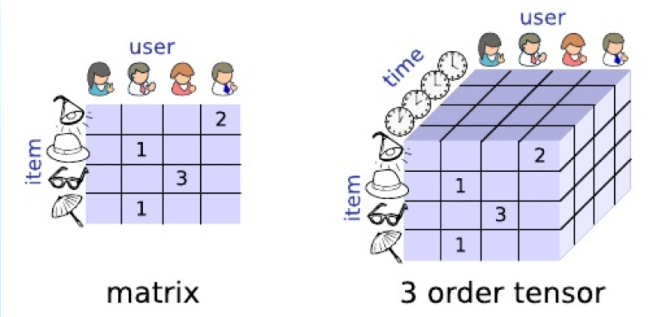
Magma/2.5.2 on bridges GPU

```
[wongk@gpu046 ~]$ module load cuda/10.1
[wongk@gpu046 ~]$ module list
Currently Loaded Modulefiles:
  1) psc_path/1.1    2) slurm/default   3) intel/19.5      4) xdusage/2.1-1   5) Magma/2.5.2     6) cuda/10.1
[wongk@gpu046 ~]$ cd /opt/packages/Magma/magma-2.5.2/testing

[wongk@gpu046 testing]$ ./testing_dgemm -N 5000:10000:500 --lapack
% MAGMA 2.5.2  compiled for CUDA capability >= 6.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 10010, driver 10020. OpenMP threads 16. MKL 2019.0.5, MKL threads 16.
% device 0: Tesla P100-PCIE-16GB, 1328.5 MHz clock, 16280.9 MiB memory, capability 6.0
% Fri Oct  2 15:04:44 2020
% Usage: ./testing_dgemm [options] [-h|--help]
% If running lapack (option --lapack), MAGMA and cuBLAS error are both computed
% relative to CPU BLAS result. Else, MAGMA error is computed relative to cuBLAS result.
% transA = No transpose, transB = No transpose
%   M     N     K   MAGMA Gflop/s (ms)  cuBLAS Gflop/s (ms)   CPU Gflop/s (ms)  MAGMA error  cuBLAS error
%=================================================================================================================
 5000  5000  5000   3646.66 (  68.56)    4299.59 (  58.15)    515.86 ( 484.63)    2.16e-17     2.16e-17   ok
 5500  5500  5500   3746.67 (  88.81)    4479.56 (  74.28)    531.21 ( 626.40)    2.16e-17     2.16e-17   ok
 6000  6000  6000   3715.05 ( 116.28)    4536.77 (  95.22)    520.63 ( 829.77)    2.12e-17     2.12e-17   ok
 6500  6500  6500   3693.40 ( 148.71)    4524.19 ( 121.40)    531.53 (1033.33)    2.06e-17     2.06e-17   ok
 7000  7000  7000   3673.32 ( 186.75)    4486.80 ( 152.89)    528.31 (1298.47)    1.99e-17     1.99e-17   ok
 7500  7500  7500   3669.31 ( 229.95)    4487.55 ( 188.02)    527.02 (1600.97)    1.92e-17     1.92e-17   ok
 8000  8000  8000   3704.85 ( 276.39)    4527.77 ( 226.16)    532.93 (1921.45)    1.85e-17     1.85e-17   ok
 8500  8500  8500   3699.27 ( 332.03)    4512.24 ( 272.20)    524.18 (2343.21)    1.94e-17     1.94e-17   ok
 9000  9000  9000   3687.36 ( 395.40)    4515.05 ( 322.92)    523.77 (2783.68)    2.06e-17     2.06e-17   ok
 9500  9500  9500   3669.67 ( 467.28)    4502.94 ( 380.81)    521.39 (3288.80)    2.12e-17     2.12e-17   ok
10000 10000 10000   3660.58 ( 546.36)    4491.71 ( 445.27)    515.41 (3880.44)    2.16e-17     2.16e-17   ok
```

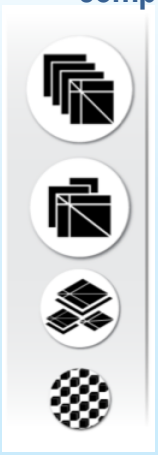
What about accelerated LA for Data Analytics?

- Traditional libraries like MAGMA can be used as backend to accelerate the LA computations in data analytics applications
- Need support for
  1) New data layouts, 2) Acceleration for small matrix computations, 3) Data analytics tools

Need data processing and analysis support for
Data that is multidimensional / relational



Small matrices, tensors, and batched computations



Fixed-size batches

Variable-size batches

Dynamic batches

Tensors

Exercises

1. A desktop computer has the following specification: quad-core, 24GB RAM, 2.5 GHz, and each core has 8 floating units (perform 8 DP operations in one clock cycle). What is the theoretical double precision peak performance of the desktop in FLOPS?

2. Given A is a M x K matrix, B is a K x N matrix, what is the number of floating point operation (FLOP) of A x B?

3. Given M=K=N for A and B, what is the maximum dimension N that the computer can hold?

4. Based on the answer above, what is the time needed to compute C=A x B if 90% of the theoretical rate of the desktop can be attained for the BLAS3 operation?

1. A desktop computer named MA6633DC has the following specification: quad-core, 24GB RAM, 2.5 GHz, and each core has 8 floating units (perform 8 DP operations in one clock cycle). What is the theoretical double precision peak performance of the desktop in FLOPS?

FLOPS in one core= (clock rate) x (floating point operation in one clock cycle)
Peak Rate = (FLOPS in one core) x (no. of core)
Peak FLOPS = 2.5 GHz * 8 * 4 cores = 80 GFLOPS

2. Given A is a M x K matrix, B is a K x N matrix, what is the number of floating point operation (FLOP) of A x B?

C=A x B ; A : A(M , K) ; B : B(K, N) , C = C(M, N) ; one element of C need (2K-1) DP
FLOP = (2K-1) x M x N

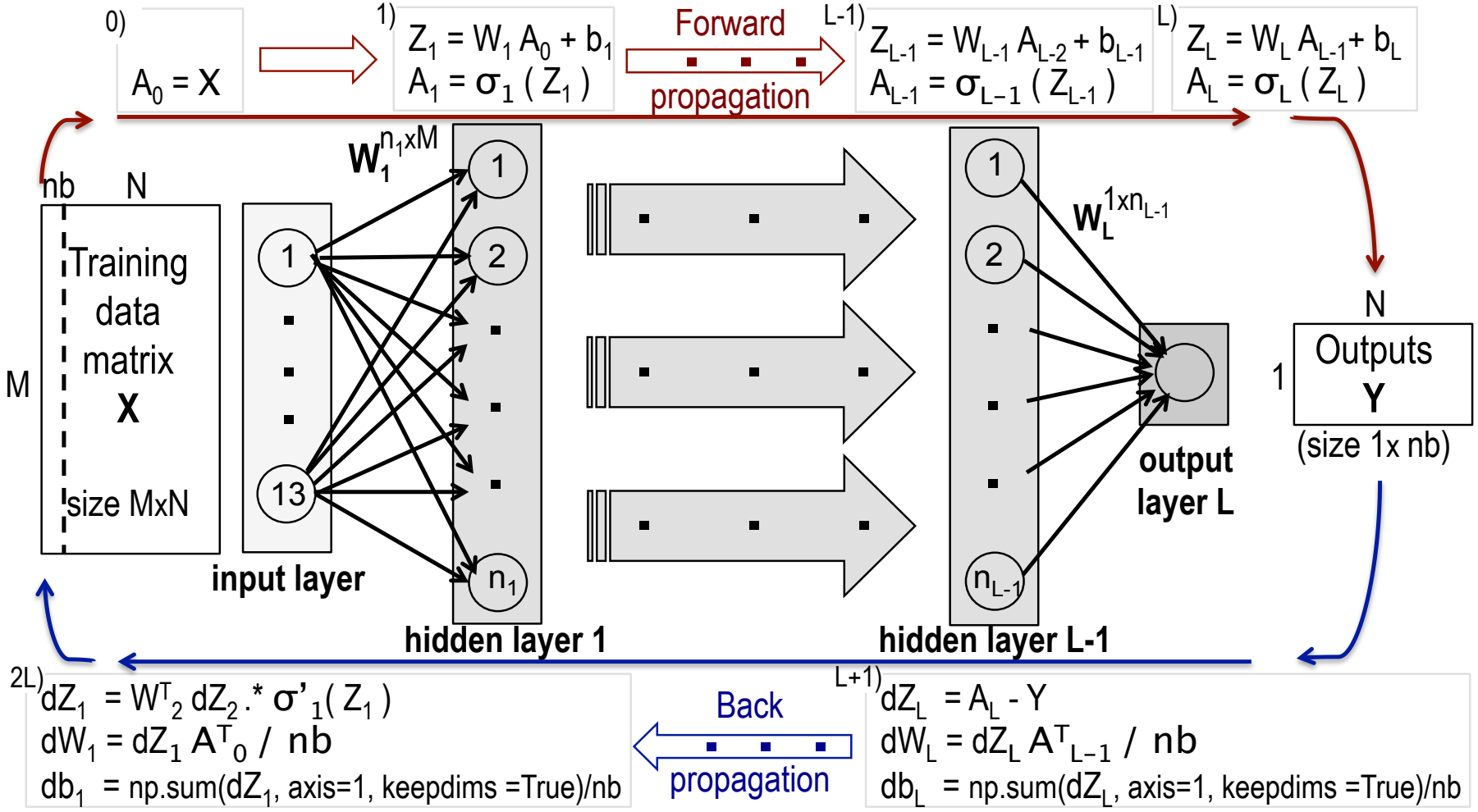
3. Given M=K=N for A and B, what is the maximum dimension N that MA6633DC can hold?

For double precision float, we need 8 bytes to store each element.
If M=N=K=N, in total we need 3N^2 elements
3N^2 * 8 = 24 GB, thus **N = floor(sqrt(1e9)) = 31622**

4. Based on the answer above, what is the time needed to compute C=A x B if 90% of the theoretical rate of MA6633DC can be attained for the BLAS3 operation?

90% * 80 GFLOPS = 72 GFLOPS
The FLOP we need to compute A x B is ~2N^3 ; **Time = 2 (31622)^3 FLOP / 72 GFLOPS = 878 seconds**

DNN MLP LA Kernels



Section 1: Big Science and Big Data Ecosystem

- **Advance Computing system**
  - **Top500, FLOPS and Performance**
  - **DOE Exascale Road Map**
  - **NSF infrastructure**
  - **Desktop, Accelerators, GPUs**
  - **Google COLAB**
- **Predictive Simulation Science**
  - **Equation based simulation sciences**
  - **Mathematics Essentials, Calculus**
  - **Linear Algebra, BLAS**

- **Computer Sciences and Software tools**
  - **Computational thinking**
  - **Workflow**
  - **ML Framework**
  - **Languages**
- **Data Intensive Sciences**
  - **Statistical Learning**
  - **DNN**
  - **Bayes' Theorem**
  - **Data mining**

Software – Operating Systems

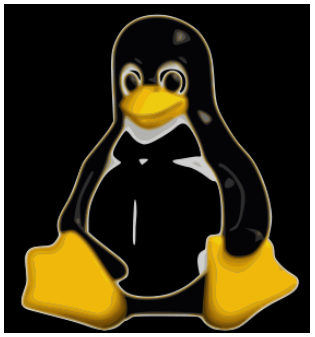
- ✓ Microsoft – Windows OS, 80% primarily for Desktop applications
- ✓ Mac – Mac OS – 10 % Desktop, BSD type, functionality similar to Linux
- ✓ Linux – open source, ubuntu, Centos, Redhat, Fedora, openSUSE, Debian,
- ✓ Entreprise : centos, Redat, ubuntu, SUSE, Cray OS..
- ✓ Phone, tablets : iOS, Android
- ✓ Virtual Box, VM,
- ✓ Container

- ✓ Cloud: Amazon, google cloud, MS Azure
- ✓ Free Access, Cheap device, …
- ✓ Web portal, google colab (GUI),
- ✓ Gateway, XSEDE gateway

