CEA 2023@Paris

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

# Heterogeneous Systems for Exascale using h3-Open-SYS/WaitIO

Shinji Sumimoto, Takashi Arakawa, Yoshio Sakaguchi, Hiroya Matsuba*, Hisashi Yashiro, Toshihiro Hanawa, Kengo Nakajima

2023/03/7

FUJITSU

国立研究開発法人 国立環境研究所
National Institute for Environmental Studies

東京大学 THE UNIVERSITY OF TOKYO
東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

RIKEN

R-CCS

*Achievements as a visiting professor at the University of Tokyo, 2021

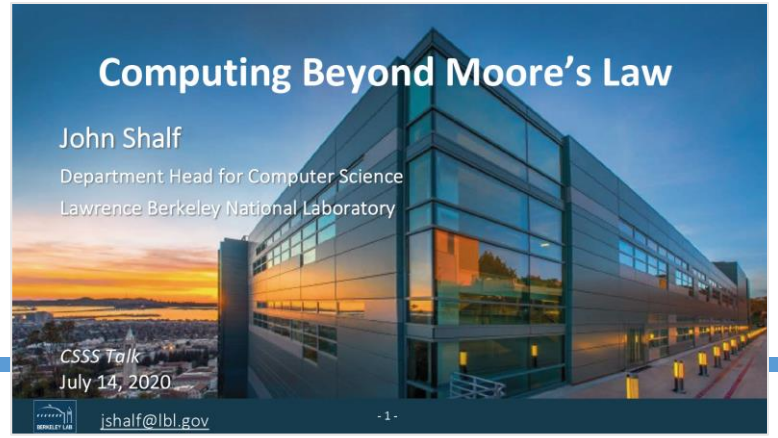
東京大学 THE UNIVERSITY OF TOKYO

# Overview of This Talk

- Background
  - Exascale Computing Trend:
    Post Moore, Disaggregated Computing, Modular Computing,
    Cloud Computing
  - Heterogeneous Coupling Computing: h3-Open-BDEC Project
- H3-Open-SYS/WaitIO
  - Design and Implementation
  - Evaluation
  - Related Work

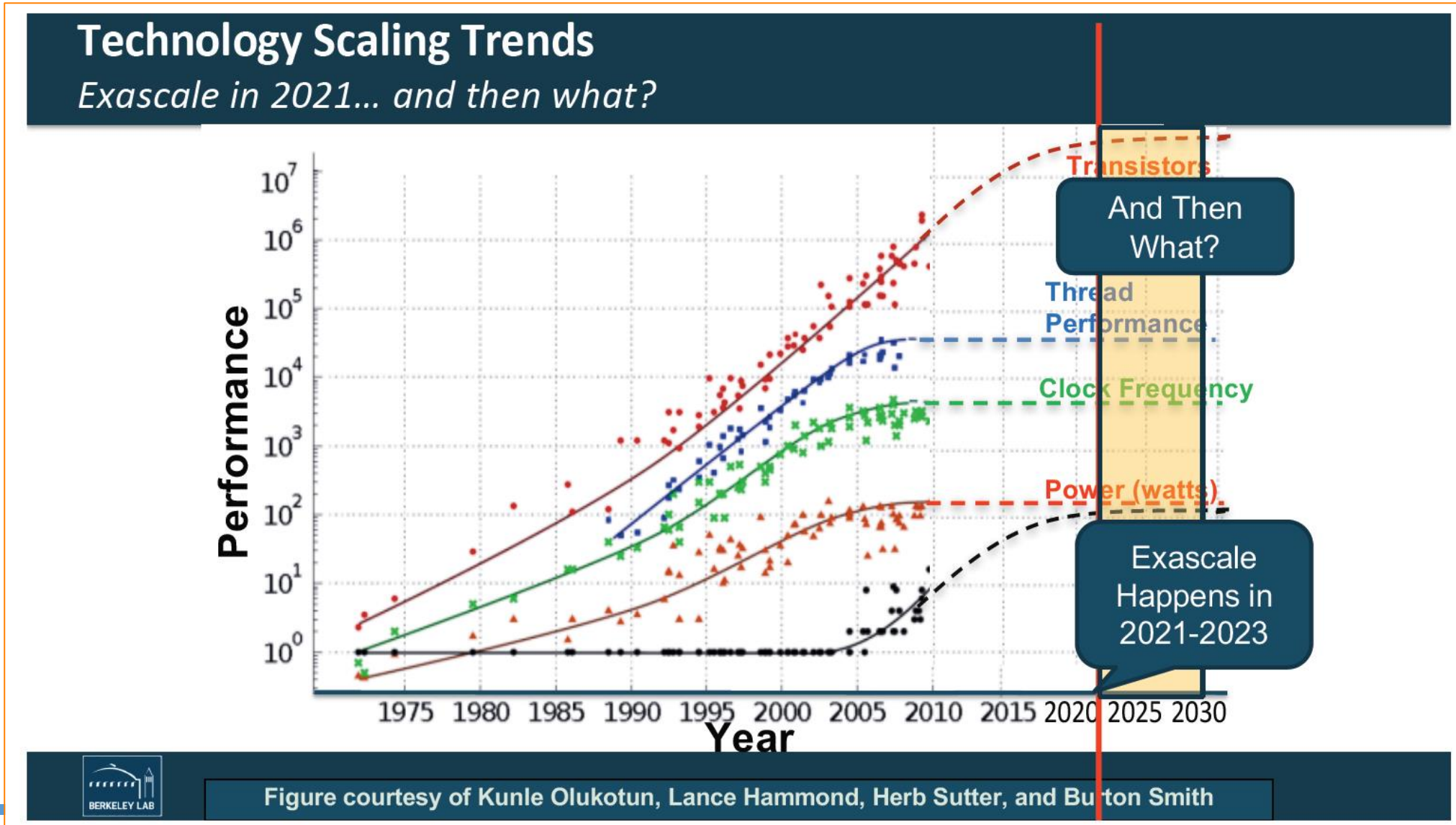
# Post Moore : Computing Beyond Moore's Law

Computing Beyond Moore's Law
https://cs.lbl.gov/assets/CSSSP-Slides/20200714-Shalf.pdf



# Computing Beyond Moore's Law

https://cs.lbl.gov/assets/CSSSP-Slides/20200714-Shalf.pdf

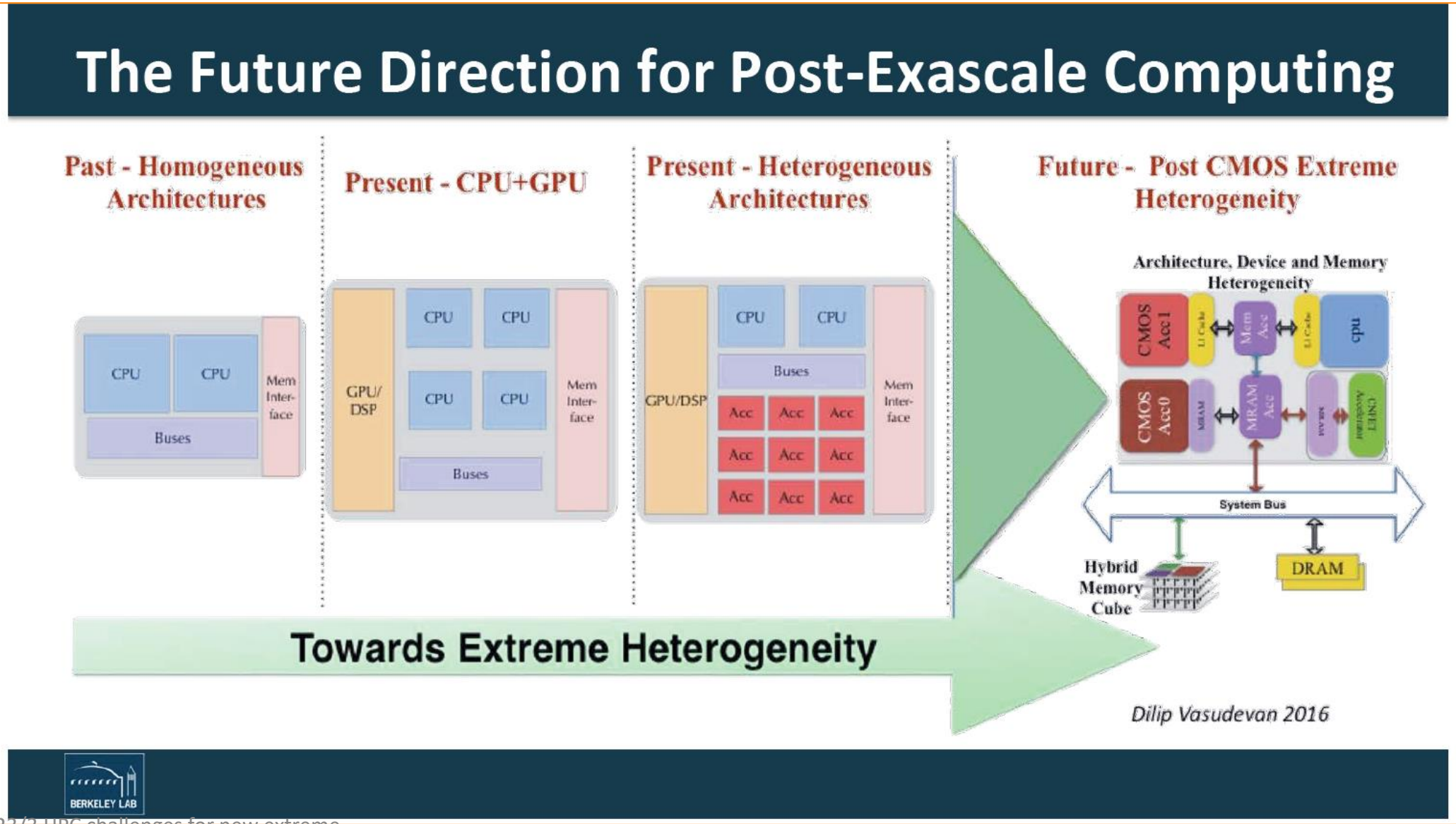


# Computing Beyond Moore's Law

https://cs.lbl.gov/assets/CSSSP-Slides/20200714-Shalf.pdf

# Computing Beyond Moore's Law

https://cs.lbl.gov/assets/CSSSP-Slides/20200714-Shalf.pdf



# Computing Beyond Moore's Law

https://cs.lbl.gov/assets/CSSSP-Slides/20200714-Shalf.pdf

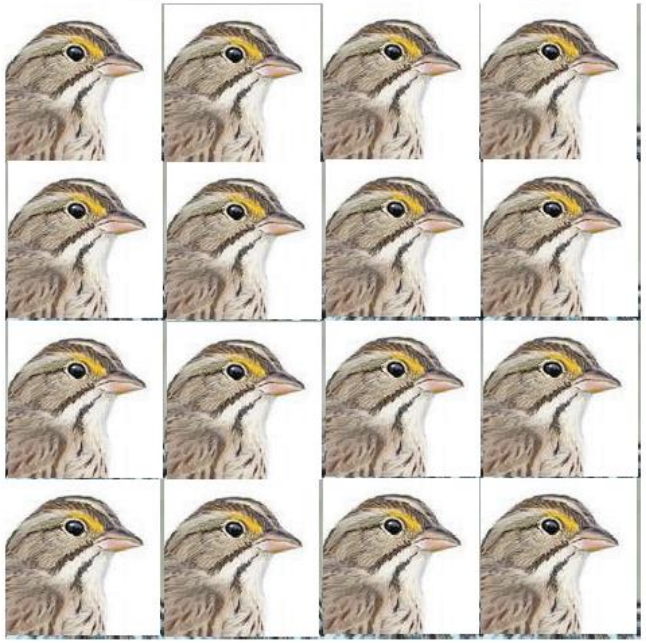
## Specialization:

### Natures way of Extracting More Performance in Resource Limited Environment

**Powerful General Purpose**

Xeon, Power

**Many Lighter Weight**
(post-Dennard scarcity)



KNL AMD, Cavium/Marvell, GPU

**Many Different Specialized**
(Post-Moore Scarcity)


Apple, Google, Amazon

BERKELEY LAB

6

# Project 38

https://crd.lbl.gov/departments/computer-science/cag/research-2/project-2/

## Project 38 -- Background

***DOD and DOE recognize the imperative to develop new mechanisms for engagement with the vendor community, particularly on architectural innovations with strategic value to USG HPC.***

**Project 38 (P38) is a set of vendor-agnostic architectural explorations involving DOD, the DOE Office of Science, and NNSA (these latter two organizations are referred to in this document as "DOE"). These explorations should accomplish the following:**

- **Near-term goal:** *Quantify the performance value and identify the potential costs of specific architectural concepts against a limited set of applications of interest to both the DOE and DOD.*
- **Long-term goal:** *Develop an enduring capability for DOE and DOD to jointly explore architectural innovations and quantify their value.*
- **Stretch goal:** *Specification of a shared, purpose built architecture to drive future DOE-DOD collaborations and investments. (purpose-built HPC by 2025)*

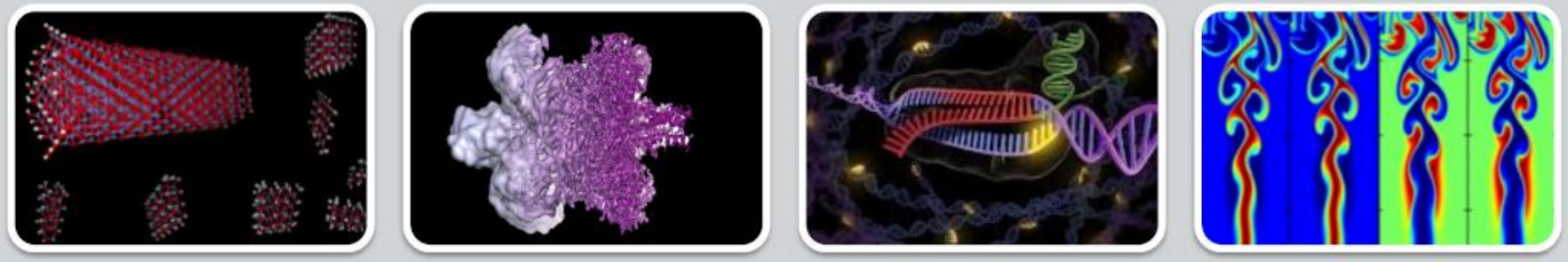

BERKELEY LAB

# Project 38

https://crd.lbl.gov/departments/computer-science/cag/research-2/project-2/

## Architecture Specialization for Science

*(hardware is design around the algorithms) can't design effective hardware without math*



| Materials | CryoEM Accelerator | Genomics Accelerator | Digital fluid Accelerator |
|---|---|---|---|
| Density Functional Theory (DFT) | LBNL detector | String matching | 3D integration |
| Use O(n) algorithm | 750 GB / sec | Hashing | Petascale *chip* |
| Dominated by FFTs | Custom ASIC near detector | 2-8bit (ACTG) | 1024-layers |
| FPGA or ASIC | | FPGA solution | General / special HPC solution |

BERKELEY LAB

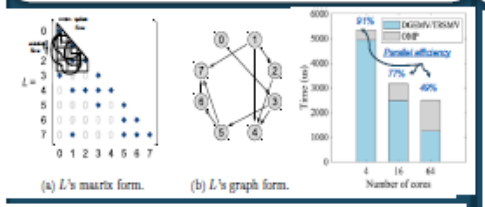
- 10 -

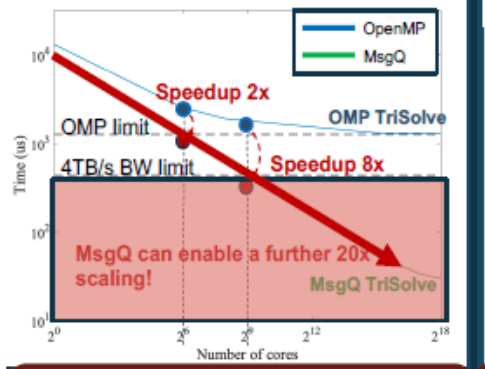
# Project 38

https://crd.lbl.gov/departments/computer-science/cag/research-2/project-2/

## Phase 1 Studies

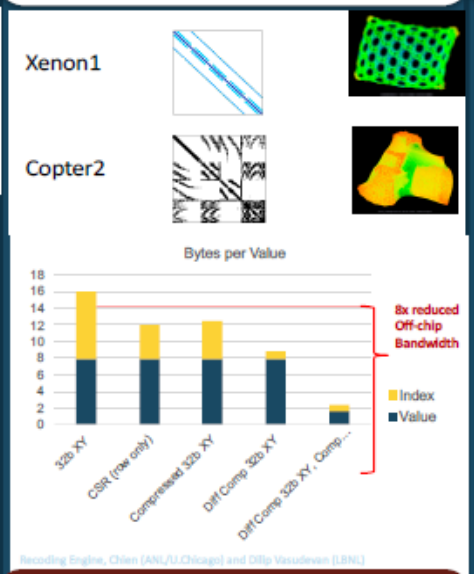
### Message Queues (Sparse Matrix TRSMV)



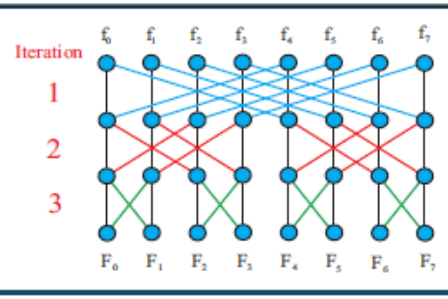

8x improved scaling (limited by memory BW) with room to grow

### Recoding Engine (Sparse Matrix Compression)




2x-8x Reduction in Memory Bandwidth req.

