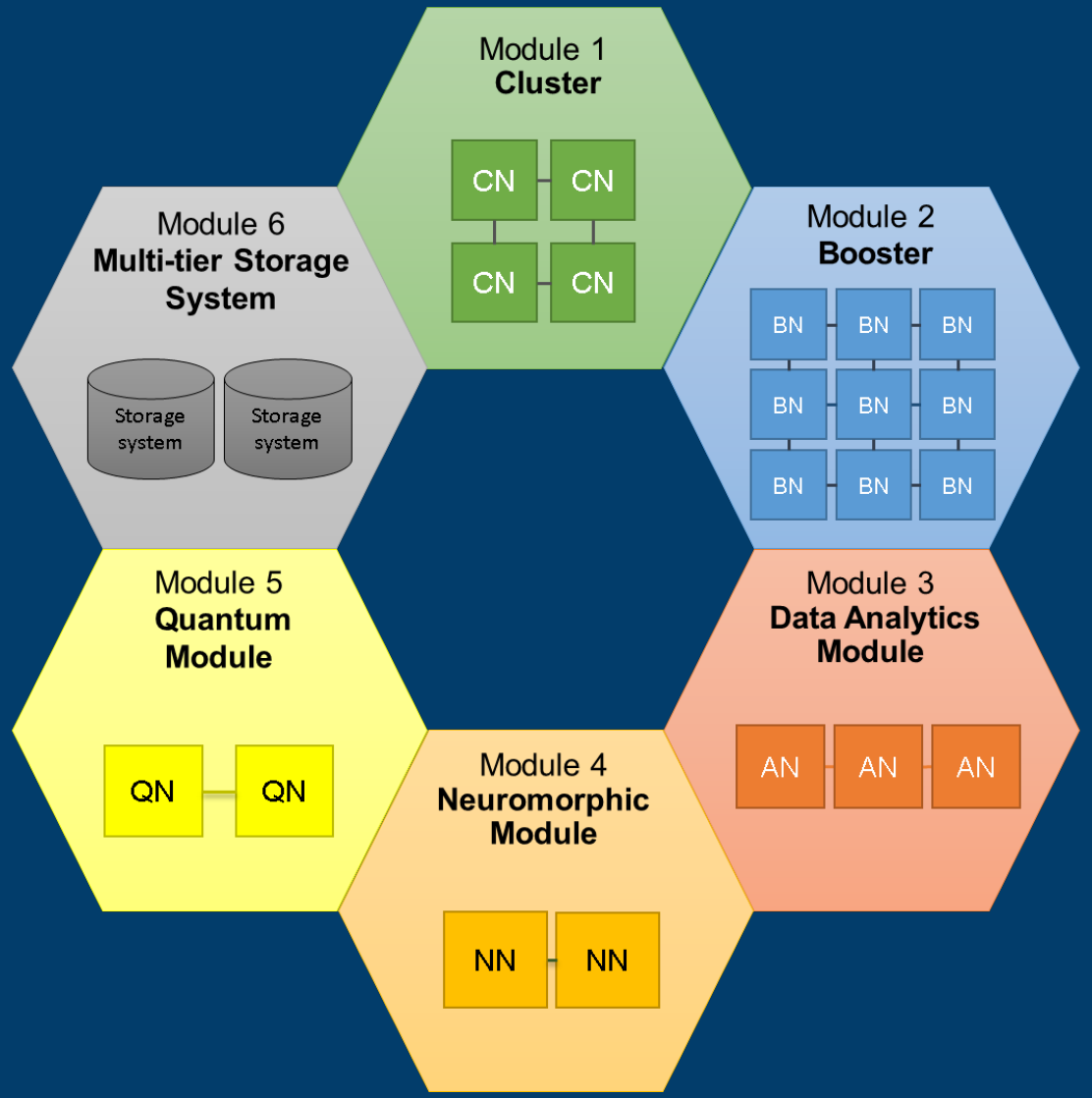
# THE MODULAR SUPERCOMPUTING ARCHITECTURE

## Hardware composability for application diversity

ZIH Colloquium I  Estela Suarez (JSC)