Basics of LINUX OS

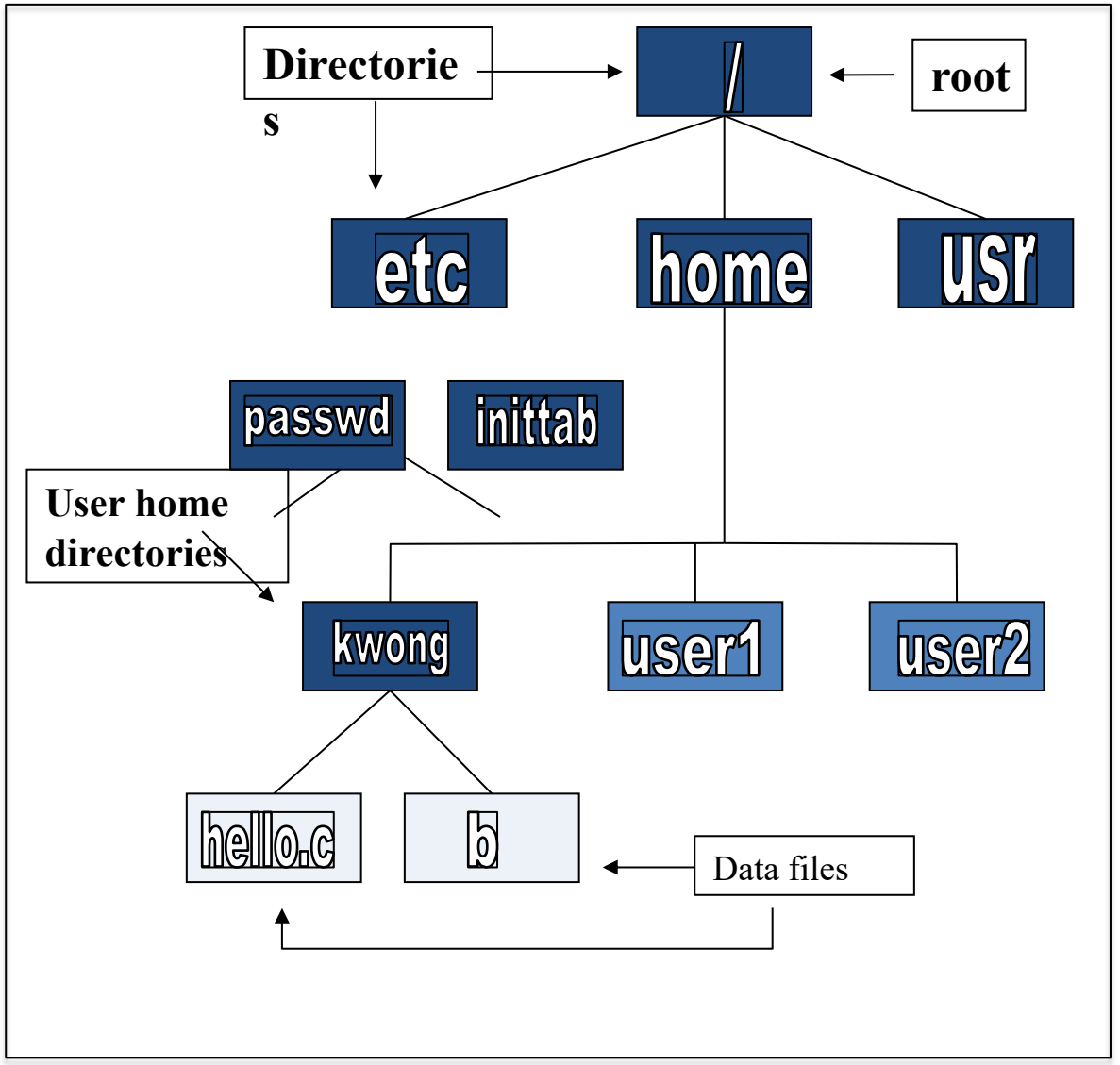
- ✓ Linux operation ? Skill level. Operating System, Software system, compiling
- ✓ In 70's, UNIX OS by Bell Lab, for main frame computer
- ✓ C is developed in 1972 by Dennis Ritchie at the Bell Laboratories for Unix OS
- ✓ In 80's, Microsoft's DOS, Apple MAC
- ✓ GNU project started by Richard Stallman in 1984 : free software, C compiler in 1991
- ✓ Linux Torvalds, a college sophomore, wrote the first Linux kernel in Sept. 1991 based on Minix developed by Anfrew Tanenbaum. UNIX on PC - LINUX
- ✓ www.linux .org, www.gnu.org

- ✓ Linux operating system in general
  - ✓ FILE, PATH, FILE MOD
- ✓ General overview Linux OS and terminal commands
  - ✓ file, program, executable program
  - ✓ cd , ls, mkdir, cp, mv , rm, xedit, gedit, env, path
- ✓ Tools and simple programming skills
- ✓ Compilers – gcc, g++
  - ✓ "gcc –o pexe ./prime.c" ; " ./pexe "



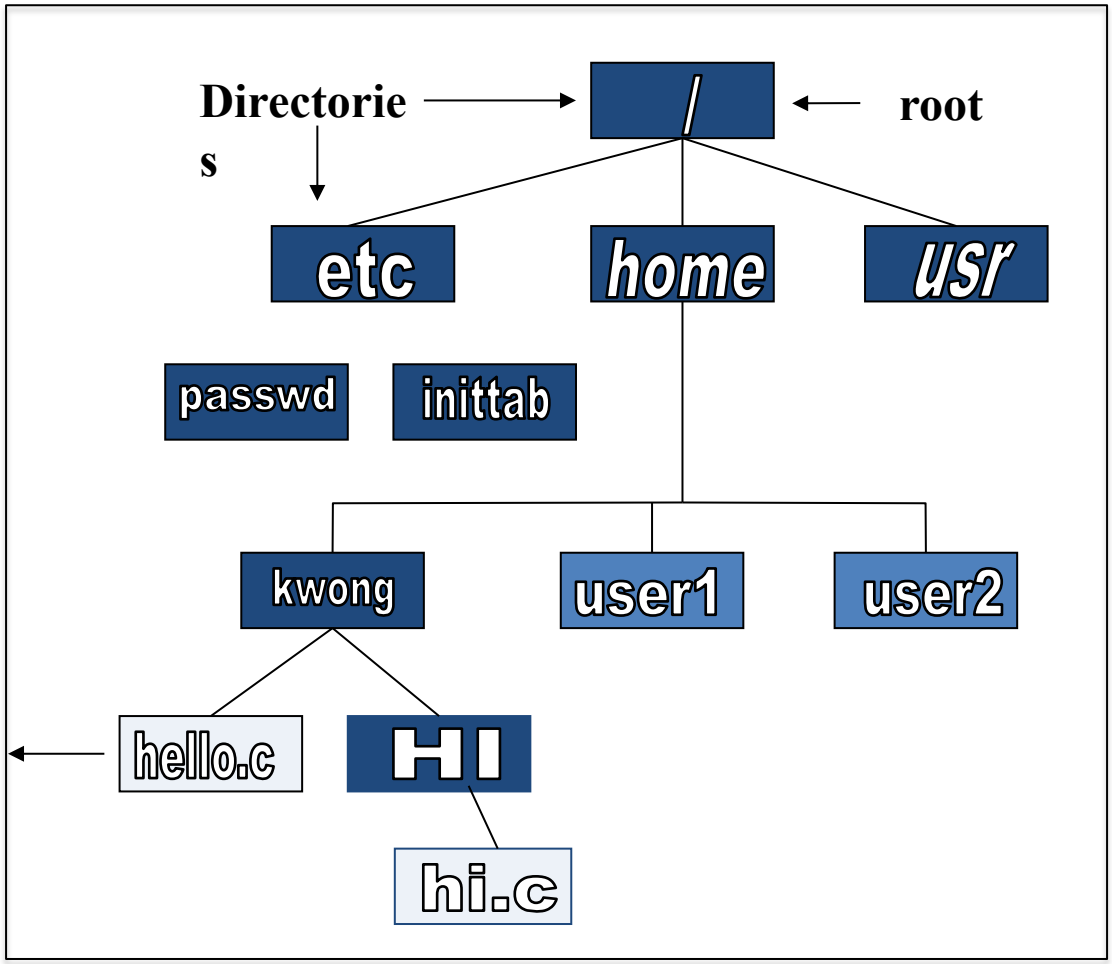
Linux File System

- Linux files are stored in a single rooted, hierarchical file system
  - Data files are stored in directories (folders)
  - Directories may be nested as deep as needed



Directory and Path : Absolute and Relative Path

```
$> cd
$>pwd
/home/kwong
$> ls
hello.c  HI
$>cd HI
$> ls
hi.c
$>pwd
/home/kwong/HI
$> cd ..
$> pwd
/home/kwong
```



- ✓ Absolute path - use pwd to find the address of the file, e.g. /home/kwong/CLASS/"file" in the CLASS directory"
- ✓ Relative path - use ./ to tell the computer the program is in the current directory, e.g use the ./"selected file in the current directory"

File Permissions : read (r), write (w), execute(x)

✓ The long version of a file listing (`ls -l`) will display the file permissions:

```
-rwxrwxr-x   1 kwong   kwong       5224 Dec 30 03:22 hexe
-rw-rw-r--   1 kwong   kwong        221 Dec 30 03:59 hello.c
-rw-rw-r--   1 kwong   kwong       1514 Dec 30 03:59 hello.s
drwxrwxr-x   7 kwong   kwong       1024 Dec 31 14:52 HI
```

Permissions Owner Group

-rwx rwx rwx (777) READ ~ 4 ; WRITE ~ 2 ; EXECUTE ~ 1 === total ~ 7

Other permissions
Group permissions
Owner permissions
Directory flag (d=directory; l=link)

```
chmod 755 file  # Owner=rwx Group=r-x Other=r-x
chmod 500 file2 # Owner=r-x Group=--- Other=---

chmod 644 file3 # Owner=rw- Group=r-- Other=r--
chmod +x  file  # Add execute permission to file for all
chmod o-r file  # Remove read permission for others
chmod a+w file  # Add write permission for everyone
```

Linux Commands

http://bootlin.com/doc/legacy/command-line/

LINUX COMMANDS CHEAT SHEET

SYSTEM

```
#uname -a          =>Displaylinux system information
#uname -r          =>isplay kernel release information
#uptime            =>Show how long the system has been running + load
#hostname          =>Show system host name
#hostname -i       =>Display the IP address of the host
#last reboot       =>Show system reboot history
#date              =>Show the current date and time
#cal               =>Show this month calendar
#w                 =>Display who is online
#whoami            =>Who you are logged in as
#finger user       =>Display information about user
```

HARDWARE

```
#dmesg             =>Detected hardware and boot messages
#cat /proc/cpuinfo =>CPU model
#cat /proc/meminfo =>Hardware memory
#cat /proc/interrupts =>Lists the number of interrupts per CPU per I/O device
#lshw              =>Displays information on hardware configuration of the
                     system
#lsblk             =>Displays block device related information in Linux
#free -m           =>Used and free memory (-m for MB)
#lspci -tv         =>Show PCI devices
#lsusb -tv         =>Show USB devices
#dmidecode         =>Show hardware info from the BIOS
#hdparm -i /dev/sda      =>Show info about disk sda
#hdparm -tT /dev/sda     =>Do a read speed test on disk sda
#badblocks -s /dev/sda   =>Test for unreadable blocks on disk sda
```

USERS

```
#id                =>Show the active user id with login and group
#last              =>Show last logins on the system
#who               =>Show who is logged on the system
#groupadd admin    =>Add group "admin"
#useradd -c "Sam Tomshi"   =>g admin -m sam #Create user "sam"
#userdel sam       =>Delete user sam
#adduser sam       =>Add user "sam"
#usermod           =>Modify user information
```