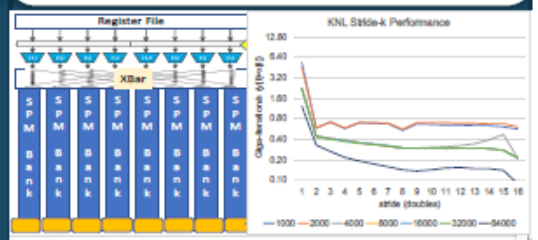
### Fixed Function Accelerators (FFT as example)




**1GHz @ 14nm**
**100 GB/s off-chip memory**
**~1.5TFLOPs uses 4.5mm²**
**1TB/s off-chip memory**
**~15TFLOPs uses 47mm²**

Up to 20x-100x less area consumption for same perf. (viable FFTx/BLAS)

### Gather/Scatter Scratchpads (for Tensor Contractions)



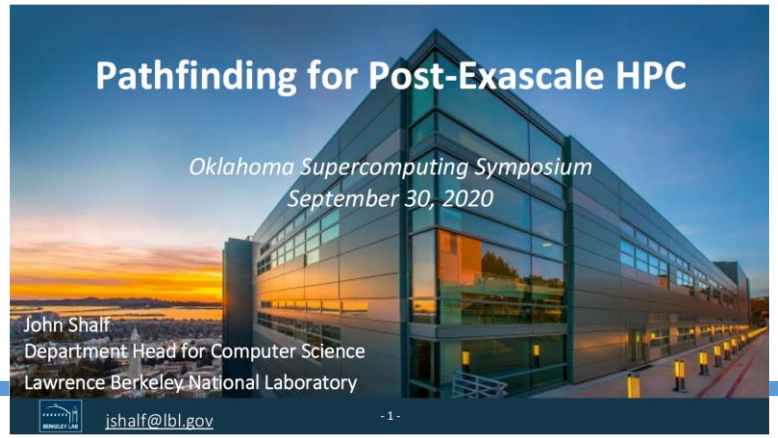

| Bandwidth waste for loading the t3 or v in inner loop | Bandwidth waste for the entire application |
|---|---|
| 55% | 36% |
| 100% | 65.4% |
| 700% | 457.8% |
| 154% | 100.7% |
| 166% | 109% |

55%-700% reduction in data movement

BERKELEY LAB

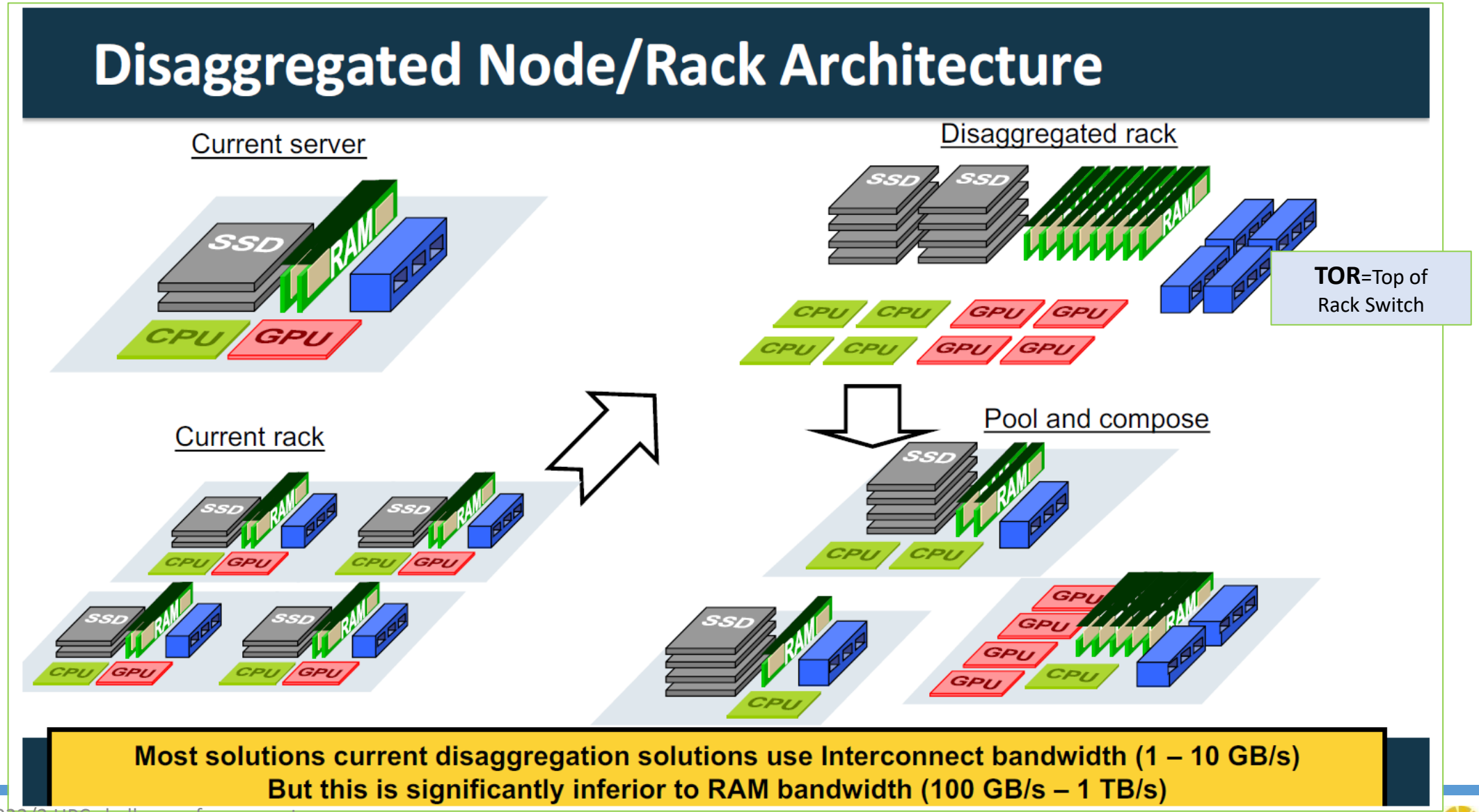
# Disaggregated Computing

https://www.oscer.ou.edu/Symposium2020/oksupercompsymp2020_talk_shalf_20200930.pdf



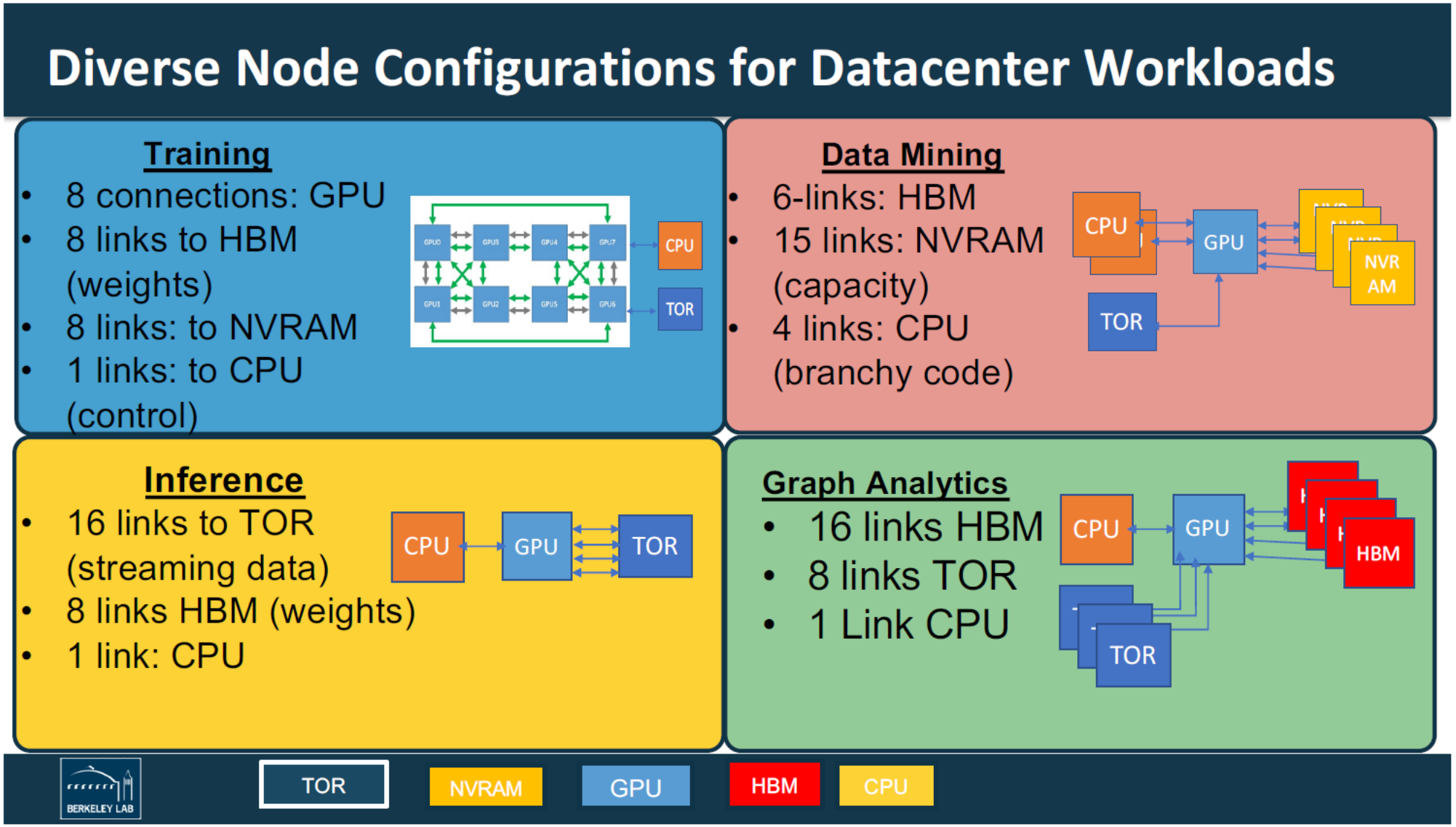
# Disaggregated Computing

https://www.oscer.ou.edu/Symposium2020/oksupercompsymp2020_talk_shalf_20200930.pdf



# Disaggregated Computing

https://www.oscer.ou.edu/Symposium2020/oksupercompsymp2020_talk_shalf_20200930.pdf

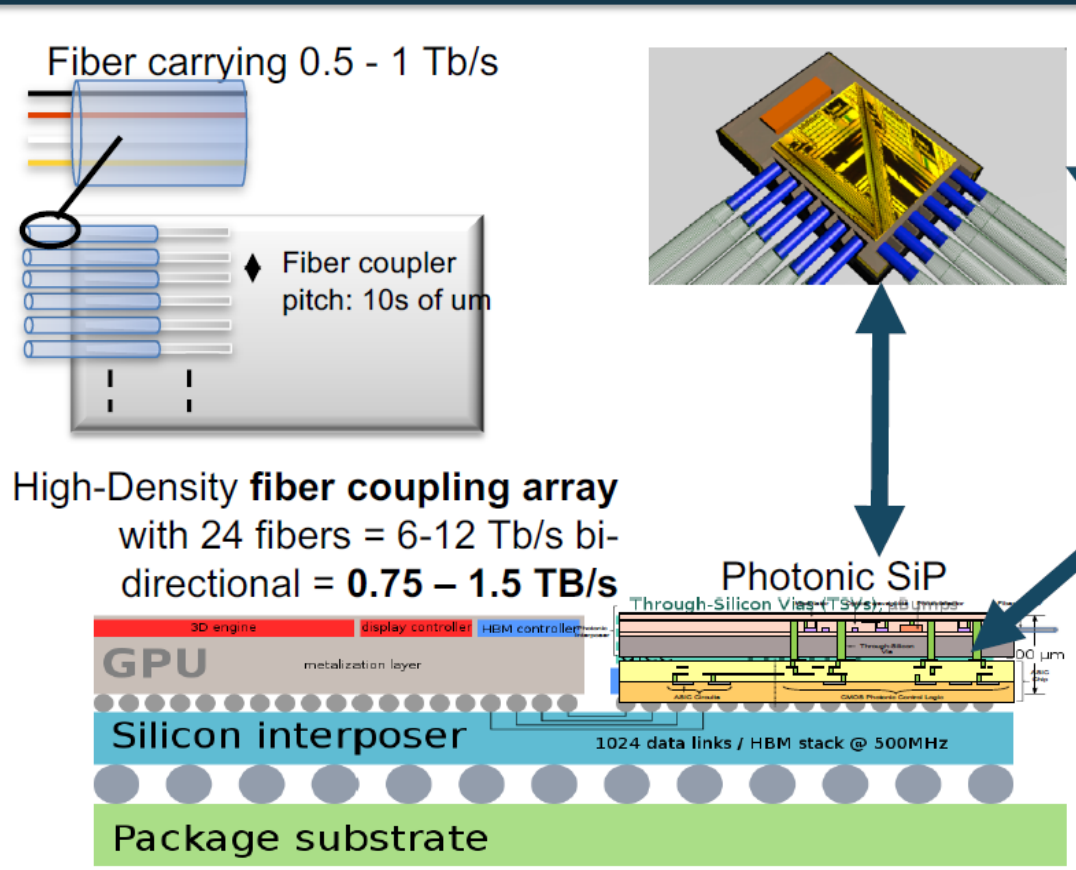



TOR=Top of Rack Switch

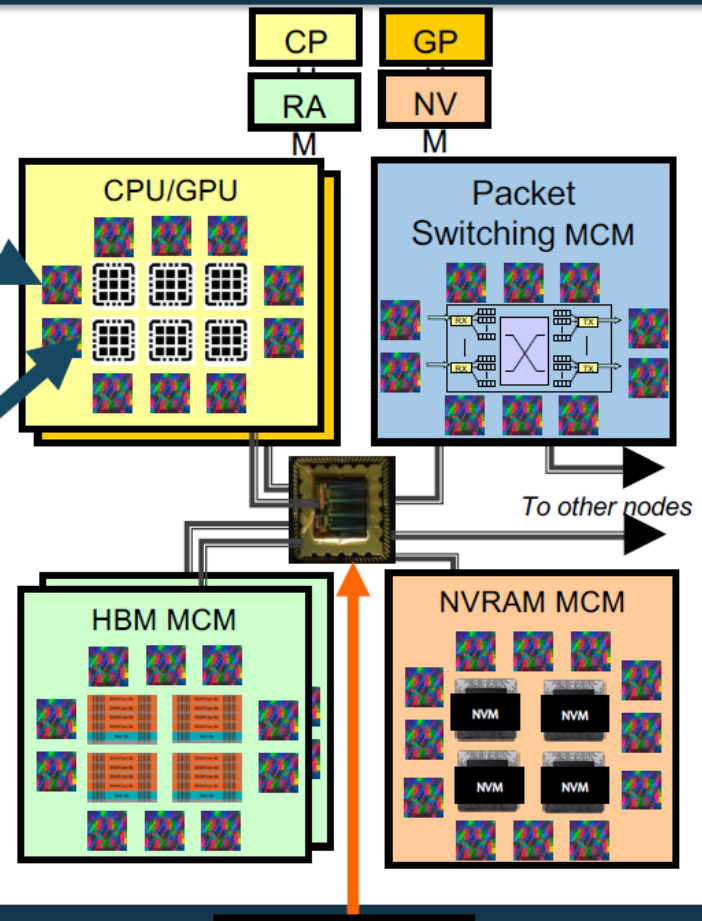
# Disaggregated Computing

https://www.oscer.ou.edu/Symposium2020/oksupercompsymp2020_talk_shalf_20200930.pdf