# OUTLINE

- **Evolution of HPC architectures**
  - Global historical evolution
  - Dual architecture at JSC
  - Cluster-Booster
  - Modular Supercomputing Architecture
- **Software**
  - Software stack
  - Network bridging
  - Programming environment
  - Scheduling and resource management
- **Application experience**
- **Conclusions and Next steps**



# Historical evolution in HPC Architectures

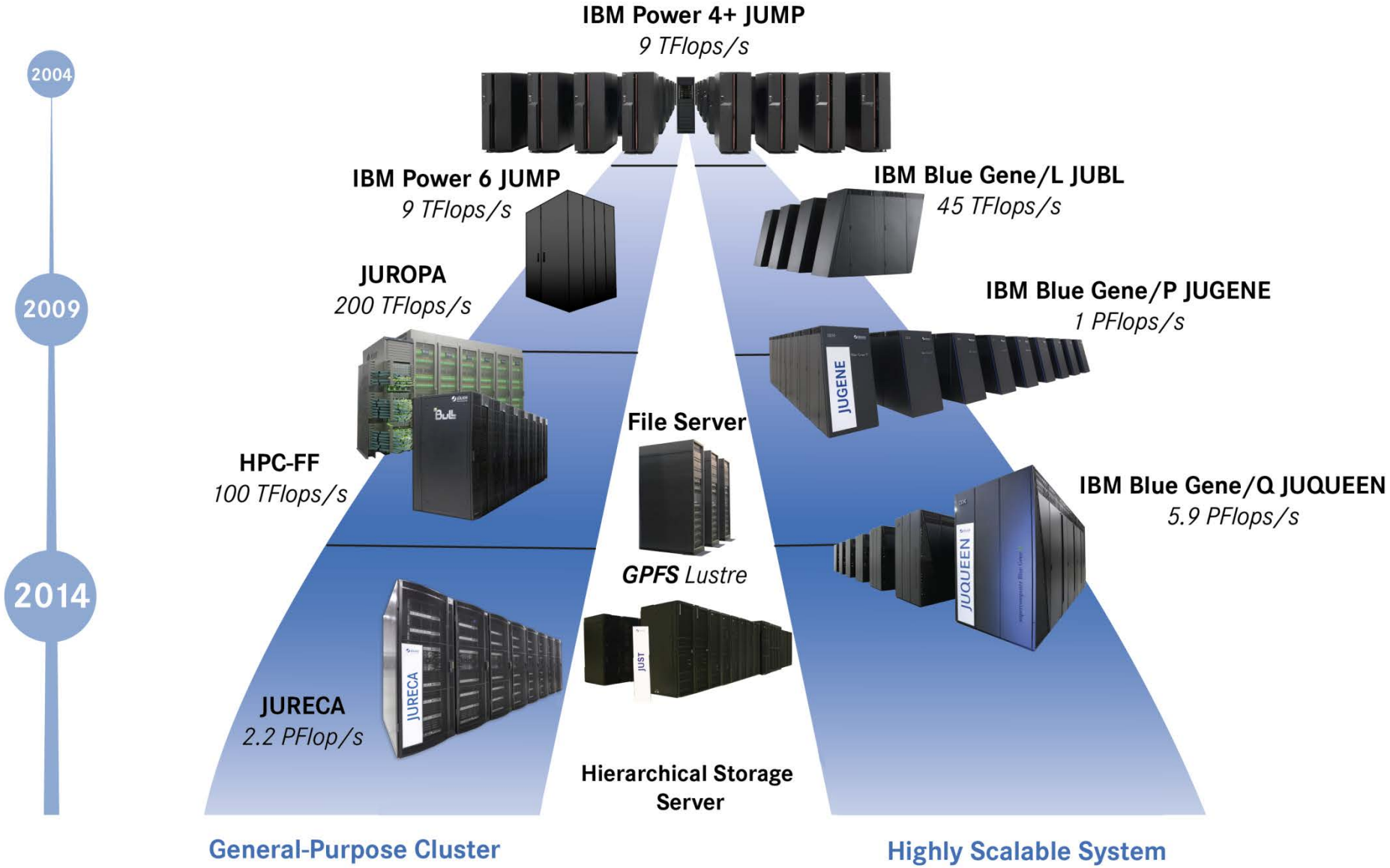
- **1940 – 1950**: first computers are Supercomputers
  - Specialized, very expensive
- **1960 – 1980**: general purpose computers appear
  - Still special machines needed to solve very complex problems
    → Supercomputing (High Performance Computing - HPC)
    - Focus: floating point operations (linear Algebra)
    - Special purpose technologies (fast vector processors, parallel architectures)
    - Only few machines produced
- **1990 – 2000**: integrate standard processors
  - Many „computers“ connected through fast network
    - Distributed memory → MPI
  - Both in proprietary **Massively Parallel (MPP)** and **Cluster Computing**
- **2010 - today**: heterogeneous cluster systems
  - Use accelerator technologies (GPU, many-core)