FILE COMMANDS

```
#ls –al            =>Display all information about files/ directories
#pwd               =>Show the path of current directory
#mkdir directory-name  =>Create a directory
#rm file-name      =>Delete file
#rm -r directory-nam   =>Delete directory recursively
#rm -f file-name   =>Forcefully remove file
#rm -rf directory-name  =>Forcefully remove directory recursively
#cp file1 file2    =>Copy file1 to file2
#cp -r dir1 dir2   =>Copy dir1 to dir2, create dir2 if it doesn't exist
#mv file1 file2    =>Rename source to dest / move source to directory
```

cont.

```
#ln –s /path/to/file-name link-name   #Create symbolic link to file-name
#touch file        =>Create or update file
#cat > file        =>Place standard input into file
#more file         =>Output contents of file
#head file         =>Output first 10 lines of file
#tail file         =>Output last 10 lines of file
#tail -f file      =>Output contents of file as it grows starting with the
                     last 10 lines
#gpg -c file       =>Encrypt file
#gpg file.gpg      =>Decrypt file
#wc                =>print the number of bytes, words, and lines in files
#xargs             =>Execute command lines from standard input
```

PROCESS RELATED

```
#ps                =>Display your currently active processes
#ps aux | grep 'telnet'  =>Find all process id related to telnet process
#pmap              =>Memory map of process
#top               =>Display all running processes
#killpid           =>Kill process with mentioned pid id
#killall proc      =>Kill all processes named proc
#pkill process-name =>Send signal to a process with its name
#bg                =>[stopped] background jobs
#fg                =>Bring the recent job to foreground
#fg n              =>Bring job n to the foreground
```

FILE PERMISSION RELATED

```
#chmod octal file-name   =>Change the permissions of file to octal
Example
#chmod 777 /data/test.c  =>Set rwx permission for owner,group,world
#chmod 755 /data/test.c  =>Set rwx permission for owner,rw for group
                           and world
#chown owner-user file   =>Change owner of the file
#chown owner-user:owner-group file-name   =>Change owner and group
                                            owner of the file
#chown owner-user:owner-group directory   =>Change owner and group
                                            owner of the directory
```

NETWORK

```
#ifconfig –a       =>Display all network ports and ip address
#ifconfig eth0     =>Display specific ethernet port
#ethtool eth0      =>Linux tool to show ethernet status
#mii-tool eth0     =>Linux tool to show ethernet status
#ping host         =>Send echo request to test connection
#whois domain      =>Get who is information for domain
#dig domain        =>Get DNS information for domain
#dig -x host       =>Reverse lookup host
#host google.com   =>Lookup DNS ip address for the name
#hostname –i       =>Lookup local ip address
#wget file         =>Download file
#netstat -tupl     =>List active connections to / from system
```

COMPRESSION / ARCHIVES

```
#tar cf home.tar home    =>Create tar named home.tar containing home/
#tar xf file.tar         =>Extract the files from file.tar
#tar czf file.tar.gz files  =>Create a tar with gzip compression
#gzip file               =>Compress file and renames it to file.gz
```

INSTALL PACKAGE

```
#rpm -i pkgname.rpm      =>Install rpm based package
#rpm -e pkgname          =>Remove package
```

INSTALL FROM SOURCE

```
#./configure
#make
#make install
```

SEARCH

```
#grep pattern files      =>Search for pattern in files
#grep -r pattern dir     =>Search recursively for pattern in dir
#locate file             =>Find all instances of file
#find /home/tom -name 'index*'   =>Find files names that start with "index"
#find / -size +10000k -print     =>Find files larger than 10000k in /home
```

LOGIN (SSH AND TELNET)

```
#ssh user@host           =>Connect to host as user
#ssh -p port user@host   =>Connect to host using specific port
#telnet host             =>Connect to the system using telnet port
```

FILE TRANSFER

```
scp
#scp file.txt server2:/tmp   =>Secure copy file.txt to remote host
                               /tmp folder
rsync
#rsync -a /home/apps /backup/   =>Synchronize source to destination
```

DISK USAGE

```
#df –h             =>Show free space on mounted filesystems
#df -i             =>Show free inodes on mounted filesystems
#fdisk -l          =>Show disks partitions sizes and types
#du -ah            =>Display disk usage in human readable form
#du -sh            =>Display total disk usage on the current directory
```

DIRECTORY TRAVERSE

```
#cd ..             =>To go up one level of the directory tree
#cd                =>Go to $HOME directory
#cd /test          =>Change to /test directory
```

MORE DETAILED : HTTP://LINOXIDE.COM/GUIDE/LINUX-COMMAND-SHELF.HTML

Linux Cheat Sheet

Python programming

- ✓ Python is a programming language, (mycode.py)
- ✓ Many python programs for specific applications (libraries, packages) have been written by the community developers and could be “imported” to any python codes, e.g. numpy, pandas, etc..
- ✓ Jupyter notebook is a web-based interactive computing interface (GUI) mainly to run python (mycode.ipynb). It can be considered as a package in python.
- ✓ Anaconda is a software distribution managing a large collection of python packages primarily for data sciences applications. It is a popular platform to run python.
- ✓ Google Colab is web-based interface running jupyter notebook on computers provided by Google. It is a computing platform (free computers!!)

Colab → jupyter notebook → python → Google computers

Anaconda → jupyter notebook → python → your computer

Python Basic

```
# This is a comment statement
import math
A=10
B=10
C= A*B
print (“ C”)
D = math.sqrt(C)
print (“D”)
```

```
#Timing Example
import time
time.time()
t0=time.time()
Let's test your typing speed!
t1=time.time()
t1-t0
```

Reference – what the exact description of how this function work

Python Basic – Control Flow, Timing

A. if, elif and else

```
if … :
    …
elif …:
    …
else:
    …
```

Example

```
a=5
b=7
c=8
d=4
if a<b and c>d:
    print('correct')
```

B. for loop : for i in range(N)

The command will loop from $i = 0$ to $i = N-1$.
range(N) is a sequence from 0, 1, 2, …, to N-1.

Example

```
total = 0
for i in range(4):
    total = total + i
    print(i)
print(total)
```

Example

```
for i in range(4):
    for j in range(4):
        if j>i:
            break
        print((i,j))
```

C. while loop , Example

```
er=10
count = 0
while er > 1:
    er = er - 0.5
    count = count + 1
print(er)
print(count)
```

Python Basic - List

In Python, a list is a sequence. It is enclosed in a pair of squared brackets '[]'. For example,
a = [1,2,3]

Elements are numbered from '0'. To call for a particular element, we give the index in square bracket:
a[0] = 1
a[1] = 2
a[2] = 3

To add an element, we use the 'append' function

a.append(4), which gives
a = [1,2,3,4]

To insert an element at a specific location, we use the 'insert' function

a.insert(1,4), which gives

a = [1,4,2,3,4]

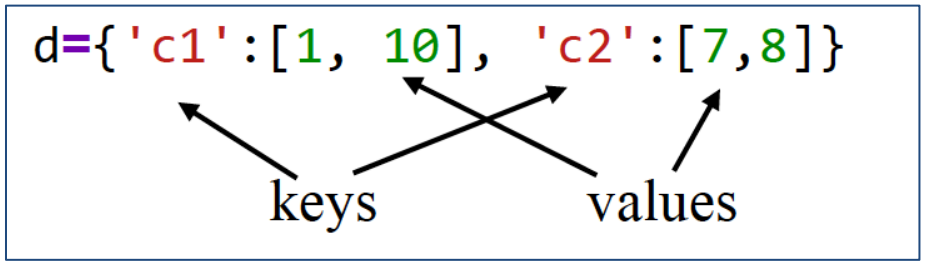
That is, the entry '4' is added into the location with index '1', in other words, the second entry.

Datatype can be different in a list and it is not necessarily numbers.
If we have two lists, we can easily concatenate (combine) the two lists by the '+' sign. Note that this is not numerical addition.

```
In [8]: a
Out[8]: [1, 4, 2, 3, 4]

In [9]: b=[1,3,5,7,9]

In [10]: a+b
Out[10]: [1, 4, 2, 3, 4, 1, 3, 5, 7, 9]
```

Python Basic - Dictionary

Dictionary is an associative array with key and value. To define a dictionary, we enclose the content with a pair of curly brackets '{}'. The names of the keys are enclosed in ''. We can get the values under a particular key by dict['key']




```
In [12]: d['c1']
Out[12]: [1, 10]

In [13]: d['c2']
Out[13]: [7, 8]
```

The keys and values in a dictionary can be obtained by listing them out.

```
In [14]: list(d.keys())
Out[14]: ['c1', 'c2']

In [15]: list(d.values())
Out[15]: [[1, 10], [7, 8]]
```

Entries can easily be added by defining a new key: d['c3']=['Hello','World']

Then, the dictionary is updated to : d={'c1': [1, 10], 'c2': [7, 8], 'c3': ['Hello', 'World']}

On the other hand, we delete entries by 'del' : **del d['c1']**
d={'c2': [7, 8], 'c3': ['Hello', 'World']}

Python – Numpy, Arrays – LA - BLAS

NumPy stands for Numerical Python. It is the package for numerical computation in Python. We will learn about : Arrays, Matrix Operations, Universal Functions, Random number, Statistics, etc. To use the NumPy package, we first import it:

```
import numpy as np
```

We define an array by specifying that it is an NumPy array

```
a=np.array([1,2,3])
```

As in C++ , the indexing starts from 0. That is,
a[0] = 1, a[1]=2 and a[2]=3. a[3] is undefined.

Similarly, we can define a matrix by adding '[]'. For example, to define the following matrix

$$b = \begin{bmatrix} 1 & 0 & -5, \\ -2.5. & 7.3 & 4.6 \end{bmatrix}$$; in Python, we write :

```
b=np.array( [  [1,0,-5] , [-2.5, 7.3, 4.6]  ] )
```

Indexing starts from the outer brackets: b[0,2] = –5, b[1,1] = 7.3.

Python Numpy – Dimension

The dimension or size of an array is given by the 'shape' function.
To get the number of rows in b, we ask for the first entry in its shape

b.shape[0].

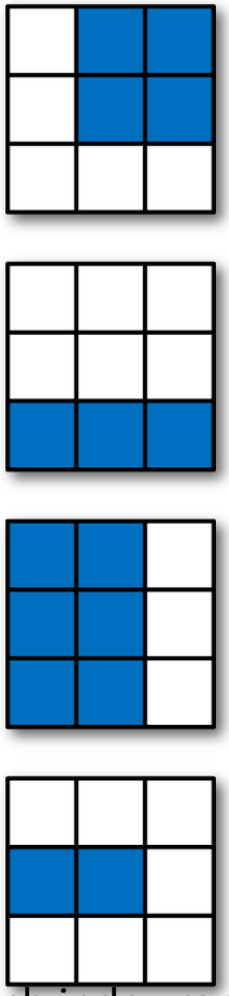
Similarly, the number of columns in b is given by

b.shape[1]

```
In [6]: a.shape

Out[6]: (3,)

In [17]: b.shape

Out[17]: (2, 3)
```

Python Numpy – Indexing and Slicing, Example

```
In [10]: b
Out[10]: array([[ 1. ,  0. , -5. ],
                [-2.5,  7.3,  4.6]])

In [8]: b[1:,:2]
Out[8]: array([[-2.5,  7.3]])

In [9]: b[:2,1:]
Out[9]: array([[ 0. , -5. ],
               [ 7.3,  4.6]])

In [11]: b[:,1:]
Out[11]: array([[ 0. , -5. ],
                [ 7.3,  4.6]])
```



Expression	Shape
arr[:2, 1:]	(2, 2)
arr[2]	(3,)
arr[2, :]	(3,)
arr[2:, :]	(1, 3)
arr[:, :2]	(3, 2)
arr[1, :2]	(2,)
arr[1:2, :2]	(1, 2)

Form : Shape of an array; running indices, each index represents a dimension
Extraction and description of an array - functions to get to what you need!!