## Photonic MCM (Multi-Chip Module)





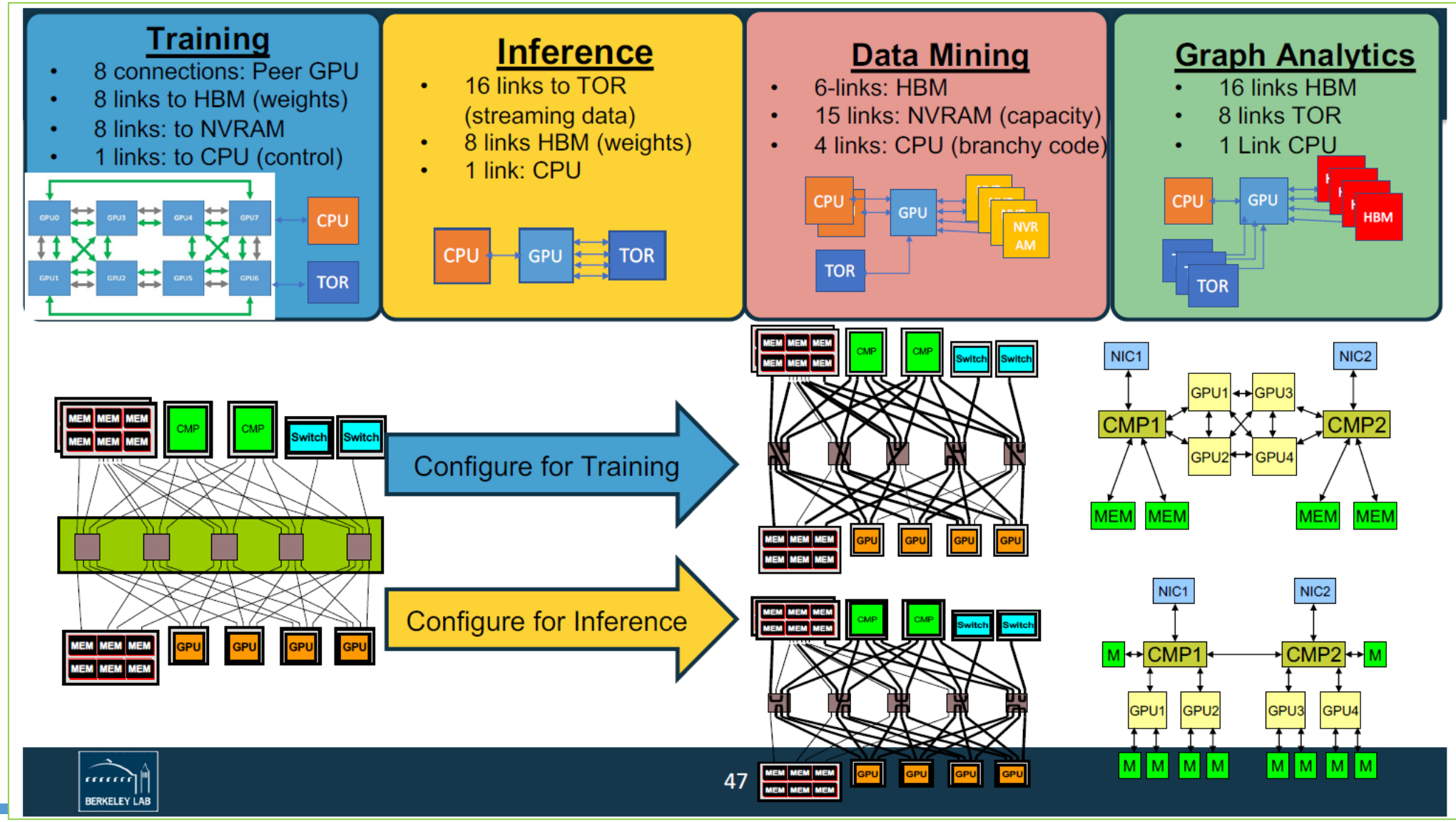




BERKELEY LAB

46

# Disaggregated Computing

https://www.oscer.ou.edu/Symposium2020/oksupercompsymp2020_talk_shalf_20200930.pdf

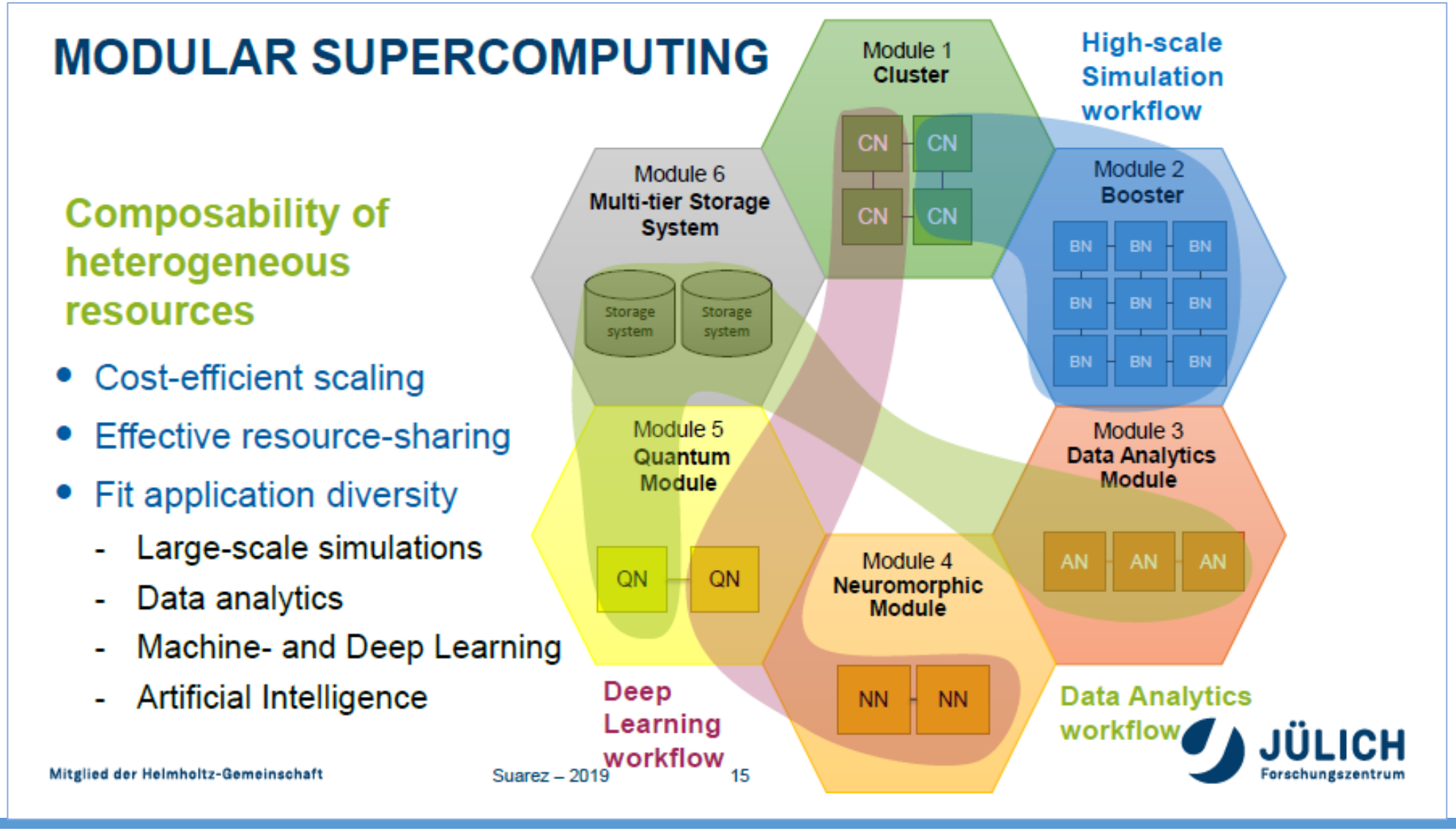


# The Modular Computing Architecture

https://tu-dresden.de/zih/die-einrichtung/ressourcen/dateien/kolloquium/2019_11_28_EstelaSuarez.pdf?lang=en

# The Modular Computing Architecture

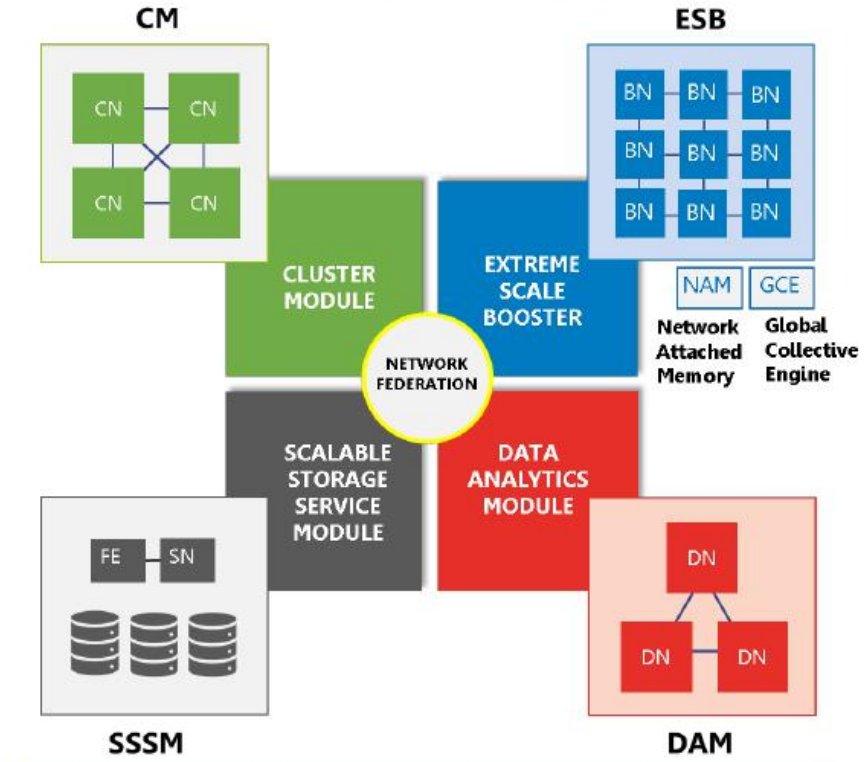
https://tu-dresden.de/zih/die-einrichtung/ressourcen/dateien/kolloquium/2019_11_28_EstelaSuarez.pdf?lang=en



# The Modular Computing Architecture

https://tu-dresden.de/zih/die-einrichtung/ressourcen/dateien/kolloquium/2019_11_28_EstelaSuarez.pdf?lang=en

## DEEP-EST prototype




Prototype co-designed with Software and Applications

DEEP Projects

Source: FZJ

Early-Access program in 2020!

16

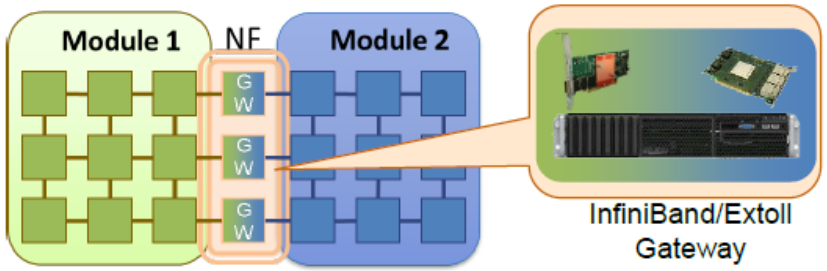
ÜLICH
chungszentrum

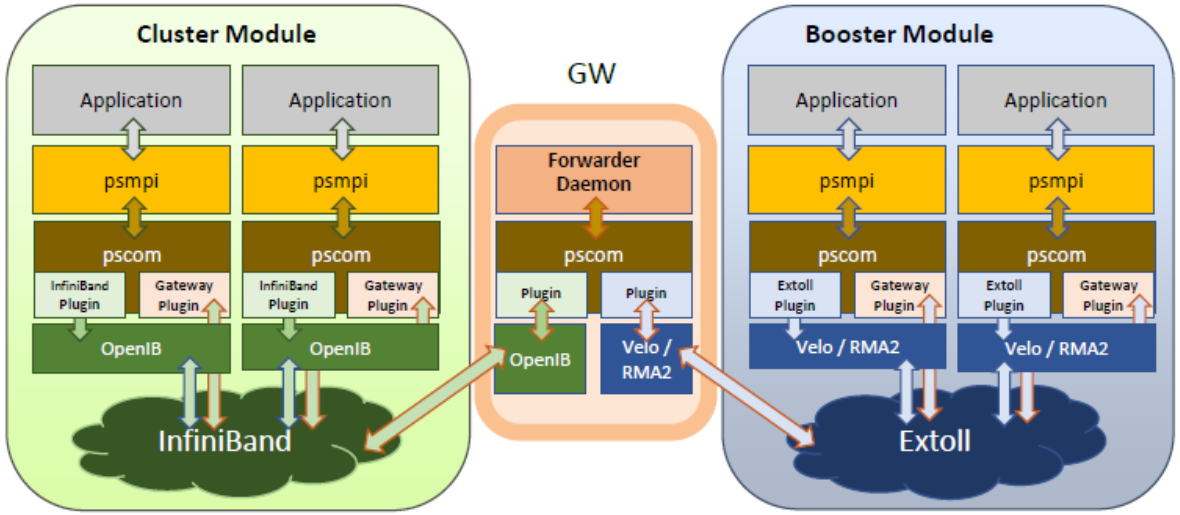
# The Modular Computing Architecture

https://tu-dresden.de/zih/die-einrichtung/ressourcen/dateien/kolloquium/2019_11_28_EstelaSuarez.pdf?lang=en

## NETWORK BRIDGING

- Classical Gateway approach
  - Just one additional hop
- Forwarder daemons translate between module-networks