Turing Bomb
(Source: theregister.co.uk)

Cray-1
(Source: Wikipedia)


JURECA
(Source: FZJ)

# JSC DUAL APPROACH



# Cluster vs. MPP

## Example-systems at JSC

- **General purpose systems**
  - \+ Highly flexible
  - – Relatively large energy consumption
  - \+ Preferred by many applications
    - o Some code parts that could profit from massively parallel system
- **Highly scalable systems (MPP)**
  - \+ Highly energy efficient
  - – Few (highly parallelizable) codes can fully exploit them

Can one combine the best of these two worlds into a single system?

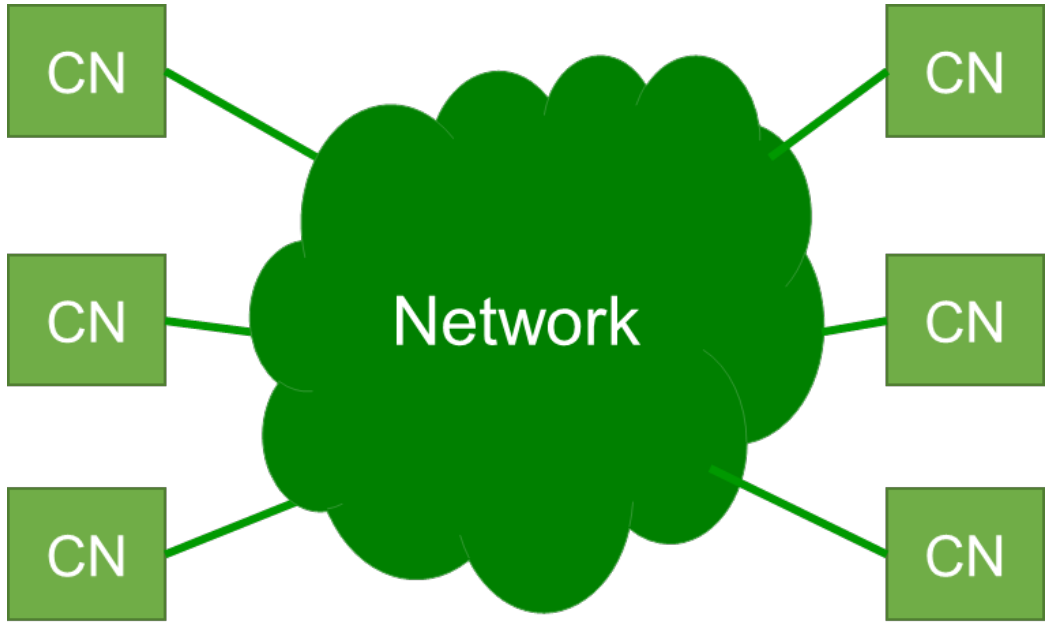
***JURECA Cluster***
**Intel Haswell**
**~ 2,2 PFlop/s**