Numpy – Reshape

'reshape' function allows us to change the dimension of an array. A 1-dimensional array can be reshaped to a 2-dimensiaonl matrix. The reshape function is useful when we deal with image data. An image is usually described as a matrix (2d). By reshape, we can turn it into a row vector. We can then stack all image data into one large matrix. A lazy way to specify the dimension is '-1' when the dimension is inferred from the data.

Example

```
np.arange(15).reshape(3,5)
```

gives

```
array( [ [ 0, 1, 2, 3, 4],
[ 5, 6, 7, 8, 9],
[10, 11, 12, 13, 14]  ] )

M=np.array ([[1,2],[11,12]] )
M.reshape(1,4) or M.reshape(1,-1)
```

gives

```
array( [[ 1, 2, 11, 12]] )
```

Numpy – Arithmetic

Basics operations are *elementwise*

+ addition
- subtraction
* multiplication
/ division
** power

For example, to take the square of every element in an array u:

```
u**2
```

The square root can be computed in two ways

```
v**0.5
```

or

```
np.sqrt(v)
```

Note that, you can sum over all entries in an array by

```
np.sum(v)
```

To get the real part of an array, we have

```
v.real
```

Empty Array

'empty' creates an array without initializing its values. : np.empty((3,2))

Zero Array

We can define a zero matrix with all entries equal to 0, : d=np.zeros((2,4))

Datatype

The datatype of an array is given by

```
b.dtype
```

The datatype of an array can be converted to another type

```
b.astype(np.int)
```

Say for example, we have an array of true or false, we wish to change the values to 0(false) and 1(true). Example

```
v_boolean=np.array([True,False,False])
v_int = v_boolean.astype(np.int)
v_intequals
array([1, 0, 0])
```

Matrix Operations

A. Matrix Multiplication

To multiply two matrices *x* and *y*, we either use the 'dot' function

```
np.dot(x,y)
```

or simply '@'

```
x @ y
```

B. Transpose :

To compute the transpose of a matrix M,

```
M.T
```

C. Norm

More functions related to linear algebra can be obtained from the 'linalg' library in NumPy. For instance, we compute the norm of a vector by the norm function. Various types of norm can be computed. If we do not specify the choice of norm, the program return the default Euclidean *l*2-norm.

```
from numpy.linalg import norm
norm(v)
```

Universal Functions

Common functions such as the exponential function, logarithmic function, etc. can be computed by the usual command.

```
np.exp(x)
np.log(x)
np.abs(x)
np.sign(x)
np.sqrt(x)
```

Random Number

A random number can be generated by

```
np.random.rand()
```

The above command yields a number draw from a uniform distribution.

```
np.random.randn(5)
```

The above command yields an array of length 5 where each entry is drawn from a normal distribution with mean 0 and standard deviation 1.

Basic Statistics : Mean, Standard Deviation, maximum

Suppose that we have a 2 by 3 matrix,

```
A=np.array( [[1,0,2],[3,5,6]] )
```

the mean of all the entries is given by

```
np.mean(A)
```

or simply

```
A.mean()
```

We can compute the mean along rows or columns by specifying the axis. By specifying 'axis=i', summation is taken w.r.t the *i*th-index. In the above example,

```
A.mean(axis=0)
```

yields the means of the three columns,

i.e. `array([2. , 2.5, 4. ])`

whereas

```
A.mean(axis=1)
```

gives the means of the two rows

i.e. `array([1., 4.66666667])`

Similarly, we can compute the standard deviation and maximum

```
A.std()
A.max()
```

Python DataFrame, Pandas

Pandas is a library in Python for handling data described in tabular form. While there are many functions in pandas, we will just give a brief introduction here. In this chapter, we will learn about DataFrame, importing data, importing data from libraries

```
import pandas as pd
```

DataFrame is a table of data. The rows are called 'index' and columns are simply referred to as 'columns'.

```
from pandas import DataFrame
```

To define a DataFrame, we need to define three objects: values, index, columns; Example

```
df = pd.DataFrame(np.arange(16).reshape(4,4), index=[1,2,3,4],columns=['a','b','c','d'])
```

	a	b	c	d
1	0	1	2	3
2	4	5	6	7
3	8	9	10	11
4	12	13	14	15

Python DataFrame, Pandas

A dictionary stores data under specific keys. We can easily define a DataFrame from a dictionary where keys are turned into columns. Suppose that we define the following dictionary:

```
data = {'Distance':[10,0.5,2,6.2],'Age':[5,40,30,10],
'Area':[100,200,500,150],'Price':[20,15,75,34]}
```

A DataFrame can be defined from the above dictionary as a well-organized and easy to read table

```
df = pd.DataFrame(data)
```

	Distance	Age	Area	Price
0	10.0	5	100	20
1	0.5	40	200	15
2	2.0	30	500	75
3	6.2	10	150	34

Pandas : Importing Data From Files

In application, the data is usually given in some files of standard format, say, excel, csv, etc. The data can be imported into Python through the 'read' function in pandas.

```
df = pd.read_csv('filename.csv',sep=';')
df = pd.read_csv('filename.csv',delimiter='\t')
```

Notice that the file should be saved in the same folder as the Jupyter notebook. The regular expression for separating the data is specified in 'sep' or 'delimiter':

',' ';' '\t' tab

Example (wine)

```
df = pd.read_csv('winequality-red.csv',sep=';')
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000

Let's take a look into the 'iris' dataset. The dataset is a classic classification problem where we determine the species of a given iris based on some physical parameters. We first import the data set.

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()
```

By checking the keys in the data set, we understand the major components in the data set.

```
iris_dataset.keys()
```

gives

```
dict_keys(['data', 'target', 'target_names',
'DESCR', 'feature_names', 'filename'])
```

where

'DESCR' gives the description of the dataset,

'target_names' gives the names of the species,

'target' stores the target values (label the species in terms of 0,1,2) of all the given data,

'feature_names' gives the descriptions of the features (sepal length, sepal width, petal length, pedal width),

'data' contains the feature values for the samples.

We can create a DataFrame of the data

```
iris_dataframe =
pd.DataFrame(iris_dataset['data'],columns=ir
is_dataset.feature_names)
```

where the data part is given by the values under the key 'data' and we name the columns with the feature names.

The defined DataFrame gives a clear table of the data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

There are in total 150 samples.

1. Write a python code to compute C=C+A x B and plot a curve of the FLOPS against the matrix size N when the computation is done on a CPU. Do the same on the GPU.
2. 2 * (N^3) FLOP / time = ? FLOPS, 2 * 5 *5 * 5 /7.4 = 33.8 GFLOPS
3. Run the same problem again using single precision.
4. Repeat question #5 using R

```
[18] import numpy as np

     A = np.random.rand(5000, 5000).astype('float64')
     B = np.random.rand(5000, 5000).astype('float64')

     %timeit np.dot(A, B)
```

```
1 loop, best of 3: 7.39 s per loop
```

```
%%R
library(dplyr)
A<-matrix(runif(25000000),nrow=5000)
B<-matrix(runif(25000000),nrow=5000)
system.time(C<-A%*%B)
```

```
   user  system elapsed
 15.449   0.025   7.840
```

LAB 2
Problem 4

Write a python code to compute C=AXB and plot a curve of the FLOPS against the matrix size N (take N=2000, 4000, 6000) using single precision when the computation is done on a CPU. DO the same on the GPU

```
import numpy as np
import time

A = np.random.rand(2000, 2000).astype('float32')
B = np.random.rand(2000, 2000).astype('float32')
%timeit np.dot(A,B)
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm2000 = timeend - timestt
gf2000 = 2*2*2*2/tmm2000
print('time = ',tmm2000)
print('GFLOPS = ', gf2000)

A = np.random.rand(4000, 4000).astype('float32')
B = np.random.rand(4000, 4000).astype('float32')
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm4000 = timeend - timestt
gf4000 = 2*4*4*4/tmm4000
print('time = ',tmm4000)
print('GFLOPS = ', gf4000)
```

```
A = np.random.rand(6000, 6000).astype('float32')
B = np.random.rand(6000, 6000).astype('float32')
timestt = time.time()
C=np.dot(A,B)
timeend = time.time()
tmm6000 = timeend - timestt
gf6000 = 2*6*6*6/tmm6000
print('time = ',tmm6000)
print('GFLOPS = ', gf6000)

import torch
A = torch.randn(6000, 6000).cuda()
B = torch.randn(6000, 6000).cuda()
timestt = time.time()
C=torch.matmul(A,B)
timeend = time.time()
tmm6000 = timeend - timestt
gf6000 = 2*6*6*6/tmm6000
print('time = ',tmm6000)
print('GFLOPS = ', gf6000)
```

```
1 loop, best of 5: 228 ms per loop
time =  0.2360849380493164
GFLOPS =  67.7722184744277
time =  1.8578176498413086
GFLOPS =  68.8980428251037
time =  6.155170440673828
GFLOPS =  70.18489644824643
```

```
1 loop, best of 5: 214 ms per loop
time =  0.22870469093322754
GFLOPS =  69.95921218192831
time =  1.8288803100585938
GFLOPS =  69.98817762759944
time =  6.09820103645324 7
GFLOPS =  70.84056386754575
```

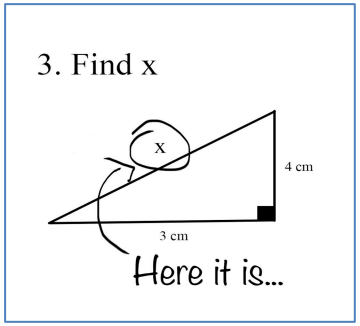
```
time =  6.029452800750732
GFLOPS =  71.6482928510878
time =  0.06104087829589844
GFLOPS =  7077.224510202169
```

Computational Ecosystem

- **Advance Computing system**
  - **Top500, FLOPS and Performance**
  - **DOE Exascale Road Map**
  - **NSF infrastructure**
  - **Desktop, Accelerators, GPUs**
  - **Google COLAB**
- **Predictive Simulation Science**
  - **Equation based simulation sciences**
  - **Mathematics Essentials, Calculus**
  - **Linear Algebra, BLAS**

- **Computer Sciences and Software tools**
  - **Linux OS**
  - **Computational thinking**
  - **Workflow**
  - **Languages**
- **Data Intensive Sciences**
  - **Statistical Learning**
  - **Data mining**
  - **Deep Learning**
  - **Framework**

Dealing with the Knowns (reducible) and Unknowns (irreducible) Uncertainty Quantification – Data Analytics



"As we know there are known knowns.

There are things we know we know.

We also know there are known unknowns.

That is to say, we know there are some things we do not know.

But there are also unknown unknowns.

The ones we don't know we don't know," (?)

Given enough data, can we find the unknowns and predict the knowns?