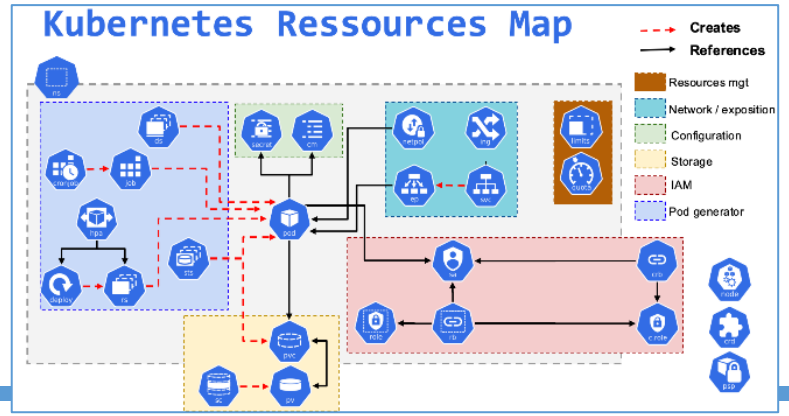

• **Eicker et al.**, *Bridging the DEEP Gap - Implementation of an Efficient Forwarding Protocol*, Intel European Exascale Labs - Report 2013 34-41, (2014)

Mitglied der Helmholtz-Gemeinschaft Suarez – 2019 20

JÜLICH Forschungszentrum

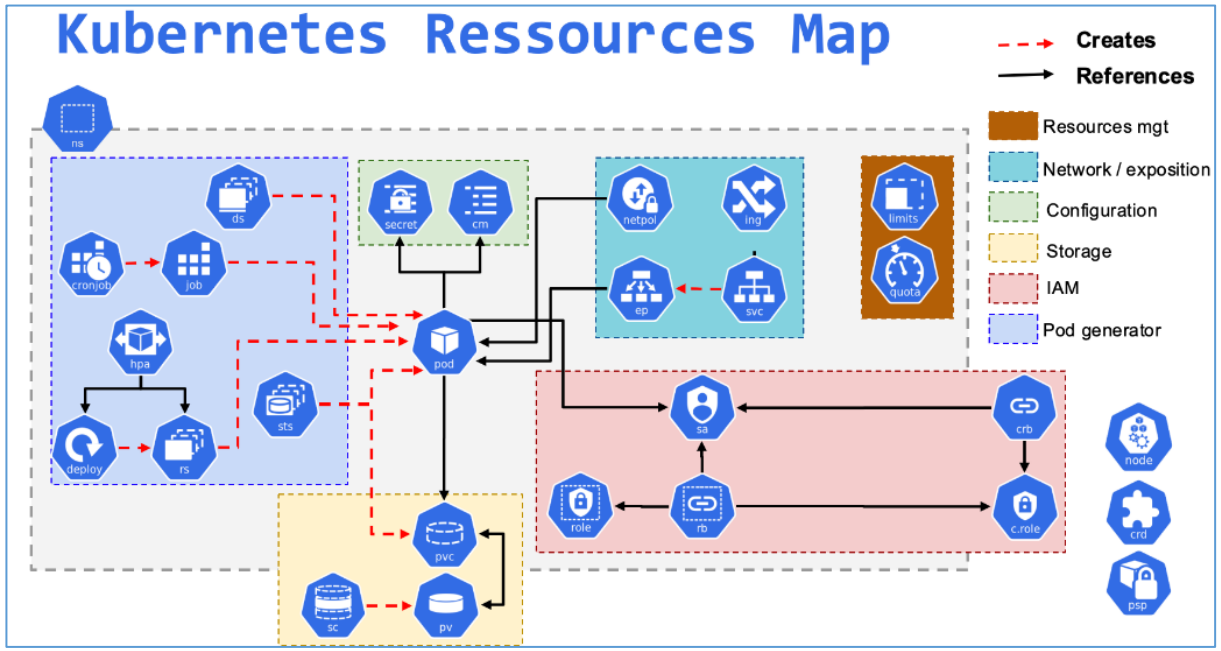
# Cloud Native Computing

https://www.cncf.io/



# Cloud Native Computing

- Server Less
  - On demand Provisioning
  - Automatic Deploy and Scaling
- Run Anywhere by Container, Docker
  - Micro Services, Isolated Processes, Event(Message) Driven
- Loosely Coupled (vs. Tightly Coupled)
  - Orchestration, Fault Tolerance, Automatic Scaling
- Not only User Program but also System Software Configured Dynamically



# Worlds Going to Heterogeneous and Dynamically Configured Systems!

# Worlds Going to Heterogeneous and Dynamically Configuring!

- Dynamically fitting to real workloads to use electricity effectively
  - Cloud Native Computing to fit various workload at system software level
  - Disaggregated Computing to fit each application specialized

- So how Heterogeneous Computing works?
  1. Tuning Single Application to use part by part using different hardware
  2. Coupling multiple application to solve more complicated problems with each application tuned by different hardware architecture

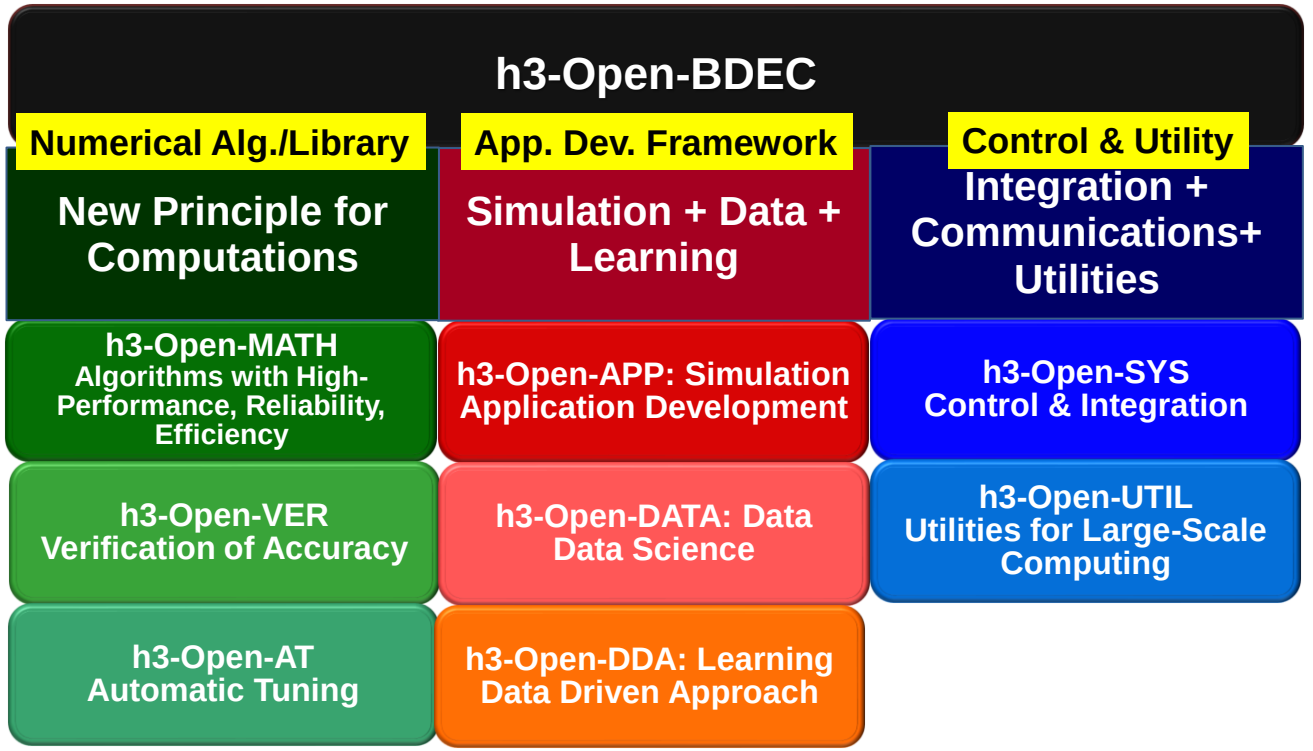
- Heterogeneous Coupling of Multiple Applications: h3-BDEC-01

# H3-BDEC-01: Heterogeneous Coupling Computing

https://h3-open-bdec.cc.u-tokyo.ac.jp/

# h3-Open-BDEC Innovative Software Platform for Integration of (S+D+L) on the BDEC System, such as Wisteria/BDEC-01

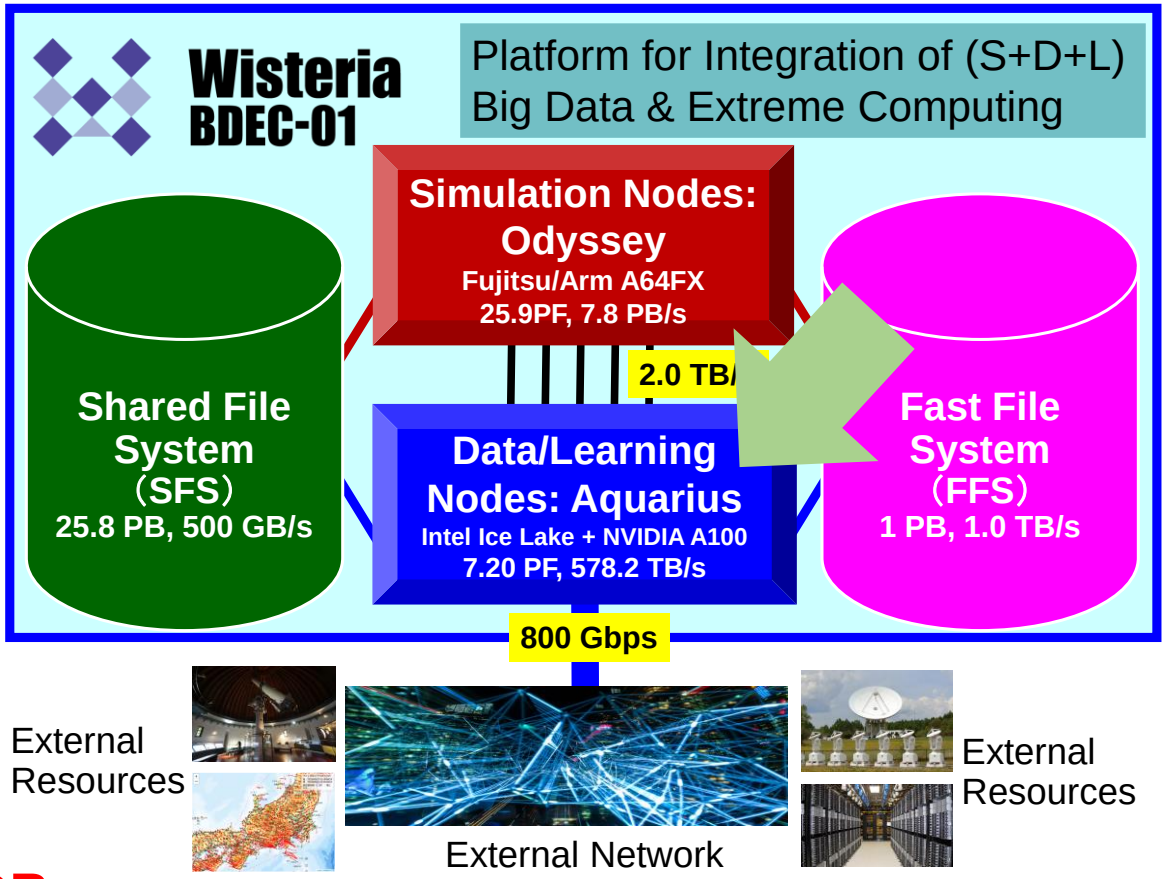
- "Three" Innovations
  - New Principles for Numerical Analysis by Adaptive Precision, Automatic Tuning & Accuracy Verification
  - Hierarchical Data Driven Approach (*h*DDA) based on Machine Learning
  - Software & Utilities for Heterogenous Environment, such as Wisteria/BDEC-01



# Wisteria/BDEC-01

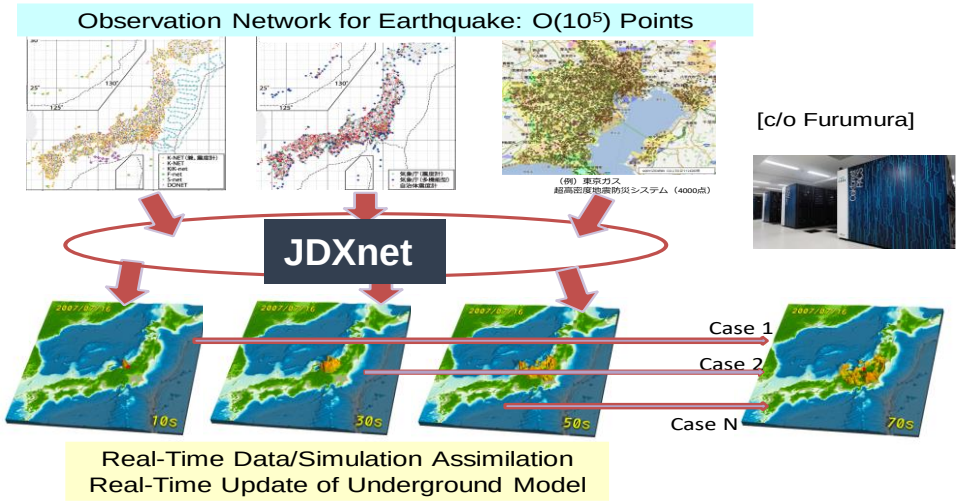
**The 1st BDEC System (Big Data & Extreme Computing) Platform for Integration of (S+D+L)**

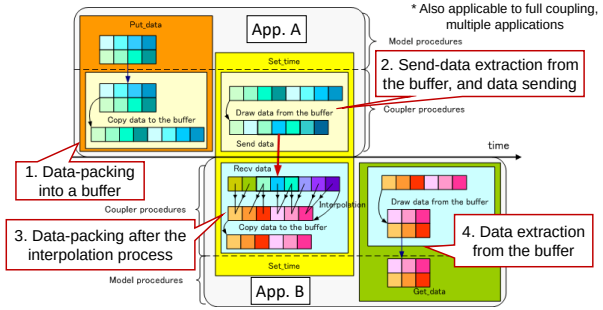
- Operation starts on May 14, 2021
- 33.1 PF, 8.38 PB/sec by **Fujitsu**
  - ~4.5 MVA with Cooling, ~360m$^2$
- **2 Types of Node Groups**
  - Hierarchical, Hybrid, Heterogeneous (h3)
  - **Simulation Nodes: Odyssey**
    - **Fujitsu PRIMEHPC FX1000 (A64FX), 25.9 PF**
      - **7,680 nodes (368,640 cores), Tofu-D**
      - **General Purpose CPU + HBM**
      - **Commercial Version of "Fugaku"**
  - **Data/Learning Nodes: Aquarius**
    - **Data Analytics & AI/Machine Learning**
    - **Intel Xeon Ice Lake + NVIDIA A100, 7.2PF**
      - **45 nodes (90x Ice Lake, 360x A100), IB-HDR**
    - **Some of the DL nodes are connected to external resources directly**
- File Systems: SFS (Shared/Large) + FFS (Fast/Small)

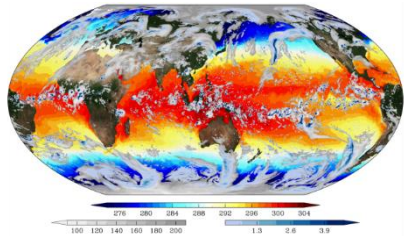


# Heterogeneous Coupling Computing

- With the spread of higher-scale computing systems, HPC applications have become able to solve more practical problems.
  - Example 1: Realizing real-time simulation with various kind of sensor data and estimate the future.
  - Example 2: Improving calculation accuracy by assimilating calculation results and real-time data.
  - Example 3: Using computer simulation results as machine learning data and doing simulation using inference computation.
- Fusion of multiple applications is the key with Simulation, Data processing and machine Learning
  - Heterogeneous Coupling Computing