***JUQUEEN***
**IBM Blue Gene/Q**
**5.9 PFlop/s**


Source pictures: FZJ

# HOMOGENOUS CLUSTER

## General purpose CPUs attached to a high-speed network




+: Easy to use

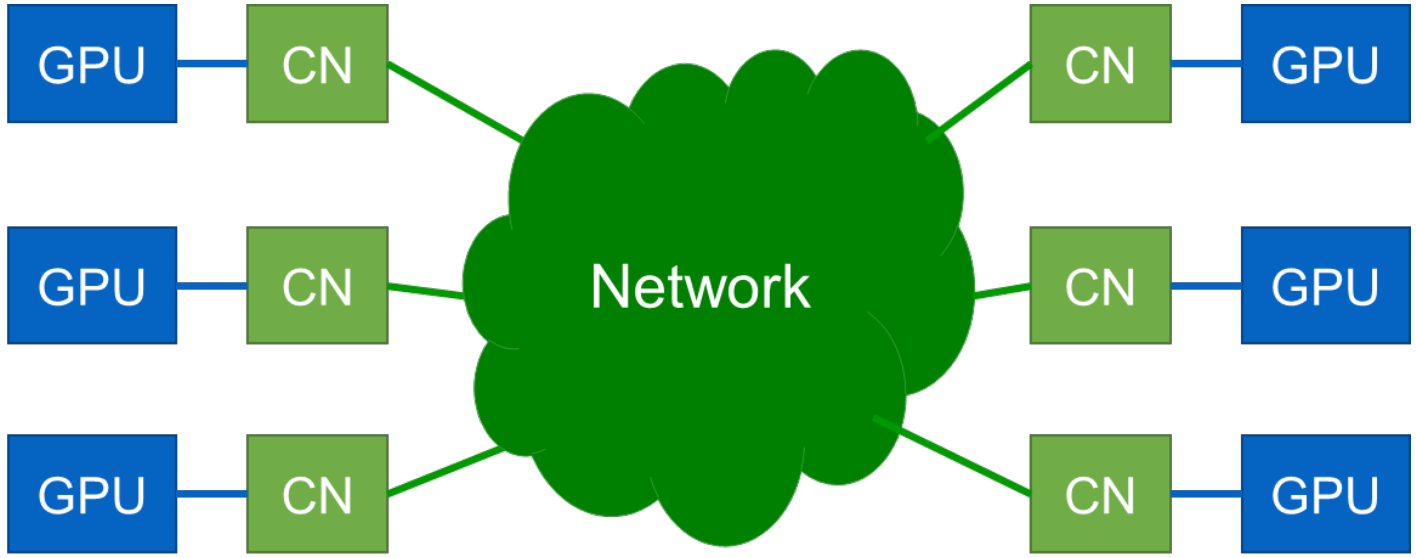
+: Very flexible

–: Power hungry

**CN**: Cluster Node (general purpose processor)

# Traditional HETEROGENEOUS CLUSTER

**Attach accelerators (e.g. GPUs) to each CPU**




+: Energy efficient