However, the more you know, the harder it is to find the right solution

Big Data → Deep Learning → Deep Neural Network (DNN)

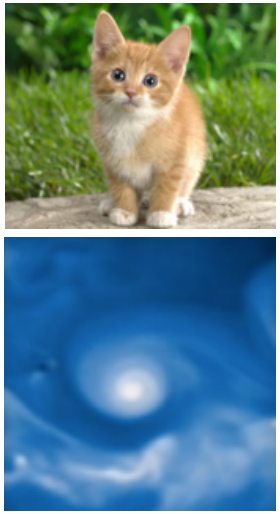
Augmented Modeling Cycle and Emergent Technology
Compute-intensive and Data Driven Model

Big Science (Physics, Chem, Biology, Social, etc.)
+ Big Data (Surveys, Historical, Geo, Web, etc.)
→ Equations + Statistics + AI
→ Learn the hidden pattern and behavior
Predict or determine new entities based on the established model

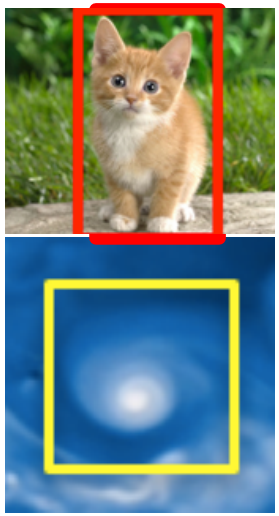
A growing field with evolving technology
Deterministic and Statistical Modeling →
Deep Learning → Supercomputing (GPU) →
Real Time Applications

Data Driven Computing, Deep Learning

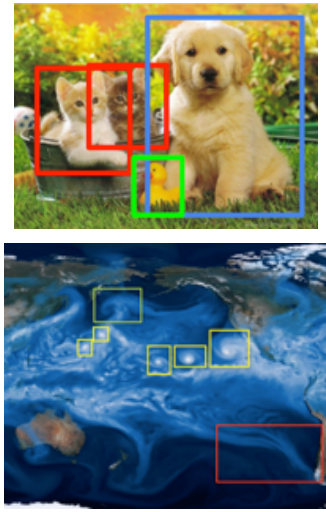
Classification



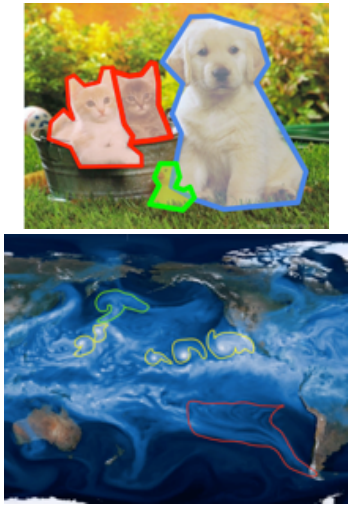
Classification + Localization



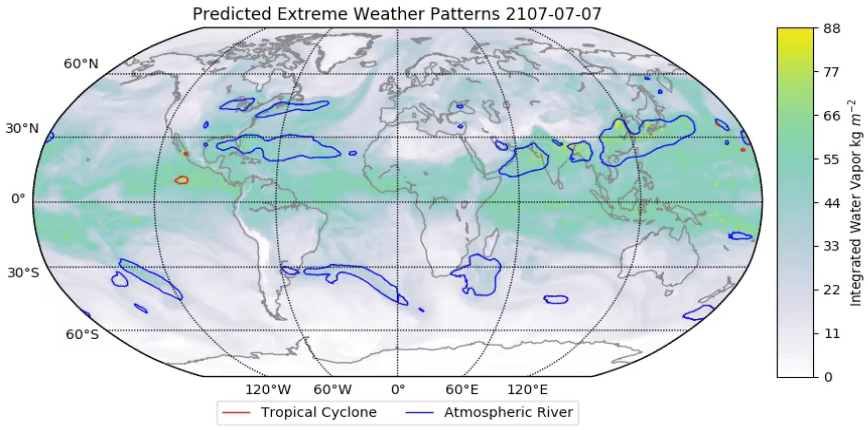
Object Detection



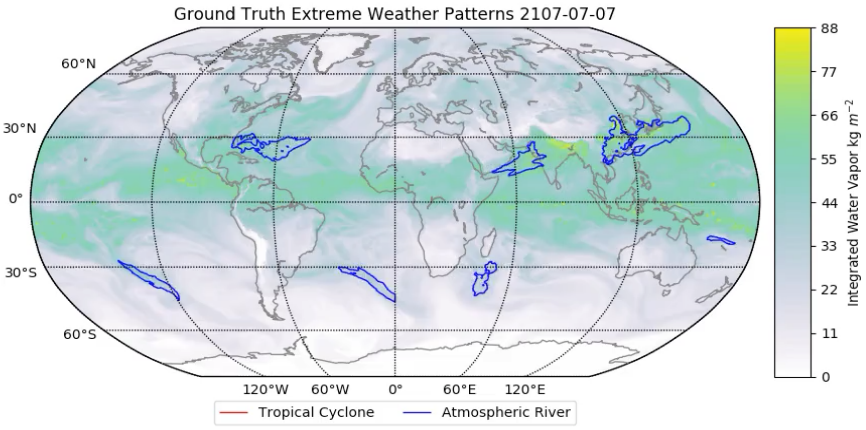
Instance Segmentation



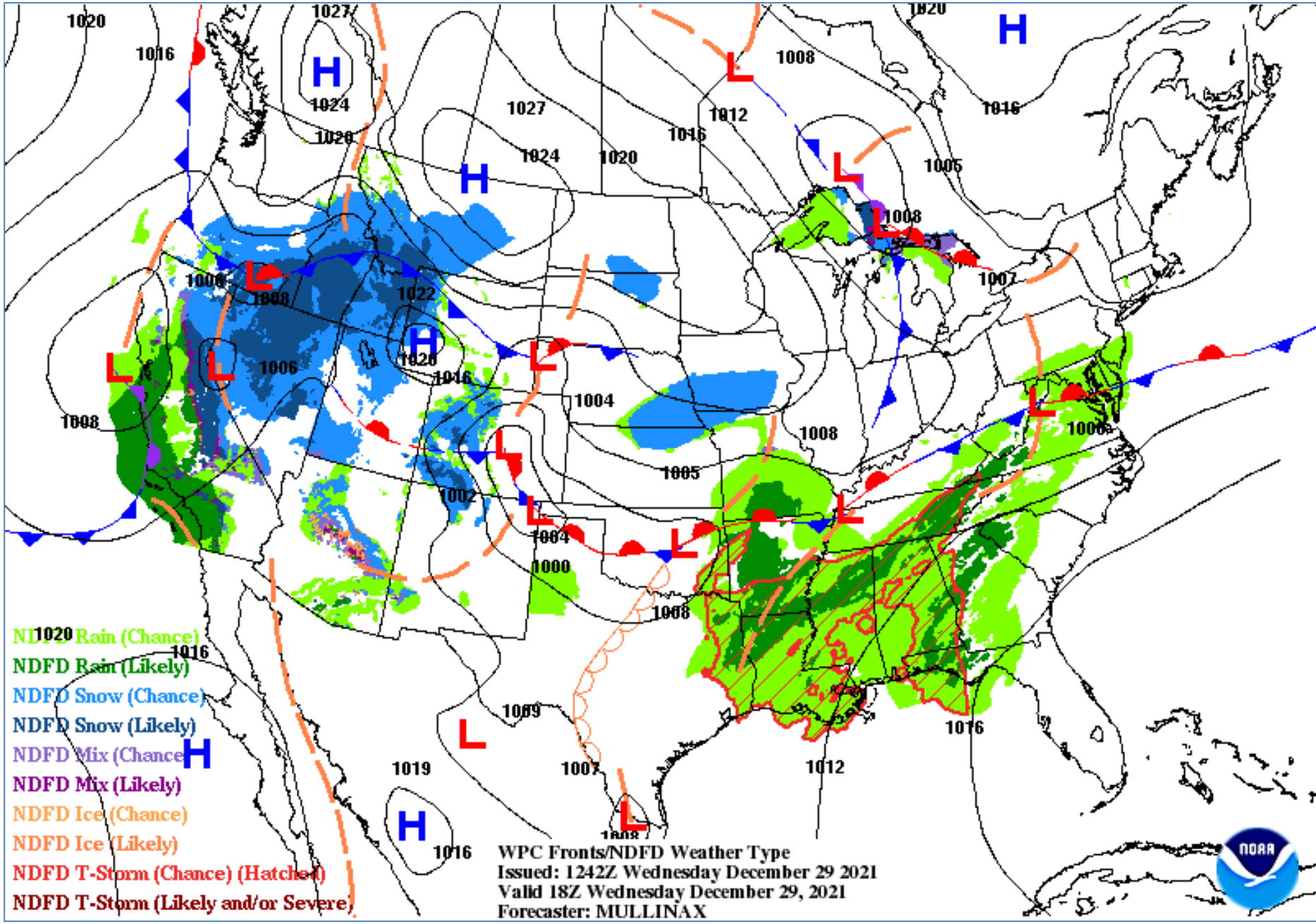
Predicted Extreme Weather



Ground Truth Extreme Weather



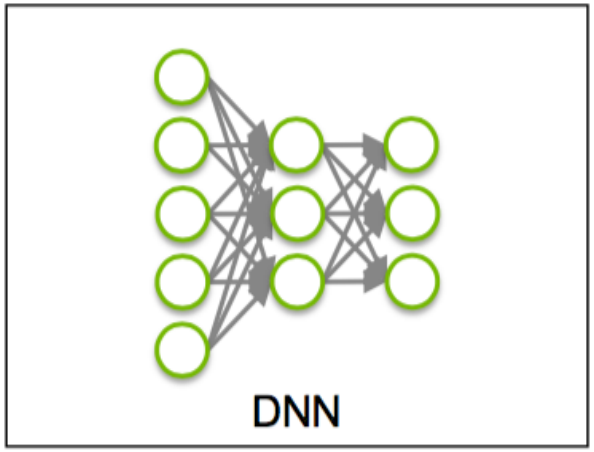
Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, Michael Houston

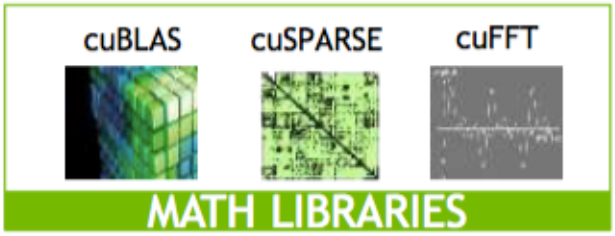

NDFD Rain (Chance)
NDFD Rain (Likely)
NDFD Snow (Chance)
NDFD Snow (Likely)
NDFD Mix (Chance)
NDFD Mix (Likely)
NDFD Ice (Chance)
NDFD Ice (Likely)
NDFD T-Storm (Chance) (Hatched)
NDFD T-Storm (Likely and/or Severe)
WPC Fronts/NDFD Weather Type
Issued: 1242Z Wednesday December 29 2021
Valid 18Z Wednesday December 29, 2021
Forecaster: MULLINAX