# Heterogeneous Coupling Computing: Issue and Requirements



- Issue:
  - No standard communication protocols between systems
- Solution:
  - Developing Communication Library: h3-Open-SYS/WaitIO(WaitIO)
- WaitIO Communication Requirements:
  - Operating on Heterogeneous Hardware and Software:
    - Multiple applications on different kinds of nodes to be combined.
    - Applicable to Heterogeneous Hardware and Software Environments
  - Combining multiple application groups with less effort:
    - Software modification should be minimal
    - Close to the programming standard
    - Should be coexistent with other system software

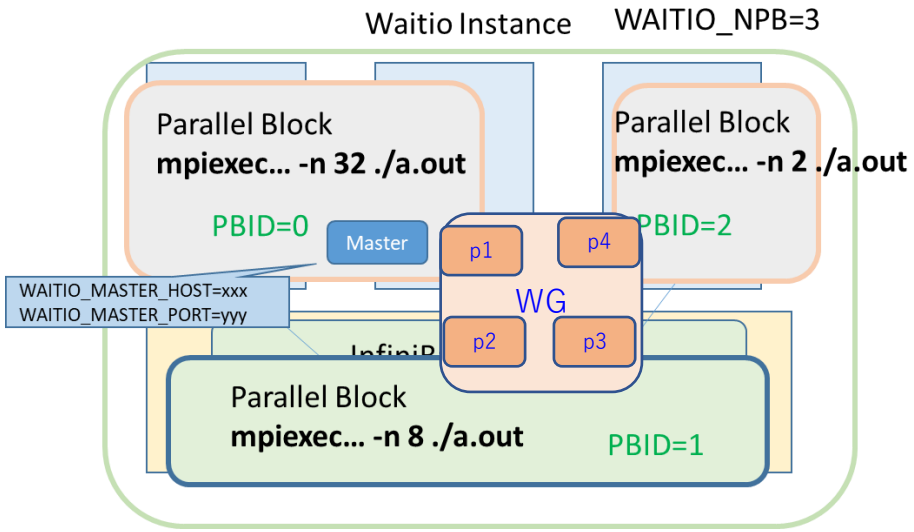
# WaitIO-Socket Design

- Adoption of Socket dedicated interface: WaitIO-Socket
  - Supports different types of processors, different types of interconnects, and different kinds of MPIs.
  - Odyssey's interconnect, Tofu-D Interconnect, also supports TCP/IP communication.
  - Possible to coexist with MPI and other system software.
- Providing APIs that conform to the MPI specifications:
  - Should be as much as possible the same as the communication libraries that are usually used.
- Operating with minimal computer resources: for > 10,000 procs.
  - Application user can select processes needed to communicate.
  - And all communication resources are allocated on demand.

# API of WaitIO: PB (Parallel Block) == Each Application

- Application is able to select communication processes among PBs

| WaitIO API | Description |
|---|---|
| waitio_isend | Non-Blocking Send |
| waitio_irecv | Non-Blocking Receive |
| waitio_wait | Termination of waitio_isend/irecv |
| waitio_init | Initialization of WaitIO |
| waitio_get_nprocs | Process # for each PB (Parallel Block) |
| waitio_create_group<br>waitio_create_group_wranks | Creating communication groups among PB's |
| waitio_group_rank | Rank ID in the Group |
| waitio_group_size | Size of Each Group |
| waitio_pb_size | Size of the Entire PB |
| waitio_pb_rank | Rank ID of the Entire PB |



# Implementation of WaitIO-Socket

- WaitIO initialization: waitio_init()
  - A set of IP addresses and port numbers on all PBs is shared among all processes on PBs.
- Communication protocol:
  - The Eager protocol (<= 128 bytes), the Rendezvous protocol ( >128 bytes)
- Communication progress processing: When waitio_wait() is called
  - Except for the Eager protocol for which communication has been established.
- Receive message processing:
  - Implemented using the “Linux epoll” (event polling) functions.
- Send message processing:
  - Implemented by the basic “write” system-call directly
  - Shifting to the processing by the “Linux epoll” when the write processing to the socket causes an error such as an EINTR factor.
- Ensuring communication reliability:
  - Magic numbers and sequence numbers are introduced in control packets to detect errors.
- Implementation code size: Library is 5.7K steps in C.

# Evaluation of WaitIO-Socket

- PingPong Communication Bandwidth
  - Multi-stream PingPong Communication
- Application Performance with h3-OpenUTIL/MP on Wisteria/BDEC-01
  - For Heterogeneous Computing(Simulation + Machine Learning)

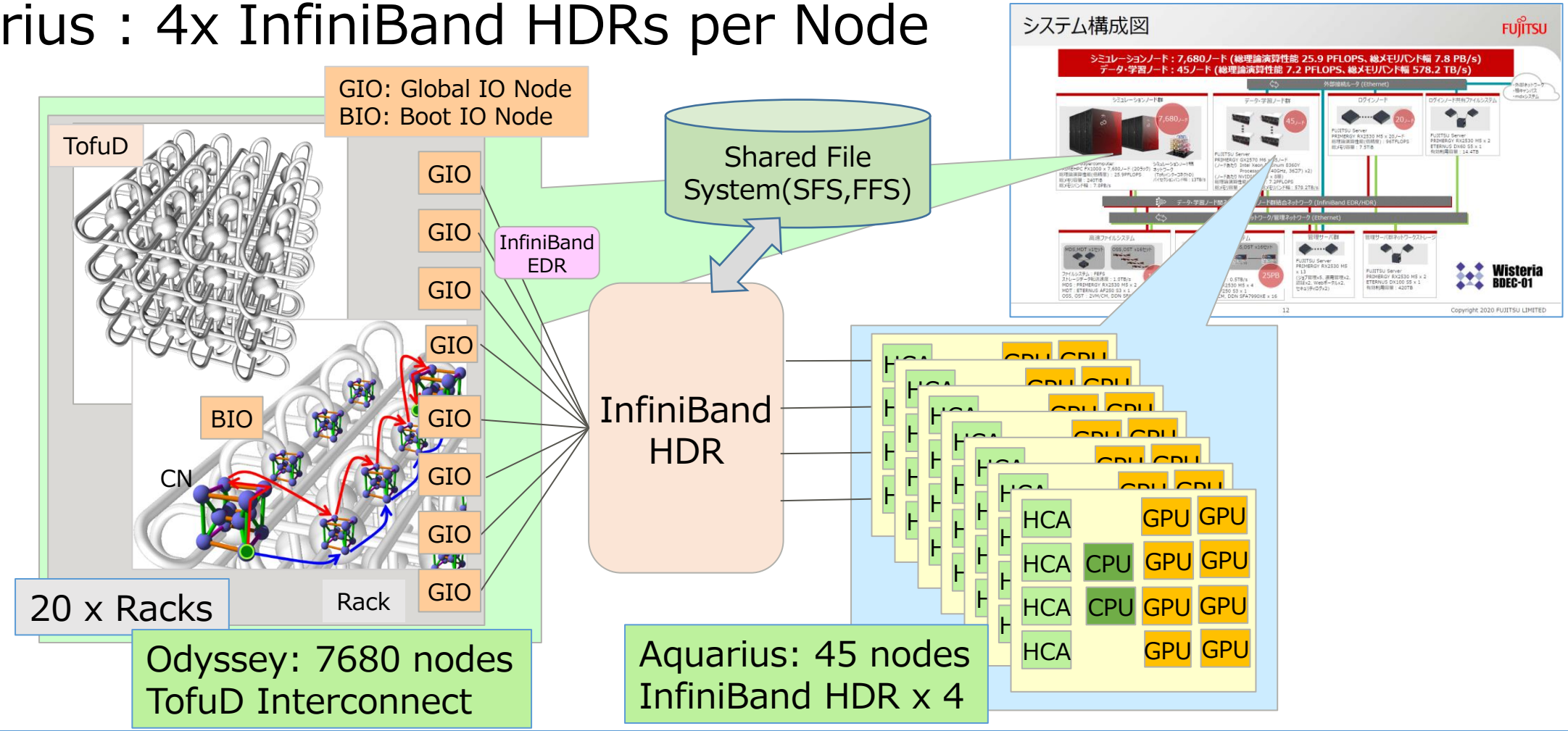
# Evaluation Environments

| | Wisteria/Odyssey | Wisteria/Aquarius |
|---|---|---|
| FLOPS | 25.9 PFLOPS | 7.2 PFLOPS |
| # of Nodes | 7,680 | 45 |
| Interconnect | Tofu-D<br>(6-Dimm Mesh/Torus) | InfiniBand HDR (200Gbps) x 4HCA (Full Fat Tree) |
| Processor/Socket | A64FX(Armv8.1+SVE), 2.2GHz,<br>1 socket (48 Core+2 or 4 Assistant Core) | Intel Xeon Platinum 8360Y, 2.4GHz 2 sockets(36+36) |
| Memory | 32 GiB | 512GiB |
| Memory BW/Node | 1024GB/s | 409.6GB/s |
| GPU/Node | - | NVIDIA A100 x8 |
| | Fujitsu Compiler, Fujitsu MPI | Intel Compiler, Intel MPI (Open MPI) |
| | Red Hat Enterprise Linux 8 | Red Hat Enterprise Linux 8 |

| | Oakbridge-CX (OBCX) | Oakforest-PACS (OFP) |
|---|---|---|
| FLOPS | 6.61 PFLOPS | 25 PFLOPS |
| # of Nodes | 1368 | 8208 |
| Interconnect | Omni-Path (100Gbps) | Omni-Path (100Gbps) |
| Processor/Socket | Intel Xeon Platinum 8280 2.7GHz,<br>2 socket (28+28) | Intel Xeon Phi 7250 1.4GHz<br>1 socket (68) |
| Memory | 192 GB | 96(DDR4)+16(MCDRAM) GB |
| Memory BW/Node | 281.6GB/s | 115(DDR4)+490(MCDRAM) GB/s |
| Compiler, MPI | Intel Compiler, Intel MPI (Open MPI | Intel Compiler, Intel MPI (Open MPI |
| Operating System | Red Hat Enterprise Linux 7, CentOS 7 | Red Hat Enterprise Linux 7, CentOS 7 |

# Wisteria Network Connection

- Odyssey: 8x GIOs with InfiniBand EDR per Rack(384 nodes)
- Aquarius : 4x InfiniBand HDRs per Node



# PingPong 1/2 RTT

Variable proc-16 Node

| usec(8B) | 1/2 RTT(Intra) | 1/2 RTT(Inter) |
|---|---|---|
| Odyssey | 26.8 | 37.1 (Tofu-D) |
| Aquarius | 6.57 | 21.1 (IB) |
| OBCX | 6.85 | 14.5 (OPA) |
| OFP | 49.7 | 83.7 (OPA) |

⇧ good

⇩ bad

- RTT depends on CPU performance: Intranode, Internode
  - Xeon CPU(Aquarius, OBCX): Around 6 usec
  - A64FX、Xeon Phi: 37 – 83.7 usec

# PingPong Bandwidth Performance

Variable proc-16 Node

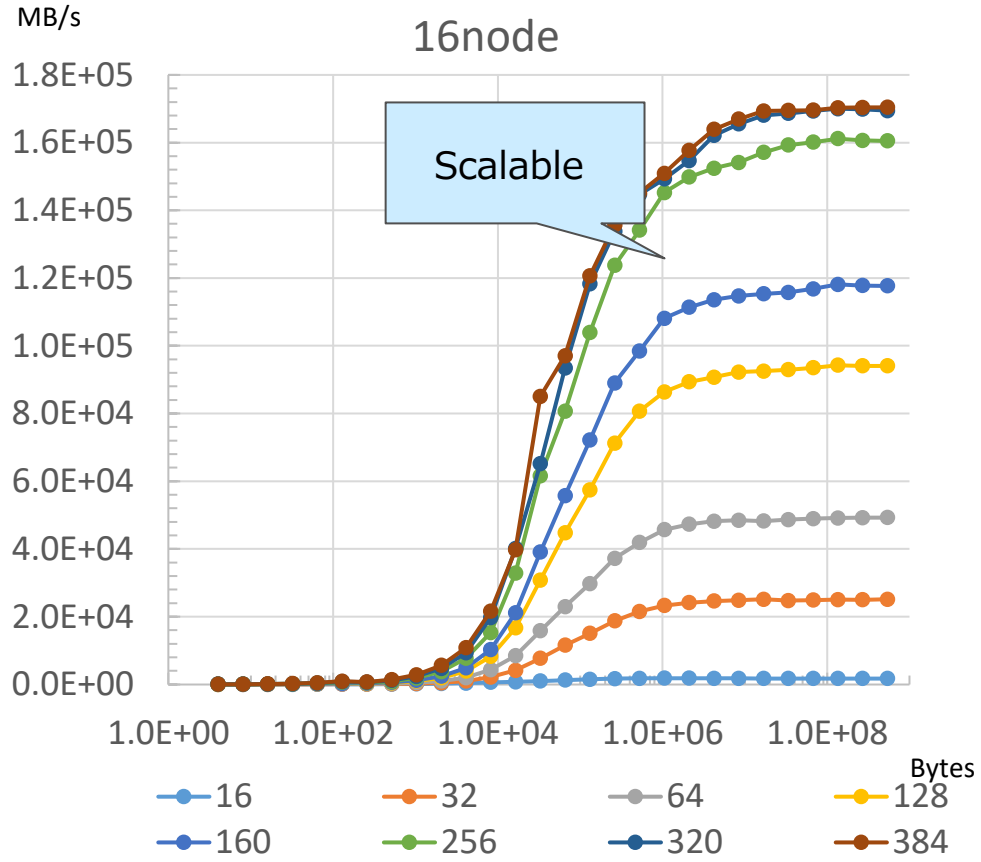
| PingPong BW GB/s | Intra | Inter |
|---|---|---|
| Odyssey | 21.3 | 0.35 (Tofu-D) |
| Aquarius | 259.7 | 27.1 (IB) |
| OBCX | 45.7 | 9.3 (OPA) |
| OFP | 8.2 | 1.12 (OPA) |