+: Easy management

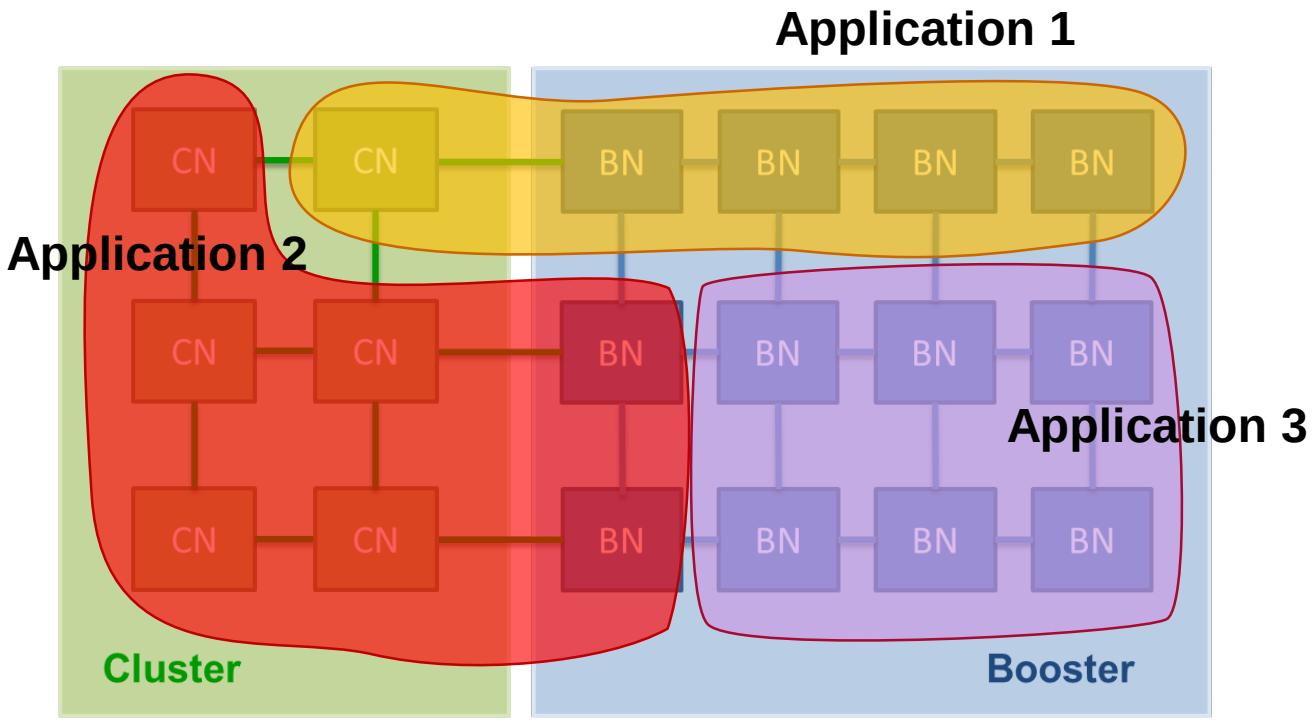
–: Static assignment of accelerators to CPUs

–: Expensive scale-up

**CN**: Cluster Node (general purpose processor)
**GPU**: Graphics Processing Unit (or any other accelerator)

# CLUSTER-BOOSTER concept

## System-level heterogeneity




+: Energy efficient
+: Better scalability
+: High flexibility
+: Dynamic resource assignment