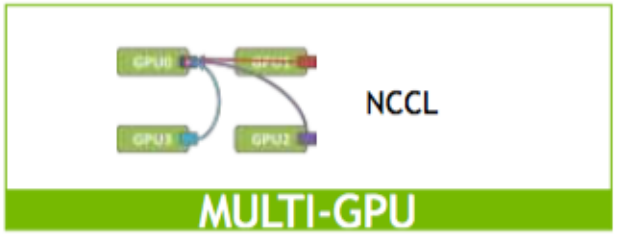
Machine Learning – GPU acceleration

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY



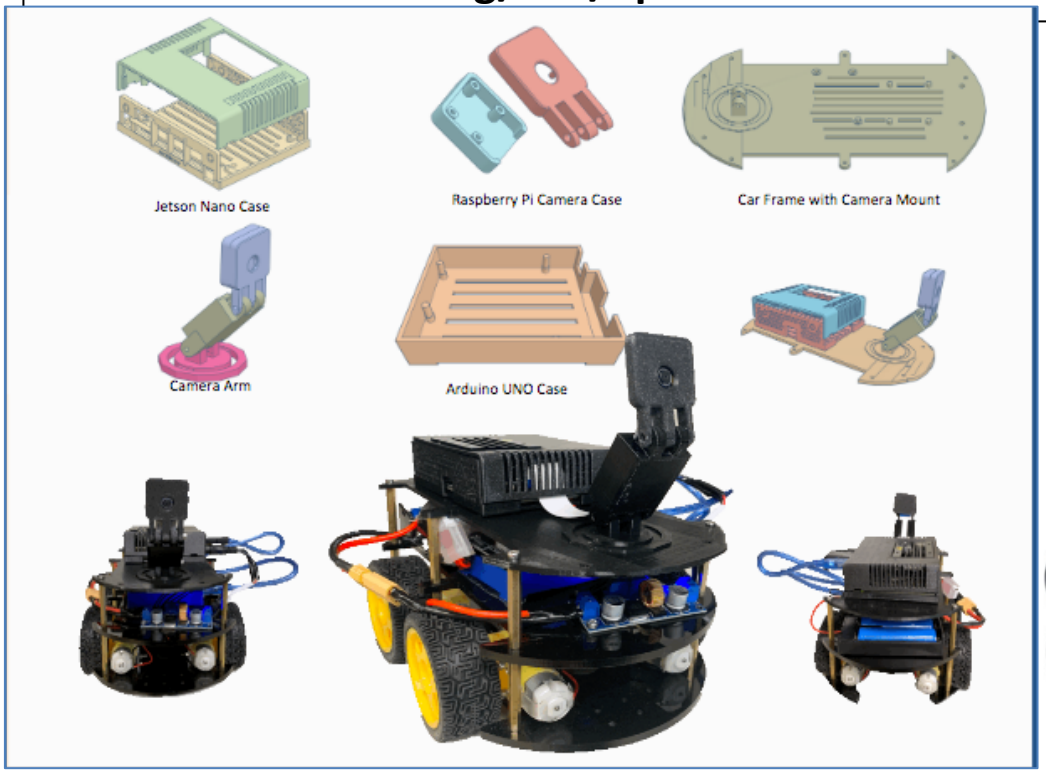


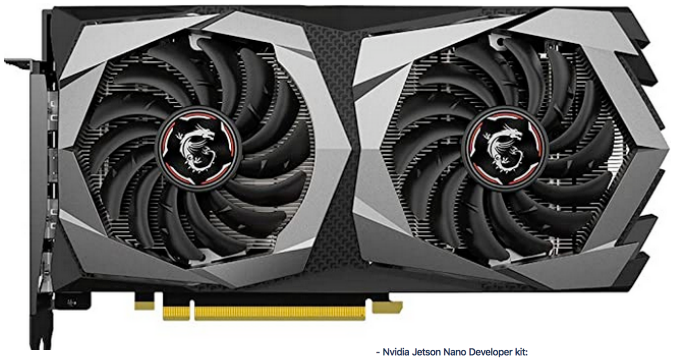


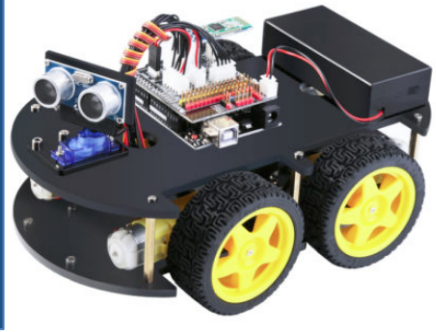
Source captured from - Julie Bernauer – HPC Advisory Council Stanford Tutorial – 2017/02/07

Desktop Computers and GPU

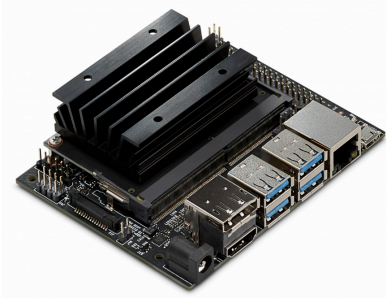
- ✓ **Computer Access, needs of each team, Desktop, laptop, Jetson Nano, Car**
- ✓ **Desktops configuration and deliveries, mid September**
- ✓ **Dell Server Box 12 cores, 48G Ram, 1TB Disk, 1650 super GPU card**
- ✓ **Ubuntu OS, anaconda individual, https://www.anaconda.com/products/individual**
- ✓ **Download, Linux, 64-Bit (x86) Installer (550 MB), python 3.8**
- ✓ **Jetson Nano card and car kit, late October**
- ✓ **www.bitbucket.org/cfdl/opendnnwheel**







- Nvidia Jetson Nano Developer kit:

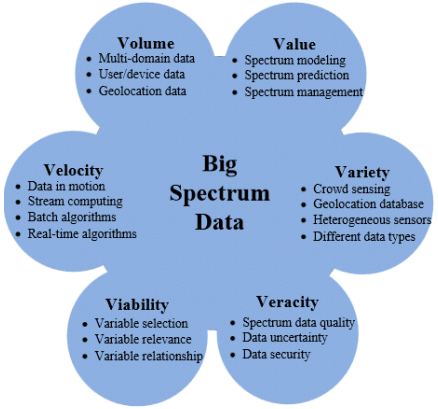


Big Data Predictive Model

- ✓ **A collection of large data sets that are asymmetric or too large to be processed by traditional tools. Often the data sets are noisy and heterogeneous but in general could be co-related to some significant events.**

Big Data Characterized by

- *Volume*
  - — How much data
- *Velocity*
  - — The speed at which data arrives and the speed with which decisions based on it must be made
- *Variety*
  - — Heterogeneity of storage platforms, data types, representation, semantic interpretation, and security classification or other distribution limitations
- *Veracity*
  - — How trustworthy is the data, what is its uncertainty, and what is the error associated with it
- *Value*
  - — What is the data worth



- ✓ **Challenges include storage, classification, mining, sharing, visualization..**
- ✓ **Need capacity, infrastructure, domain knowledge + compute , CS, Math..**

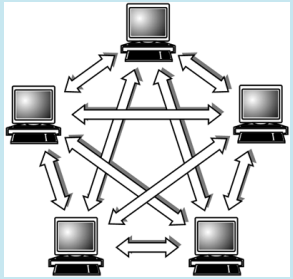
Programming Models & Tools Ecosystem:

Data sciences is inter-disciplinary,
need community effort to coordinate creation of software tools for various applications

- Flat file, Excel, CVS
- Database, SQL,
- Distributed DD, HDFS
- Large graph, matrix, SVD
- Storage, I/O, network
- Sensors, big instruments
- Data Mining, searching, compression, neural network, deep learning, smart detection, predictive models, visualization
- Images (picture, neutron, thermal, x-ray…), spatial temporal data, noise, signal, voice, smell, ….
- Healthcare, social, politics, science, finance, agriculture, entertainment, geographic, transportation …

Milestones –Capacity (Big data)

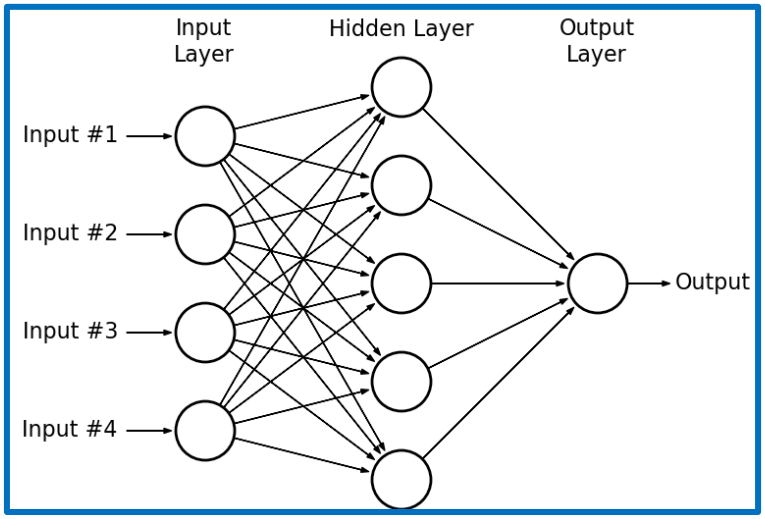
- 1973 – Internet was “officially” named
- 1990s Internet widely used
- 1993 Mosaic (NCSA), web browser.. netscape, IE, Mozzilla, Firefox..
- 1995- Google, Amazon
- 1996 – IBM Deep Blue Chess machine, first Terascale, ASCII RED
- 1999 – Grid Computing
- 2000 – Baidu
- 2004 – Facebook, MapReduce
- 2005 – Hadoop
- 2006 deep learning, Geoffrey Hinton, Neural Computing
- Clouds, machine learning framework, GPU
- 2015 – NSCI , DeepMind
- 2015 – NSF - Big Data Hub, Alpha Go, 2016 beat Lee Sedol
- DNN + Big Data Analytics + supercomputers + sensors + applications + more
- Digital.gov, wiki page, AlphaFold. ……

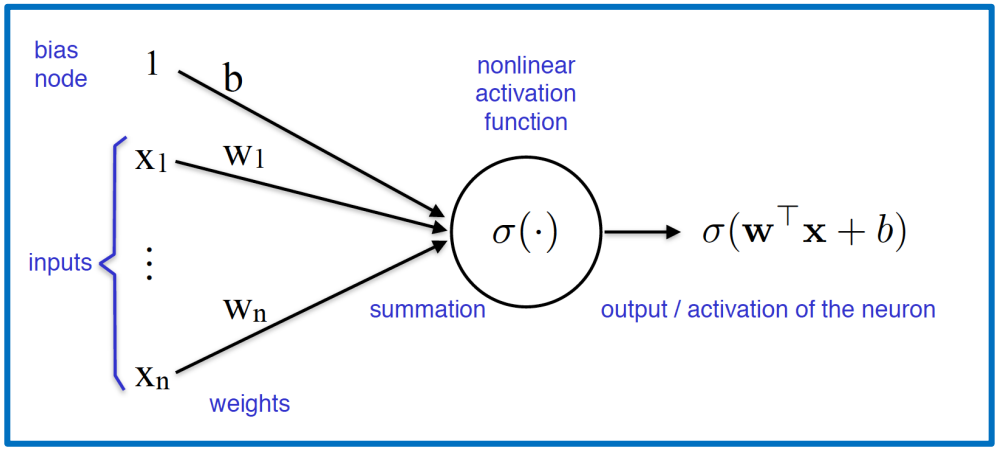


References

- ✓ An Introduction to Statistical Learning with Applications in R, by *Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani*. - http://wwwbcf.usc.edu/~gareth/ISL/
- ✓ http://cs231n.stanford.edu/
- ✓ Machine Learning course by Andrew Ng, https://www.coursera.org/learn/machine-learning
- ✓ MIT Deep Learning course, https://deeplearning.mit.edu/,
- ✓ *Deep Learning by* Ian Goodfellow and Yoshua Bengio and Aaron Courville - https://www.deeplearningbook.org/
- ✓ Deep Learning for Computer Vision by Justin Johnson, https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html
- ✓ *TensorFlow Tutorial* - https://www.tensorflow.org/tutorials
- ✓ Applications of Deep Neural Networks by Jeff Heaton T81-558 - https://github.com/jeffheaton/t81_558_deep_learning
- ✓ *TensorFlow Tutorial* - *https://www.tensorflow.org/tutorials*
- ✓ Python Tutorial for Beginner by Mosh, https://www.youtube.com/watch?v=kqtD5dpn9C8

NN Modeling





- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Inputs represent a set of vectors containing weights (w) and bias (b). They are the sets of parameters to be determined.
- ✓ Many nodes form a **neural layer**, **links** connect layers together, defining a NN model.
- ✓ Activation function (f or σ), is generally a nonlinear data operator which facilitates identification of complex features.

TensorFlow 2.0 – NLA Tensor

TensorFlow 1.X requires users to manually stitch together an abstract syntax tree (the graph) by making tf.* API calls.

Since these graphs are data structures, they can be saved, run, and restored all without the original Python code.

It then requires users to manually compile the abstract syntax tree by passing a set of output tensors and input tensors to a session.run() call.

TensorFlow 2.0 executes eagerly (like Python normally does) and in 2.0, graphs and sessions should feel like implementation details. This means TensorFlow operations are executed by Python, operation by operation, and returning results back to Python. It is also easy to debug. For some users, you may never need or want to leave Python. TF2.0 makes it more like numpy

Running TensorFlow op-by-op in Python prevents a host of accelerations otherwise available. If you can extract tensor computations from Python, you can make them into a *graph*.

A session.run() call is almost like a function call: You specify the inputs and the function to be called, and you get back a set of outputs.

In TensorFlow 2.0, you can decorate a Python function using tf.function() to mark it for JIT compilation so that TensorFlow runs it as a single graph (Functions 2.0 RFC).