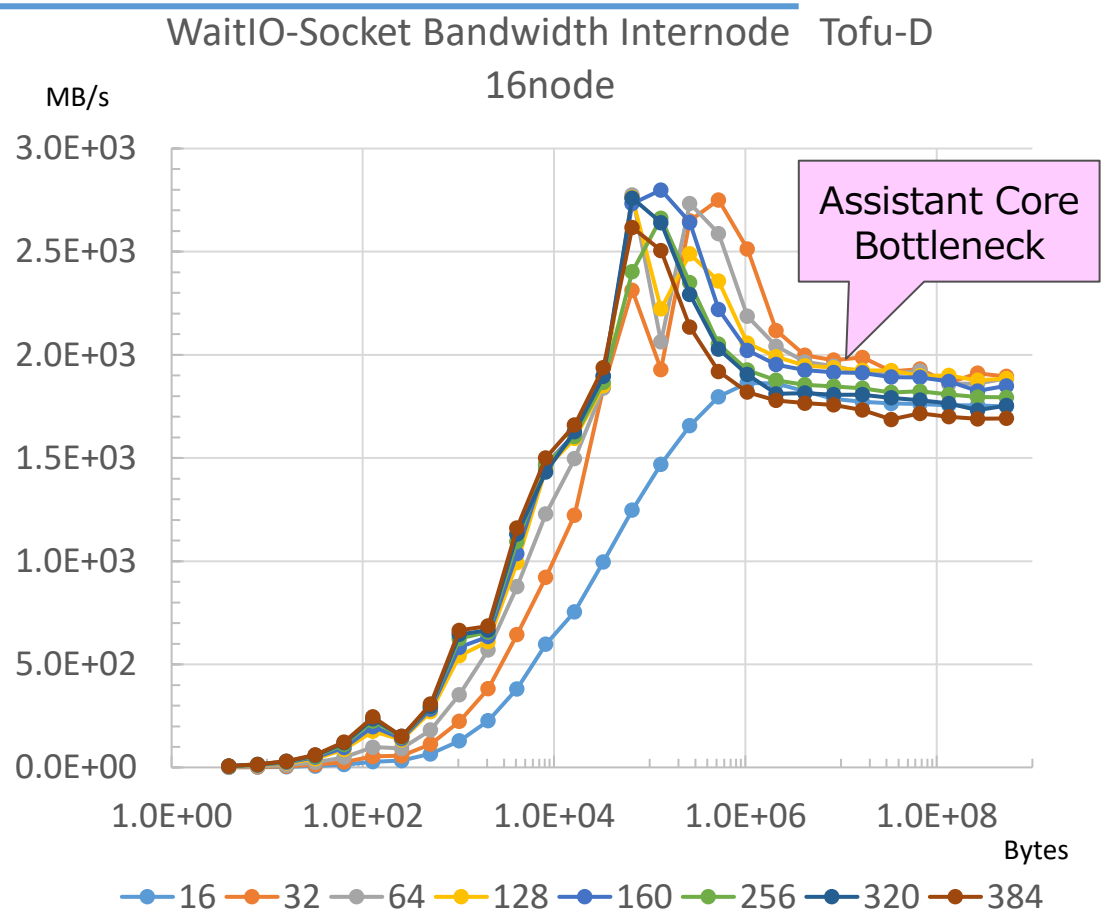
↑ good

↓ bad

- Depends on the balance of CPU and Interconnect performance
  - Odyssey, OFP: CPU performance is not enough
  - Aquarius: Well balanced

# WaitIO-Socket: PingPong Bandwidth Performance on Odyssey 16node



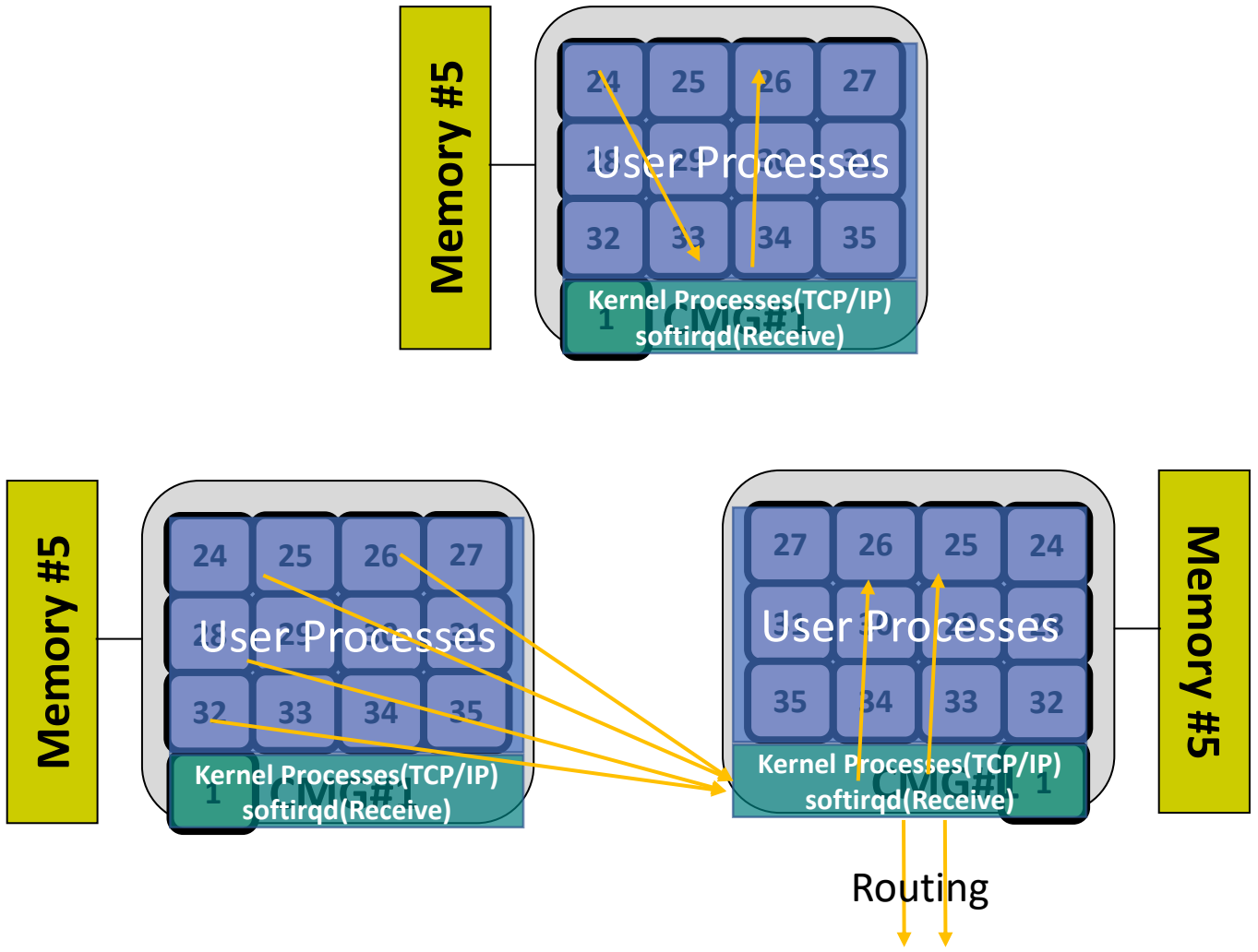




- Intranode: 170.4 GB/s (21.3 GB/s /node)
- Internode: 2.80 GB/s (0.35 GB/s /node)

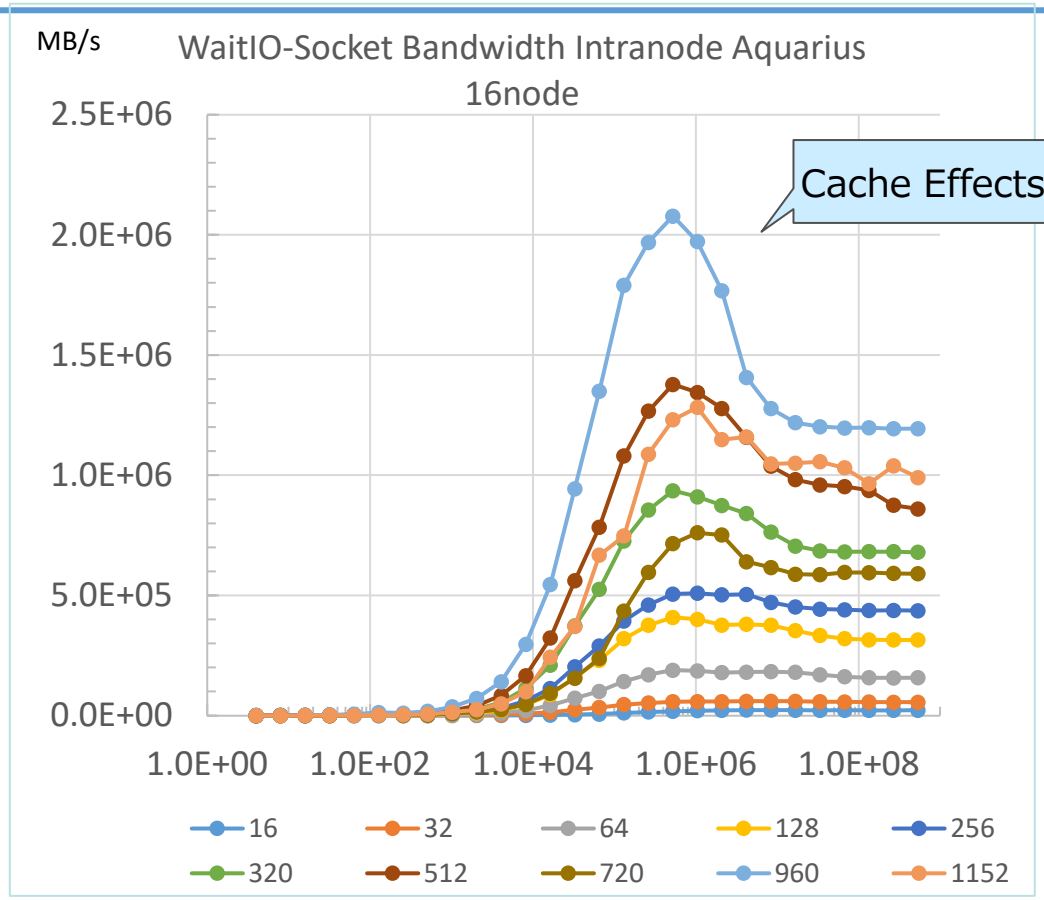
Variable proc-16 Node

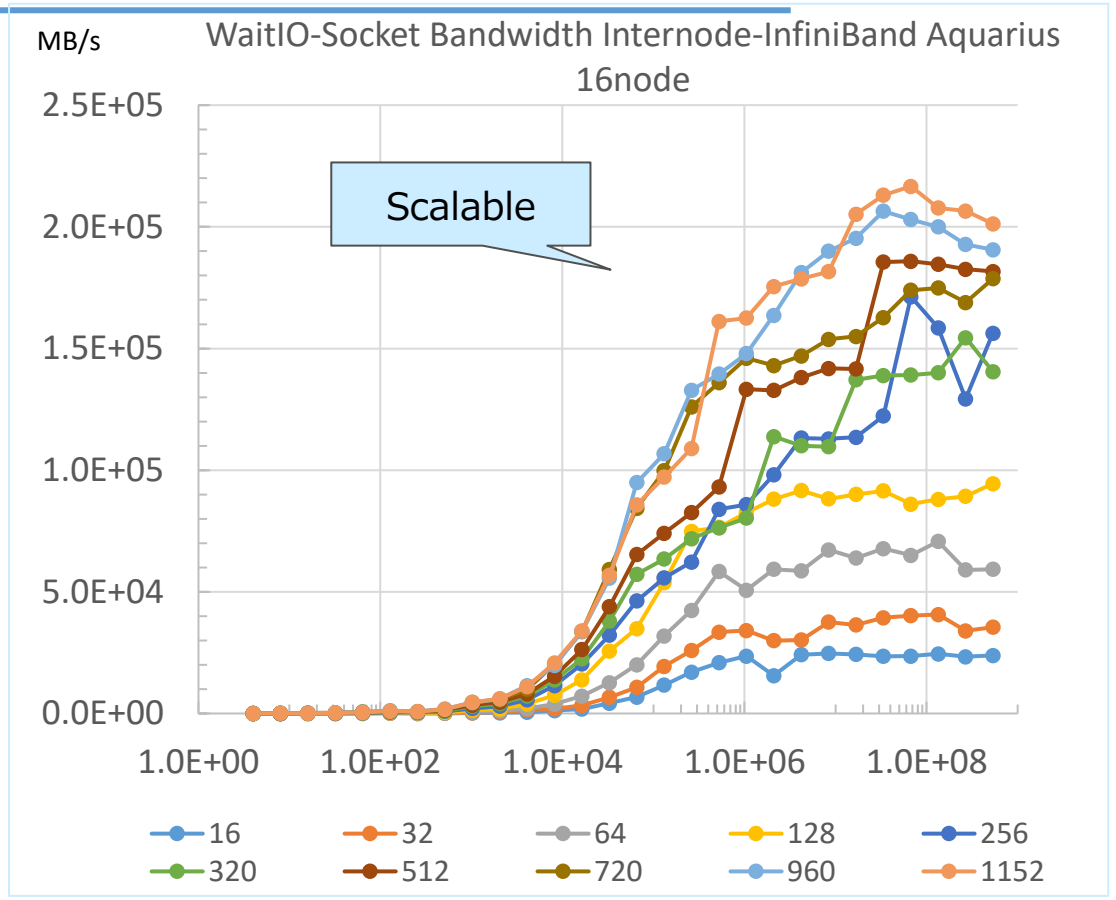
# TCP/IP Processing on FX1000 TCS Software

- Intra-node Communication
  - Processed by Computing Core CPU Copy
- Inter-node Communication
  - Sender: Processed by Computing Core
  - Receiver: Processed by Assistant Core
  - Routing: Processed by Assistant Core on GIO



# WaitIO-Socket: PingPong Bandwidth Performance on Aquarius 16node



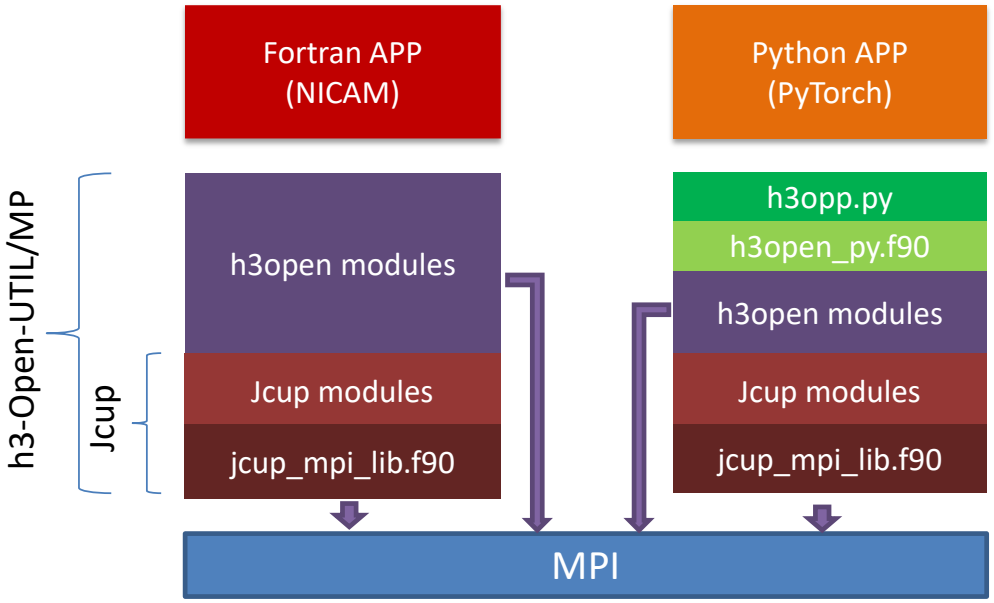



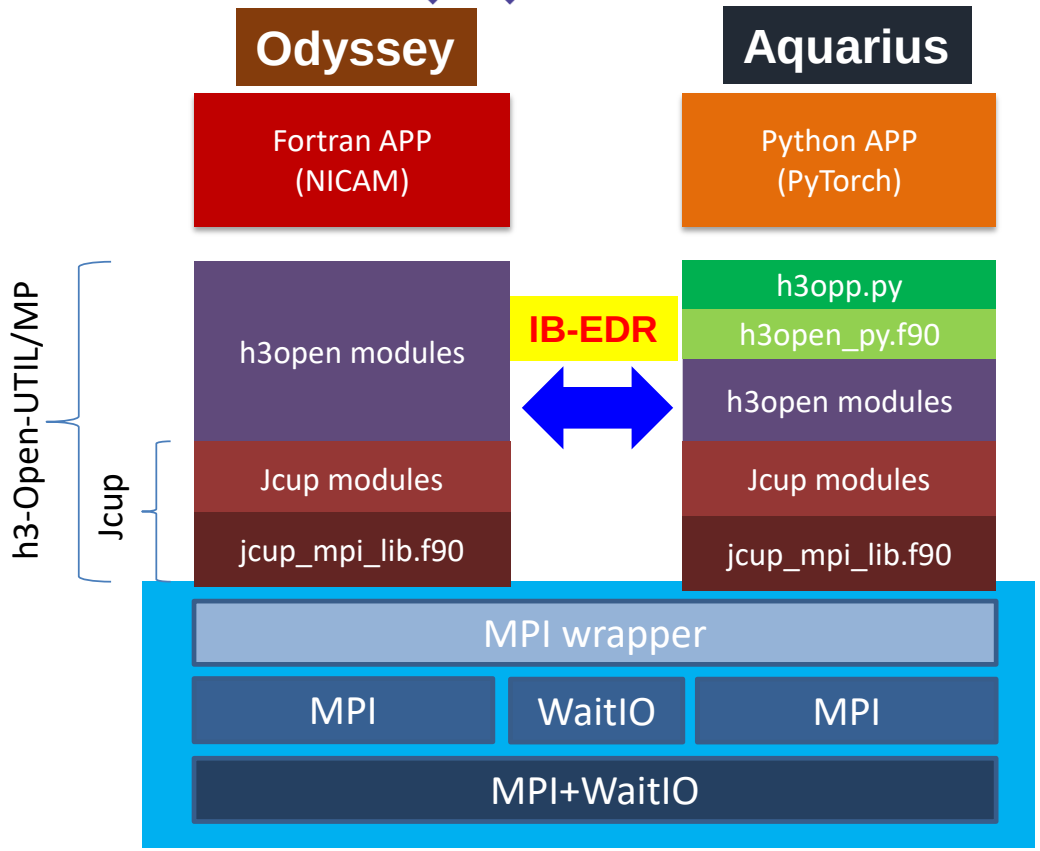

- Intranode: 2,077.6 GB/s (259.7 GB/s /node)
- Internode: 216.6 GB/s (27.1 GB/s /node)