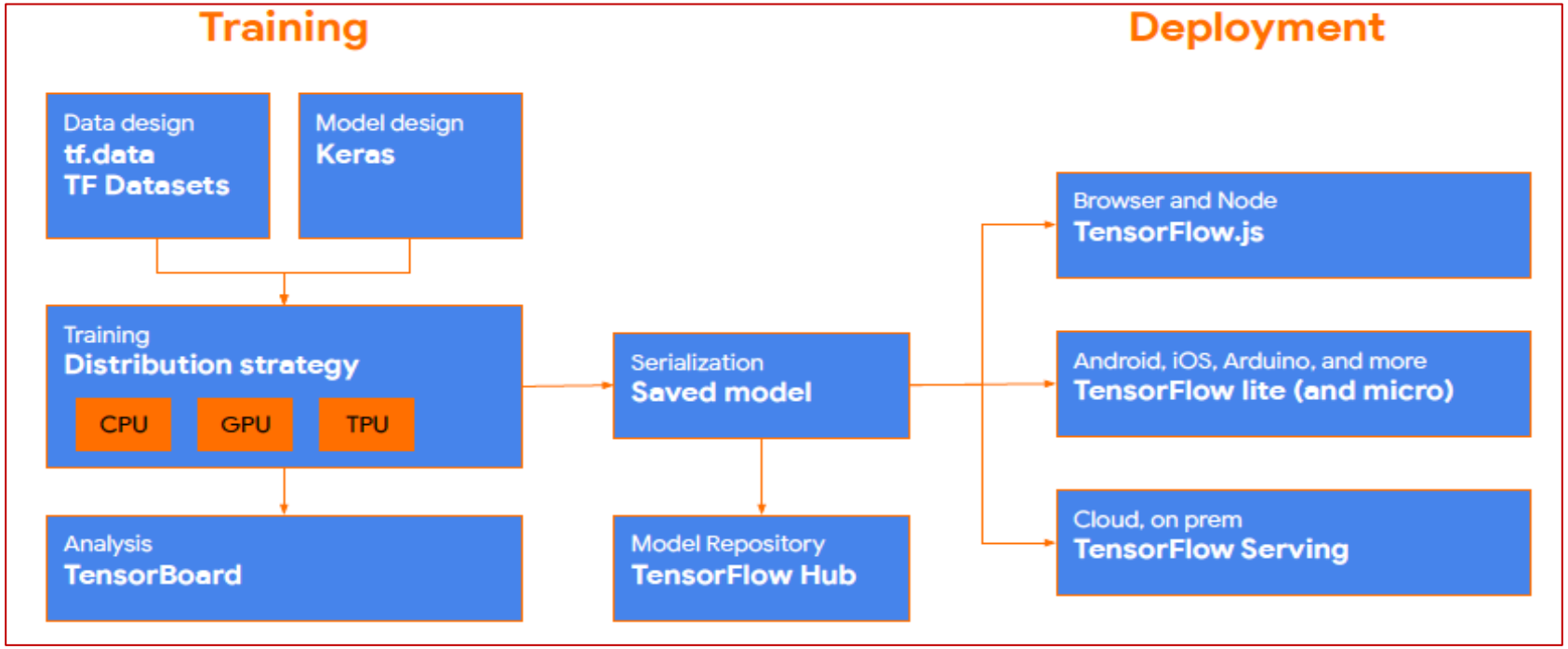
A common usage pattern in TensorFlow 1.X was the "kitchen sink" strategy, where the union of all possible computations was preemptively laid out, and then selected tensors were evaluated via session.run().

In TensorFlow 2.0, users should refactor their code into smaller functions that are called as needed. In general, it's not necessary to decorate each of these smaller functions with tf.function; only use tf.function to decorate high-level computations - for example, one step of training or the forward pass of your model.

TensorFlow 2.0 : model treatment

- ✓ Run like python
- ✓ Easier Tensor expression and reshape,
- ✓ Eager execution
- ✓ graph representation run on function,
- ✓ Use case for user
- ✓ Start with sequential API stack
- ✓ Scale up to function API or subclass
- ✓ Keros API for more
- ✓ Function API, DAG

- ✓ A wonderful thing about Deep Learning: ability to mix data types (text, Dense, images, time series, structured data) in a single model, treat Image to a vector and a question to a vector
- ✓ Run different application the same way in TF
- ✓ TF provides interfaces to other software platforms

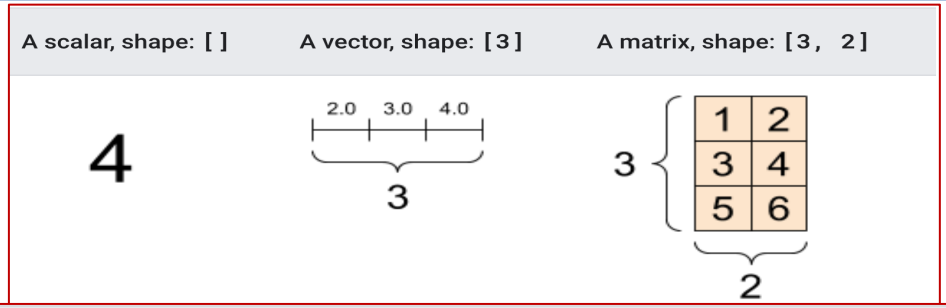


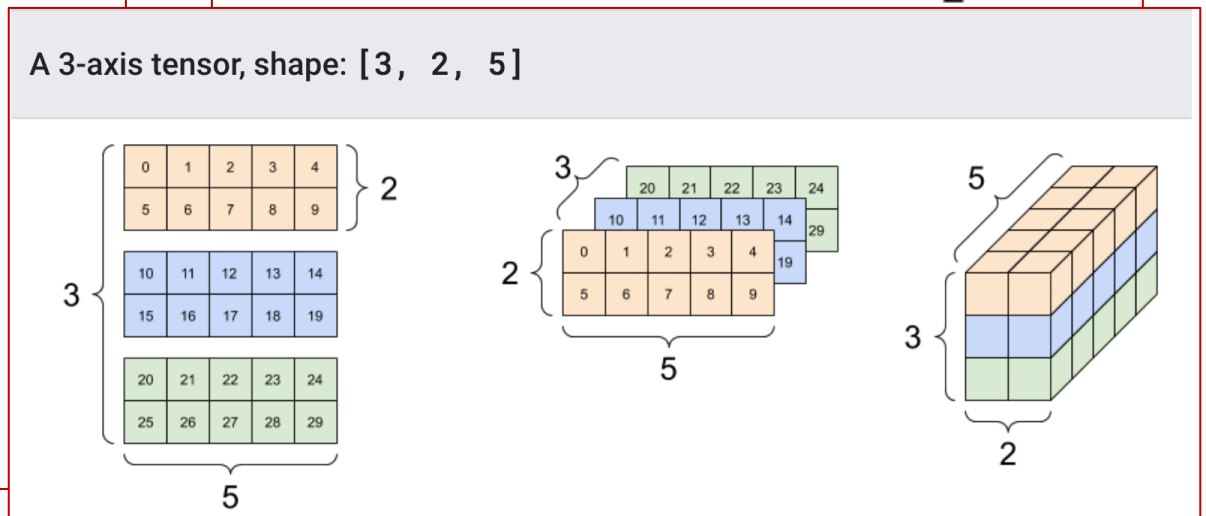
TensorFlow : Tensor

Tensors are multi-dimensional arrays with a uniform type (called a dtype).

```
import tensorflow as tf
import numpy as np
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)

rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)

rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)

rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])
print(rank_3_tensor)
```

```
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
tf.Tensor( [[1. 2.] [3. 4.] [5. 6.]], shape=(3, 2), dtype=float16)
tf.Tensor( [[[ 0 1 2 3 4] [ 5 6 7 8 9]] [[10 11 12 13 14] [15 16
17 18 19]] [[20 21 22 23 24] [25 26 27 28 29]]], shape=(3, 2,
5), dtype=int32)
```

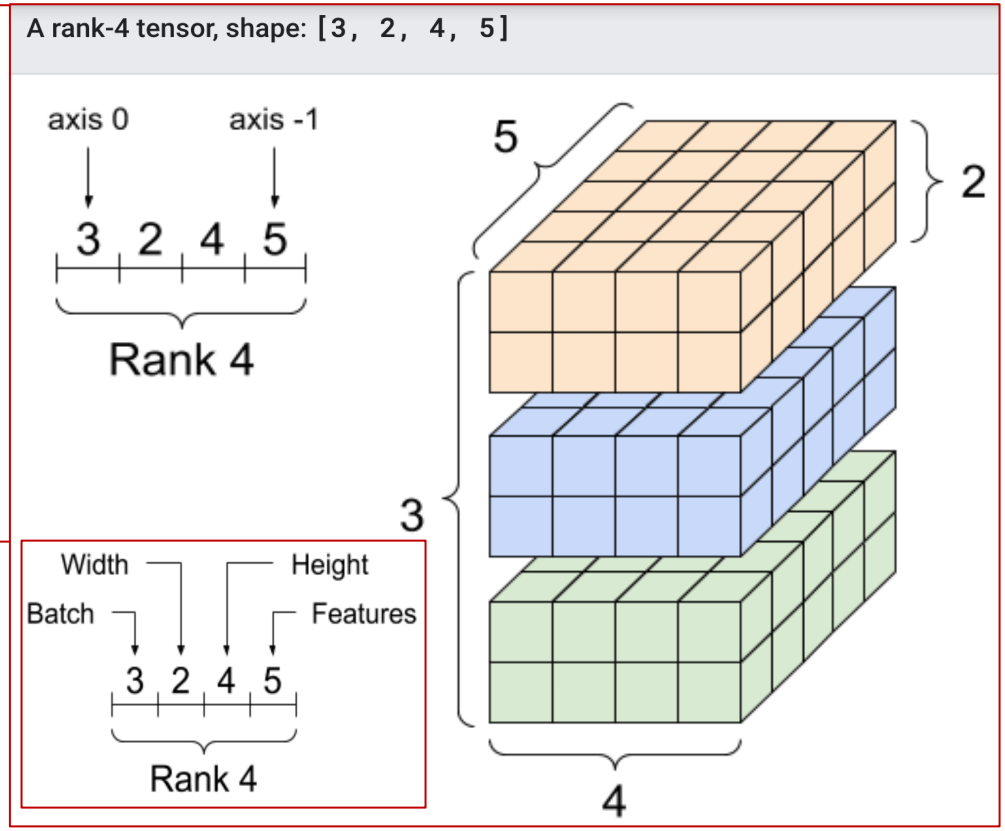




tf.TensorShape

Tensors have shapes.

- ✓ Shape: The length (number of elements) of each of the dimensions of a tensor.
- ✓ Rank: Number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix is rank 2.
- ✓ Axis or Dimension: A particular dimension of a tensor.
- ✓ Size: The total number of items in the tensor, the product shape vector




```
print("Type of every element:",
      rank_4_tensor.dtype)
print("Number of dimensions:",
      rank_4_tensor.ndim)
print("Shape of tensor:",
      rank_4_tensor.shape)
print("Elements along axis 0 of tensor:",
      rank_4_tensor.shape[0])
print("Elements along the last axis of tensor:",
      rank_4_tensor.shape[-1])
print("Total number of elements (3*2*4*5): ",
      tf.size(rank_4_tensor).numpy())
```

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```

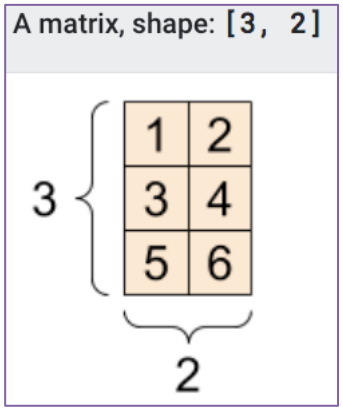
```
Type of every element: <dtype: 'float32'>
Number of dimensions: 4
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
Total number of elements (3*2*4*5): 120
```

Shape of Tensors

TensorFlow follows standard Python indexing rules, similar to indexing a list or a string in Python, and the basic rules for NumPy indexing.

- ✓ indexes start at 0
- ✓ negative indices count backwards from the end
- ✓ colons, :, are used for slices: start:stop:step

rank_1_tensor
[0 1 1 2 3 5 8 13 21 34]




```
print("Everything:", rank_1_tensor[:].numpy())
print("Before 4:", rank_1_tensor[:4].numpy())
print("From 4 to the end:", rank_1_tensor[4:].numpy())
print("From 2, before 7:", rank_1_tensor[2:7].numpy())
print("Every other item:", rank_1_tensor[::2].numpy())
print("Reversed:", rank_1_tensor[::-1].numpy())
```

```
Everything: [ 0 1 1 2 3 5 8 13 21 34]
Before 4: [0 1 1 2]
From 4 to the end: [ 3 5 8 13 21 34]
From 2, before 7: [1 2 3 5 8]
Every other item: [ 0 1 3 8 21]
Reversed: [34 21 13 8 5 3 2 1 1 0]
```

```
# Get row and column tensors
print("Second row:", rank_2_tensor[1, :].numpy())
print("Second column:", rank_2_tensor[:, 1].numpy())
print("Last row:", rank_2_tensor[-1, :].numpy())
print("First item in last column:", rank_2_tensor[0, -1].numpy())
print("Skip the first row:")
print(rank_2_tensor[1:, :].numpy(), "\n")
```

```
Second row: [3. 4.]
Second column: [2. 4. 6.]
Last row: [5. 6.]
First item in last column: 2.0
Skip the first row: [[3. 4.] [5. 6.]]
```

tf.Tensor and tf.reshape objects

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

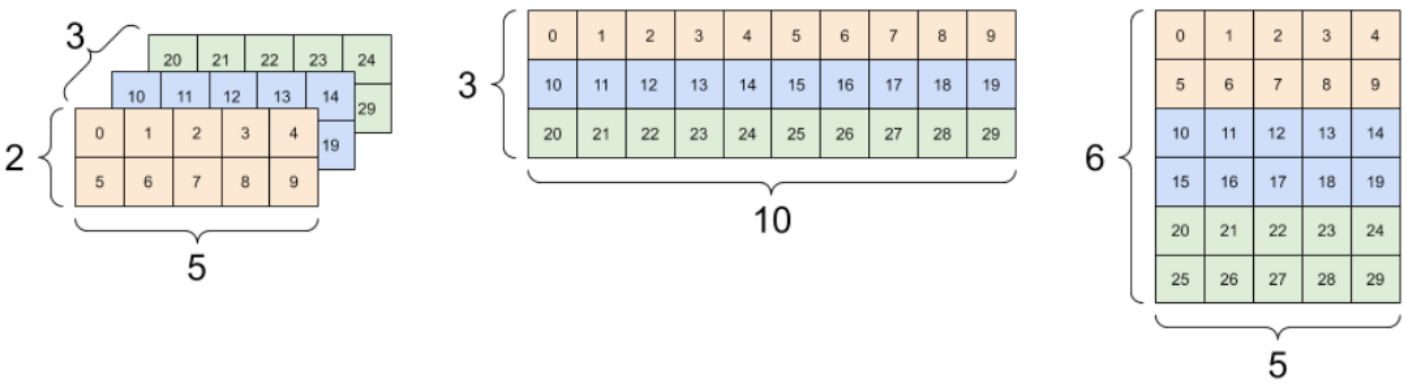
```
print(rank_3_tensor[:, :, 4])
```

```
tf.Tensor( [[ 4 9] [14 19] [24 29]],
shape=(3, 2), dtype=int32)
```

Selecting the last feature across all locations in each example in the batch


```
# A `-1` passed in the `shape` argument
says "Whatever fits", flatten a tensor
print(tf.reshape(rank_3_tensor, [-1]))
```

```
tf.Tensor( [ 0 1 2 3 4 5 6
7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29],
shape=(30,),
dtype=int32)
```




```
print(tf.reshape(rank_3_tensor, [3*2, 5]), "\n")
print(tf.reshape(rank_3_tensor, [3, -1]))
```

```
tf.Tensor( [[ 0 1 2 3 4] [ 5 6 7 8 9] [10 11 12 13 14]
[15 16 17 18 19] [20 21 22 23 24] [25 26 27 28 29]],
shape=(6, 5), dtype=int32)
tf.Tensor( [[ 0 1 2 3 4 5 6 7 8 9] [10 11 12 13 14 15
16 17 18 19] [20 21 22 23 24 25 26 27 28 29]],
shape=(3, 10), dtype=int32)
```

Tensors math operation : NLA

Joint Institute for Computational Sciences

INNOVATIVE COMPUTING LABORATORY

The base tf.Tensor class requires tensors to be "rectangular"---that is, along each axis, every element is the same size. However, Ragged tensors, Sparse tensors can handle different shapes.

addition, element-wise multiplication, and matrix multiplication.

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[1, 1], [1, 1]])

print(tf.add(a, b), "\n")
print(tf.multiply(a, b), "\n")
print(tf.matmul(a, b), "\n")

print(a + b, "\n") # element-wise addition
print(a * b, "\n") # element-wise multiplication
print(a @ b, "\n") # matrix multiplication

c = tf.constant([[4.0, 5.0], [10.0, 1.0]])

# Find the largest value
print(tf.reduce_max(c))
# Find the index of the largest value
print(tf.argmax(c))
# Compute the softmax
print(tf.nn.softmax(c))
```

```
tf.Tensor( [[2 3]
            [4 5]], shape=(2, 2), dtype=int32)
tf.Tensor( [[1 2]
            [3 4]], shape=(2, 2), dtype=int32)
tf.Tensor( [[3 3]
            [7 7]], shape=(2, 2), dtype=int32)
```

```
tf.Tensor( [[2 3] [4 5]], shape=(2, 2), dtype=int32)
tf.Tensor( [[1 2] [3 4]], shape=(2, 2), dtype=int32)
tf.Tensor( [[3 3] [7 7]], shape=(2, 2), dtype=int32)
```

```
tf.Tensor(10.0, shape=(), dtype=float32)
tf.Tensor([1 0], shape=(2,), dtype=int64)
tf.Tensor( [[2.6894143e-01 7.3105860e-01]
[9.9987662e-01 1.2339458e-04]],
           shape=(2, 2), dtype=float32)
```

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax

$$\sigma(\vec{z})_1 = \sigma(3) = \frac{e^3}{e^3 + e^0} = 0.953$$

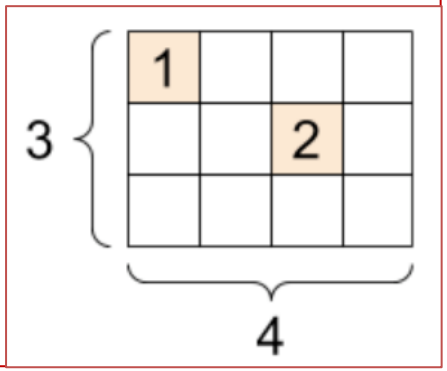
$$\sigma(\vec{z})_2 = \sigma(0) = \frac{e^0}{e^3 + e^0} = 0.0474$$

Sigmoid

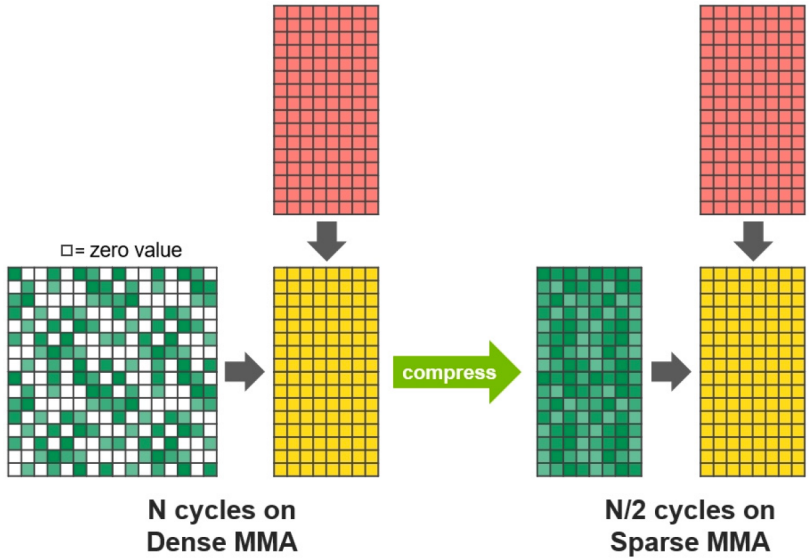
$$S(x) = S(3) = \frac{1}{1 + e^{-3}} = \frac{1}{1 + e^{-3}} = 0.953$$

Sparse Tensors

```
# Sparse tensors store values by index in a memory-efficient manner
sparse_tensor = tf.sparse.SparseTensor(indices=[[0, 0], [1, 2]],
                                       values=[1, 2],
                                       dense_shape=[3, 4])
print(sparse_tensor, "\n")

# We can convert sparse tensors to dense
print(tf.sparse.to_dense(sparse_tensor))
```




```
SparseTensor(indices=tf.Tensor( [[0 0] [1 2]],
shape=(2, 2), dtype=int64),
values=tf.Tensor([1 2], shape=(2,),
dtype=int32), dense_shape=tf.Tensor([3 4],
shape=(2,), dtype=int64))

tf.Tensor( [[1 0 0 0] [0 0 2 0] [0 0 0 0]],
shape=(3, 4), dtype=int32)
```



Example Dense MMA and Sparse MMA operations using 16x16 sparse matrix (Matrix A), multiplied by a dense 16x8 matrix (Matrix B). Sparse MMA operation on right doubles throughput by skipping compute of zero values

Figure 13. Example Dense MMA and Sparse MMA operations

TensorFlow tf.Variable objects

```
my_tensor = tf.constant([[1.0, 2.0], [3.0, 4.0]])
my_variable = tf.Variable(my_tensor)

# Variables can be all kinds of types, just like tensors
bool_variable = tf.Variable([False, False, False, True])
complex_variable = tf.Variable([5 + 4j, 6 + 1j])
```

```
print("A variable:",my_variable)
print("\nViewed as a tensor:",
tf.convert_to_tensor(my_variable))
print("\nIndex of highest value:", tf.argmax(my_variable))

# This creates a new tensor; it does not reshape the variable.
print("\nCopying and reshaping: ", tf.reshape(my_variable,
([1,4])))
```

```
A variable: <tf.Variable 'Variable:0' shape=(2, 2) dtype=float32,
numpy= array([[1., 2.], [3., 4.]], dtype=float32)>
Viewed as a tensor: tf.Tensor( [[1. 2.] [3. 4.]], shape=(2, 2),
dtype=float32) Index of highest value: tf.Tensor([1 1], shape=(2,),
dtype=int64)
Copying and reshaping: tf.Tensor([[1. 2. 3. 4.]], shape=(1, 4),
dtype=float32)
```

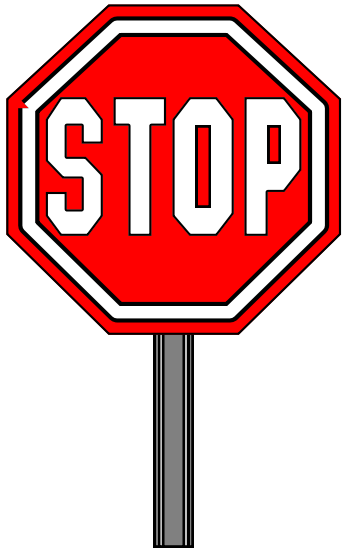
```
a = tf.Variable([2.0, 3.0])
# Create b based on the value of a
b = tf.Variable(a)
a.assign([5, 6])

# a and b are different
print(a.numpy())
print(b.numpy())

# There are other versions of assign
print(a.assign_add([2,3]).numpy())  # [7. 9.]
print(a.assign_sub([7,9]).numpy())  # [0. 0.]
```

```
[5.  6.]
[2.  3.]
[7.  9.]
[0.  0.]
```

```
with tf.device('CPU:0'):
  a = tf.Variable([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
  b = tf.Variable([[1.0, 2.0, 3.0]])

with tf.device('GPU:0'):
  # Element-wise multiply
  k = a * b

print(k)
```

```
tf.Tensor( [[ 1. 4. 9.] [ 4. 10. 18.]], shape=(2, 3), dtype=float32)
```

The End



- The End!