Variable proc-16 Node

# h3-Open-UTIL/MP + h3-Open-SYS/WaitIO-Socket

## Operation Started in June 2022



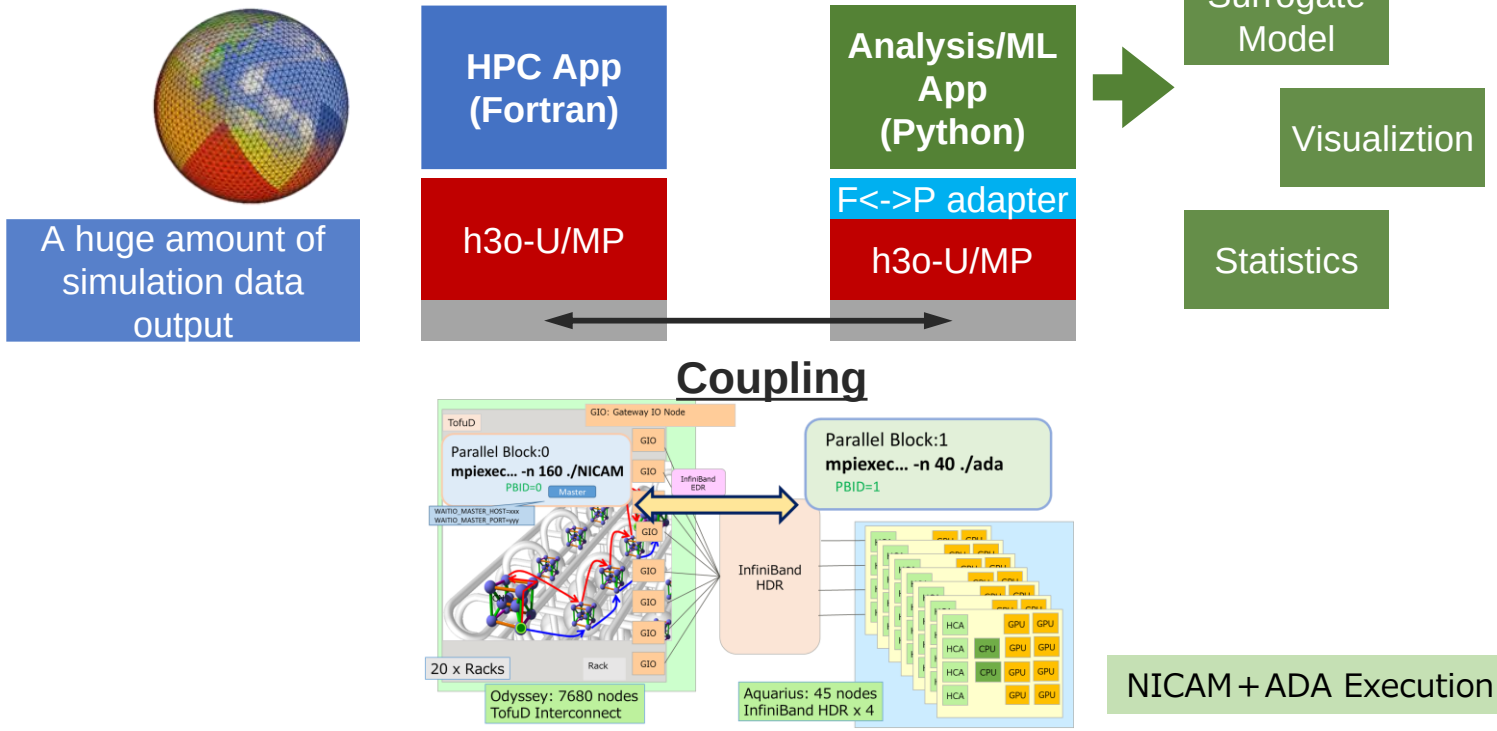




**May 2021: Single MPI Environment**

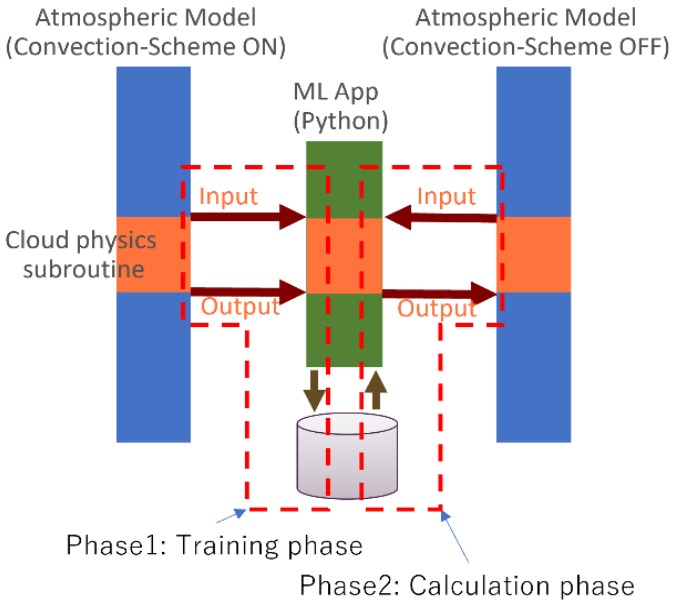
**June 2022: Coupler＋WaitIO**

# h3-Open-UTIL/MP+h3-Open-SYS/WaitIO Application Performance Evaluation

- NICAM-AI(ADA): Performance Evaluation of Coupling Computing
  - Total air density, Internal energy, Density of water vapor
  - Framework : PyTorch, Method : Three-Layer MLP
    - Resolution : horizontal : 10240, vertical : 78



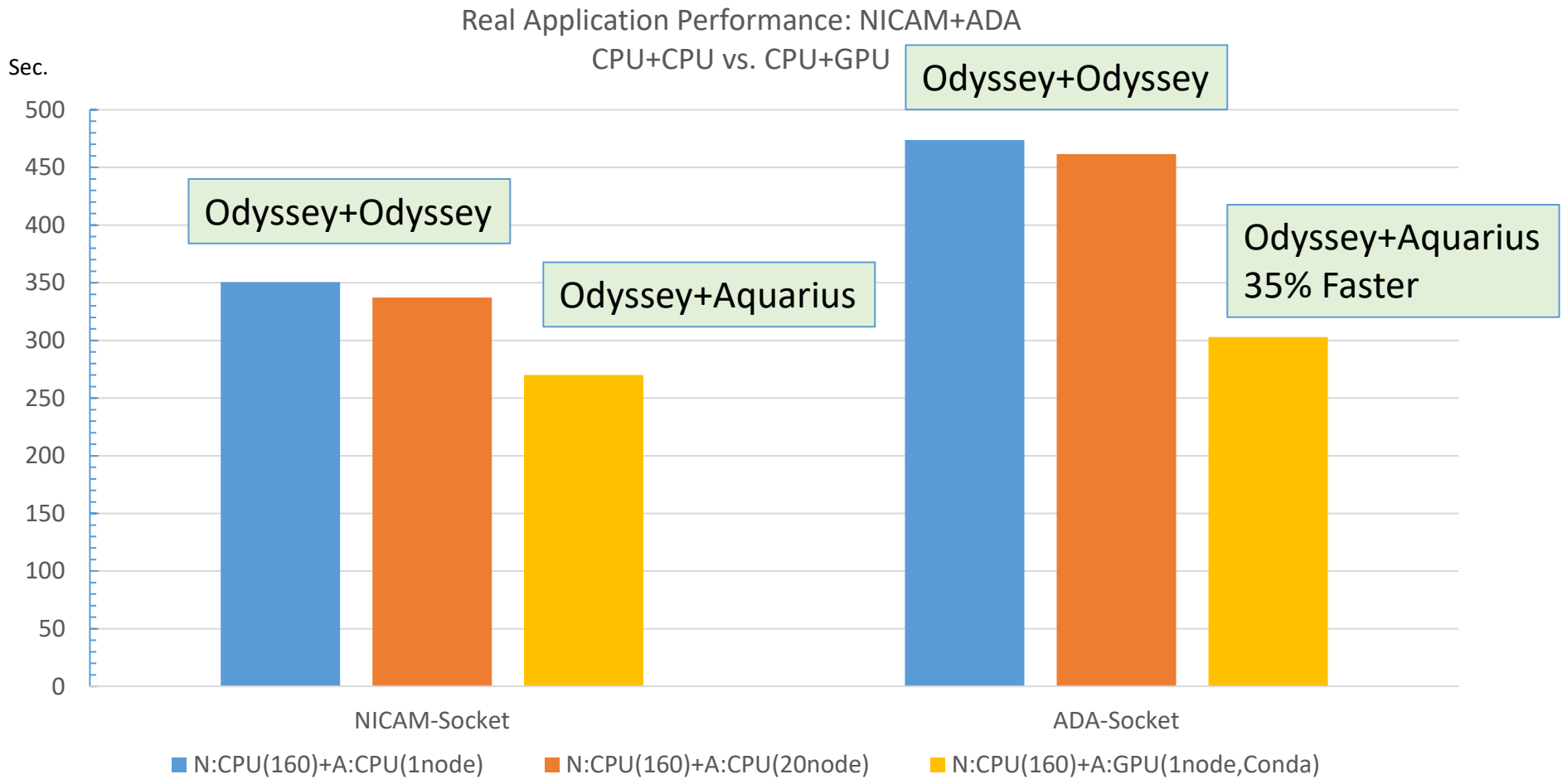

NICAM＋ADA Execution Image




ML Processing Model for Climate Simulation

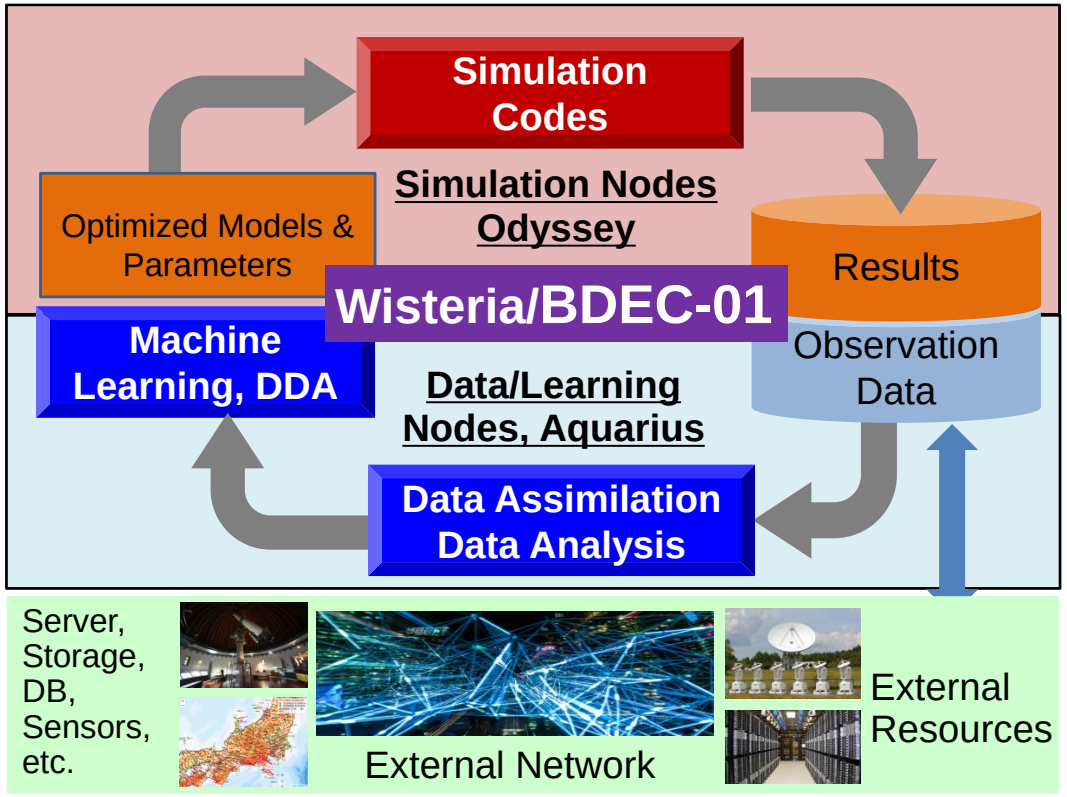
# NICAM+ADA: Performance Results

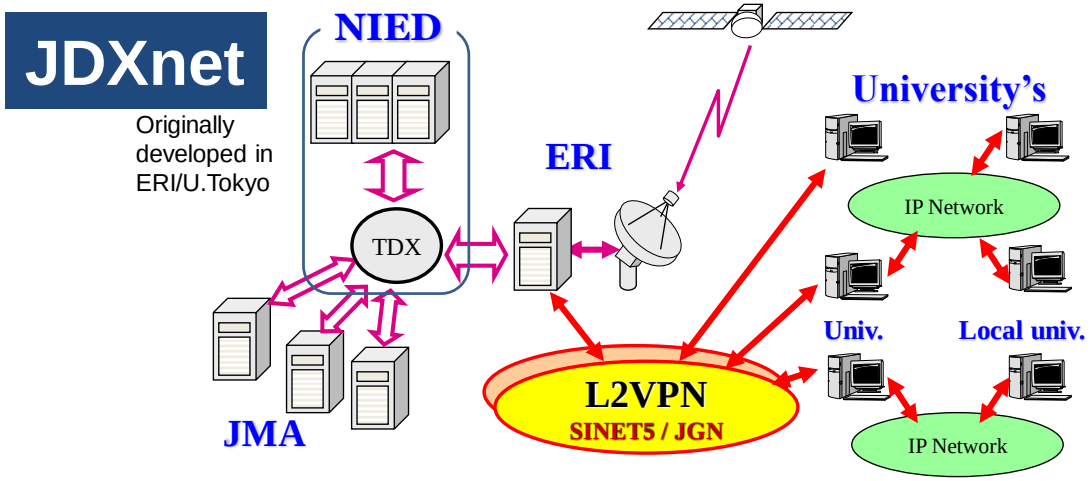
- CPU+GPU is 35% faster than CPU+CPU



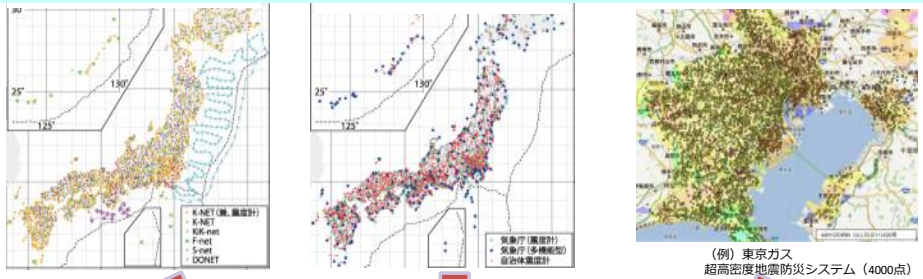
# 3D Earthquake Simulation with Real-Time Data Observation/Assimilation

## Simulation of Strong Motion (Wave Propagation) by 3D FDM




Observation Network for Earthquake: O($10^5$) Points



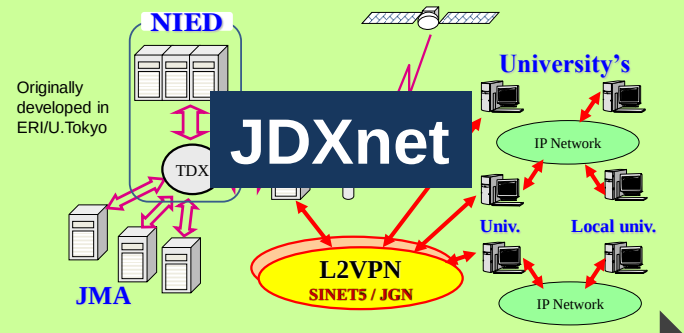

[c/o Furumura]




Real-Time Data/Simulation Assimilation
Real-Time Update of Underground Model

[c/o Prof. T.Furumura (ERI/U.Tokyo)]

# System on Wisteria/BDEC-01 using WaitIO




Observed Data

Obs. Data

SINET5

External Server for Observed Data

Wisteria BDEC-01

Data/Learning Nodes: Aquarius
Intel Ice Lake + NVIDIA A100
7.20 PF, 578.2 TB/s

2.0 TB/s

Simulation Nodes: Odyssey
Fujitsu/Arm A64FX
25.9PF, 7.8 PB/s

Output & Movie

Filter

Visualizer

h3-Open-SYS/WaitIO-Socket

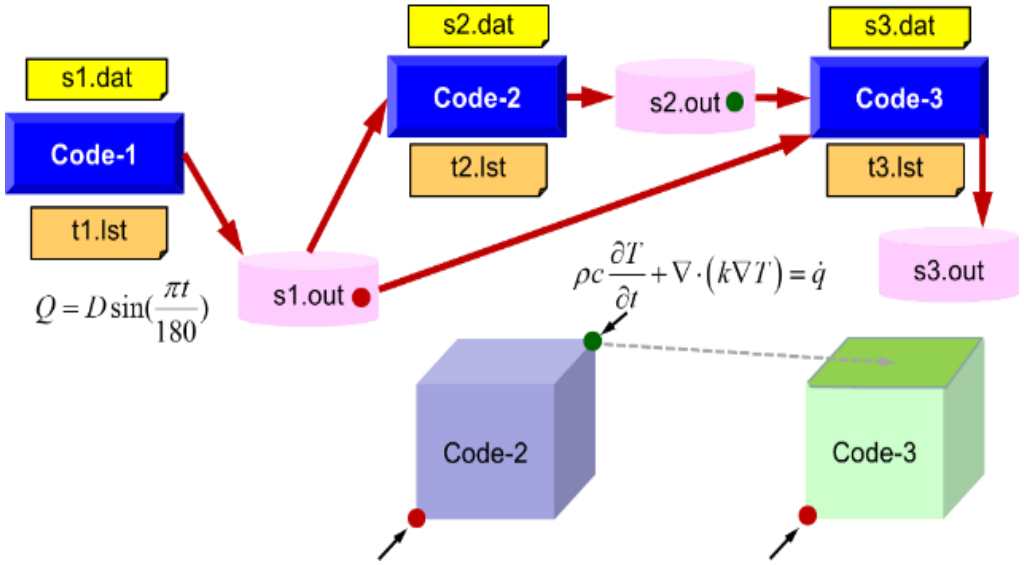
Seism3D/ OpenSWPC-DAF

Mesh Data

UG Model

# WaitIO: Program Conversion Example

- Converting a Sample Coupling Program using File Coupling to WaitIO Coupling
  - Toy Programs of 3D unsteady-state heat transfer problems with the finite element method (FEM) using iterative linear solvers
  - Converting Three Toy Program Coupling from file based to WaitIO Based




| | Code-1 | Code-2 | Code-3 |
|---|---|---|---|
| Computing | Point source calculation | Unsteady three-dimensional heat conduction equation by FEM (MPI) | Unsteady three-dimensional heat conduction equation by FEM (MPI) |
| Input | s1.dat | s2.dat, s1.out | s3.dat, s1.out, s2.out |
| Output | s1.out, t1.lst | s2.out, t2.lst | S3.out, t3.out |

Code-3: Reading Code-2 Output
Forced temperature fixed conditions in the plane of Z= Zmax

# Code-1 Program Source: File Based

- Code-1: Original Code Using File Coupling

```
001     implicit REAL*8(A-H,O-Z)
002     real(kind=8) :: Interval
003     include 'mpif.h'

004     call MPI_Init(ierr)
005     open (11,file='s1.dat',status='unknown')
006     read (11,*) ITERmax, Period, Interval, Val
007     write (*,'(a,i8)')      '##ITERmax ', ITERmax
008     write (*,'(a,1pe16.6)') '##Period  ',  Period
009     write (*,'(a,1pe16.6)') '##Interval',  Interval
010     write (*,'(a,1pe16.6//)') '##Val     ',  Val
011     close (11)

012     open (12,file='s1.out',status='unknown')

013     pi  = 4.d0*datan(1.d0)
014     coef= pi/180.d0
015     time= 0.d0
016     S0time= mpi_wtime  ()
```

```
017     do
018       S1time= mpi_wtime  ()

019       do
020         do iter= 1, ITERmax
021           a= 1.d0
022         enddo
023         E0time= mpi_wtime  (ierr)
024         if ((E0time-S1time).gt.Interval)  exit
025       enddo

026       ttt= E0time - S0time
027       Source= Val * dsin(ttt*coef)
028 !     rewind  (12)
029       write (12,'(2(1pe16.6))') ttt, Source
030     enddo

031     call MPI_Finalize()
032     stop
033     end
```

# Code-1 Program Source: WaitIO Based

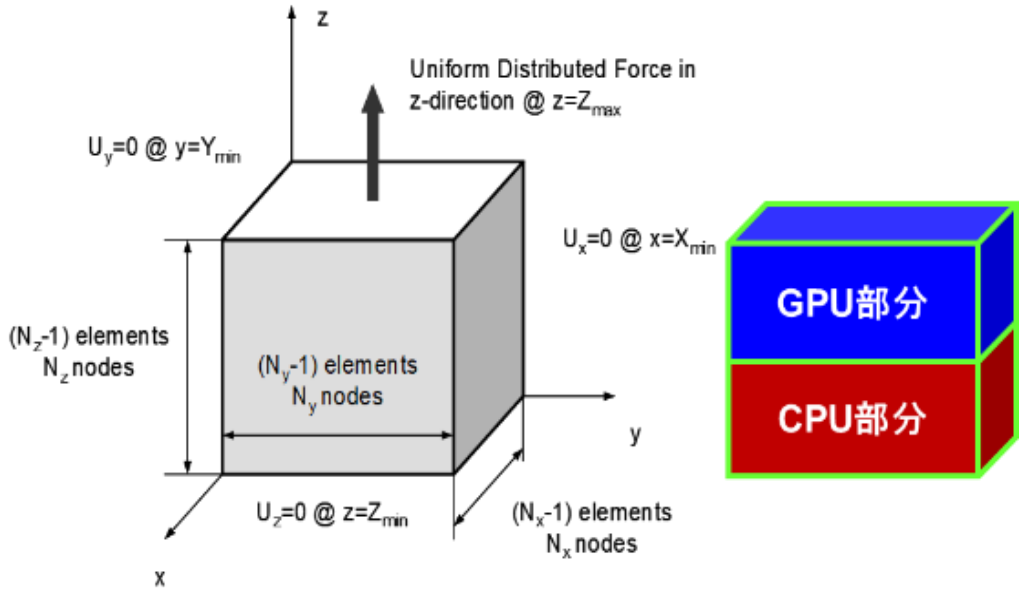
- Converting Open/Read-Write to isend-irecv/wait
  - MPI Non-blocking Send/Recv Style Conversion

```
001     implicit REAL*8(A-H,O-Z)
002     real(kind=8) :: Interval
003     integer :: dest
004     integer(kind=4 ), dimension(:,:), save,allocatable :: sta1
005     integer(kind=4 ), dimension(:,:), save,allocatable :: req1

006     include 'mpif.h'
007     include 'waitio_mpif.h'

008     call MPI_Init(ierr)
009     open (11,file='s1.dat',status='unknown')
010     read (11,*) ITERmax, Period, Interval, Val
011     write (*,'(a,i8)')      '##ITERmax ', ITERmax
012     write (*,'(a,1pe16.6)') '##Period  ',  Period
013     write (*,'(a,1pe16.6)') '##Interval',  Interval
014     write (*,'(a,1pe16.6//)') '##Val     ',  Val
015     close (11)

016     allocate (sta1(WAITIO_STATUS_SIZE,2*2))
017     allocate (req1(WAITIO_REQUEST_SIZE,2*2))
018     call WAITIO_CREATE_UNIVERSE_PBHEAD (WAITIO_SOLVER_COMM, ierr)

019     open (12,file='s1.out',status='unknown')
```

```
037       write (*,'(2(1pe16.6))') ttt, Source
038       do dest=1, 2
039         call WAITIO_MPI_Isend (ttt, 1,
040     &         WAITIO_MPI_DOUBLE_PRECISION,
041     &         dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+1), ierr)
042         if(ierr.ne.0) goto 100
043         call WAITIO_MPI_Isend (Source, 1,
044     &         WAITIO_MPI_DOUBLE_PRECISION,
045     &         dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+2), ierr)
046         if(ierr.ne.0) goto 100
047       enddo
048       call WAITIO_MPI_Waitall (4, req1, sta1, ierr)
049       if(ierr.ne.0) goto 100
050     enddo
051 100  continue
```

# WaitIO Use case: Corporative Processing

- Use case
  - GeoFEM〔18,19〕/Cube Toy Program
    - Co-processing with GPU and CPU by dividing into regions
  - Co-processing with systems using different memory size and number of CPU cores
  - Co-execute code with mixed cases (e.g. linear and non-linear problems) on each suitable system

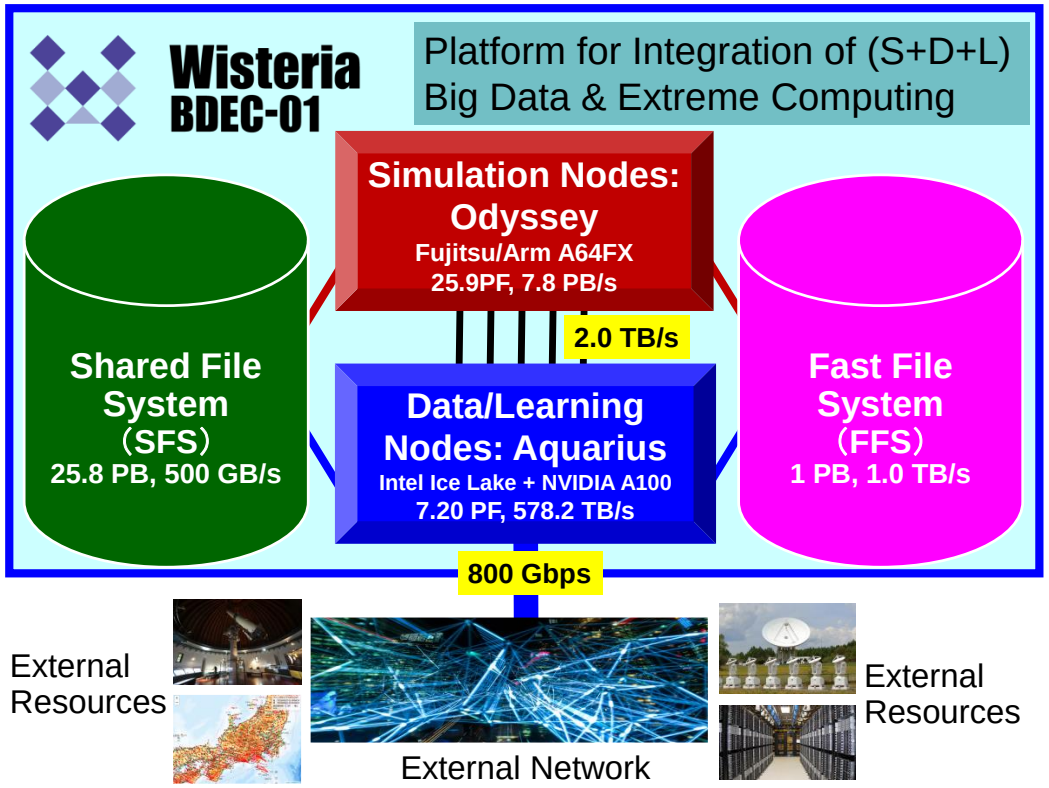


# Related Work

- IMPI, PACX-MPI, GridMPI: Grid computing using TCP/IP. Need to use same implementation
  - IMPI: communication library designed to integrate systems on the Grid.
    - Using Token-based protocol that connects systems called IMPI servers and determines the connection and communication specifications between systems.
    - IMPI also supports secure systems such as firewalls.
  - PACX-MPI: the Global view and Local view to integrate multiple application programs.
    - Transmission/reception processes relay between applications. The relay processes become a performance bottleneck.
  - GridMPI: Supports Grid and Cluster environment in single MPI library: http://aist-itri.github.io/gridmpi/
    - Supporting multiple MPI libraries with vendor MPI and others including IMPI protocol, needs to integrate in MPI level
- MPI standard: MPI_Comm_spawn() and MPI_Comm_connect() /MPI_Comm_accept():
  - Depends on its implementation, ex Open MPI and MPICH, and need to use same implementation and version
- Low-level communication compatible with multiple platforms : UCX, OFED:
  - Supporting Heterogeneous networks heterogeneous processors, and need to use same implementation
- WaitIO-Socket: Works with POSIX-Socket platform for heterogeneous coupling computing
  - Easy to use on existing systems: no needs to install the other software package
  - Scalable: Simple application-to-application direct communication with secure communication such as VPN.
  - Allows the application user to select the MPI process to participate

# Summary of WaitIO(h3-OpenSYS/WaitIO)

- High performance System-wide communication for:
  - Coupling multiple MPI applications among multiple heterogeneous systems
- No extra software needed
  - Works among systems with POSIX Socket communication
    - Existing related work needs same software stack among whole system nodes, so interoperability is limited
  - Existing MPI and compiler can be used
    - Fujitsu MPI, Intel MPI and Open MPI for GPU
- Future Work
  - WaitIO-File for Shared File System
  - Real Application Porting




- Acknowledgement
  - This work is supported by "JSPS Grant-in-Aid for Scientific Research (S) (19H05662)",and by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (jh210022-MDH)".

# Questions?