**CN**: Cluster Node (general purpose processor)
**BN**: Booster Node (autonomous accelerator)

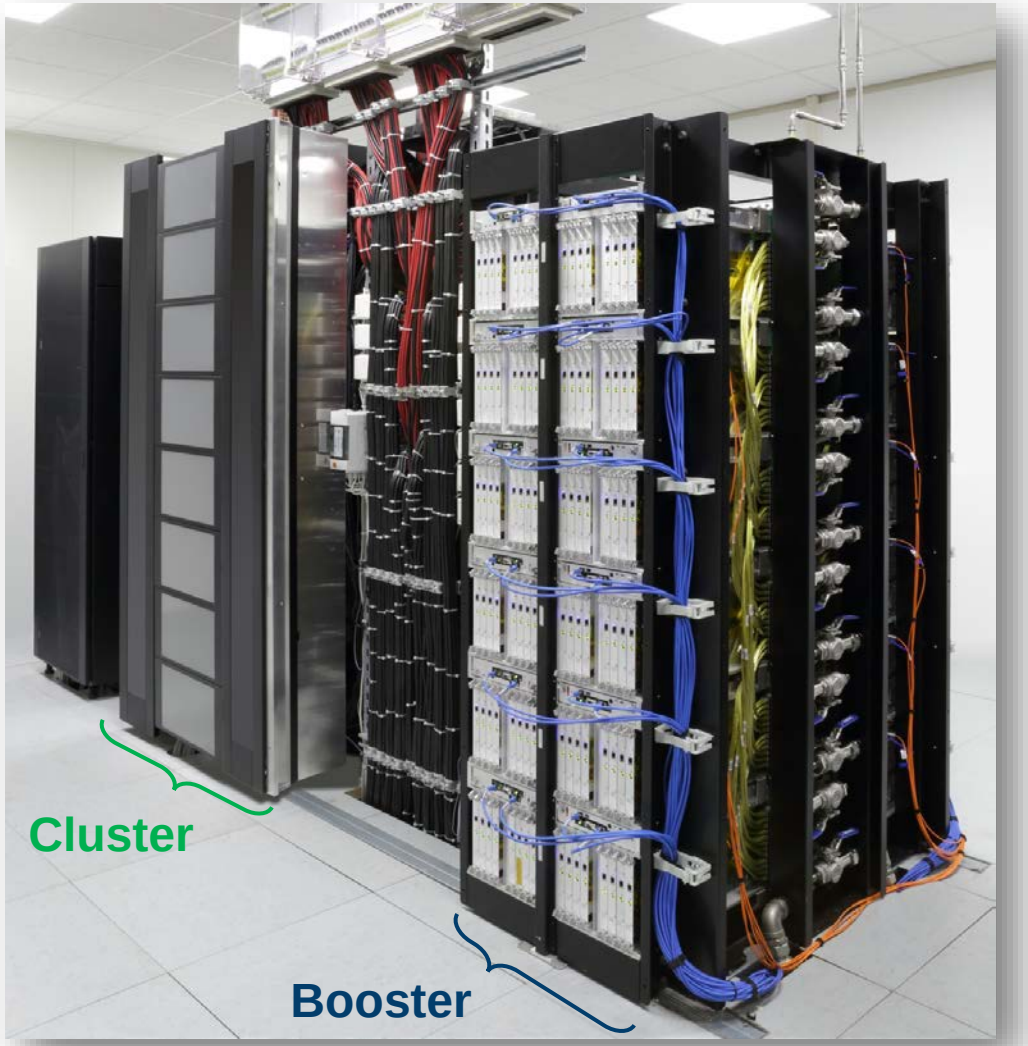
N. Eicker, Th. Lippert, Th. Moschny, and **E. Suarez**, "The DEEP Project - An alternative approach to heterogeneous cluster-computing in the many-core era", Concurrency and computation: Practice and Experience, Vol. 28, p. 2394–-2411 (2016), doi = 10.1002/cpe.3562.

# THE DEEP PROJECTS

- **DEEP** (2011 – 2015)
  - Introduced the Cluster-Booster architecture
- **DEEP-ER** (2013 – 2017)
  - Added I/O and resiliency functionalities
- **DEEP-EST** (2017 – 2021)
  - Extends the concept to a Modular Supercomputer Architecture

27 partners (>80 people)
EU funding: 30 M€
Budget: 45 M€

# THE DEEP PROTOTYPE




| | Cluster | Booster |
|---|---|---|
| Node count | 128 | 384 |
| Processor | Intel Xeon (Sandy Bridge) | Xeon Phi (**KNC**) |
| Cores / node | 8 (×2 socket) | 61 |
| Threads /node | 32 | 244 |
| Frequency | 2,7 GHz | 1,2 GHz |
| Memory (GB) | 32 – RAM | 16 RAM |
| Interconnect | InfiniBand (QDR) | EXTOLL (FPGA) |
| BW | 32 Gbit/s | 20 Gbit/s |
| Peak Perf. | 45 TFlops/s | 500 TFlop/s |

Decommissioned in Summer 2018

# CLUSTER-BOOSTER in Production

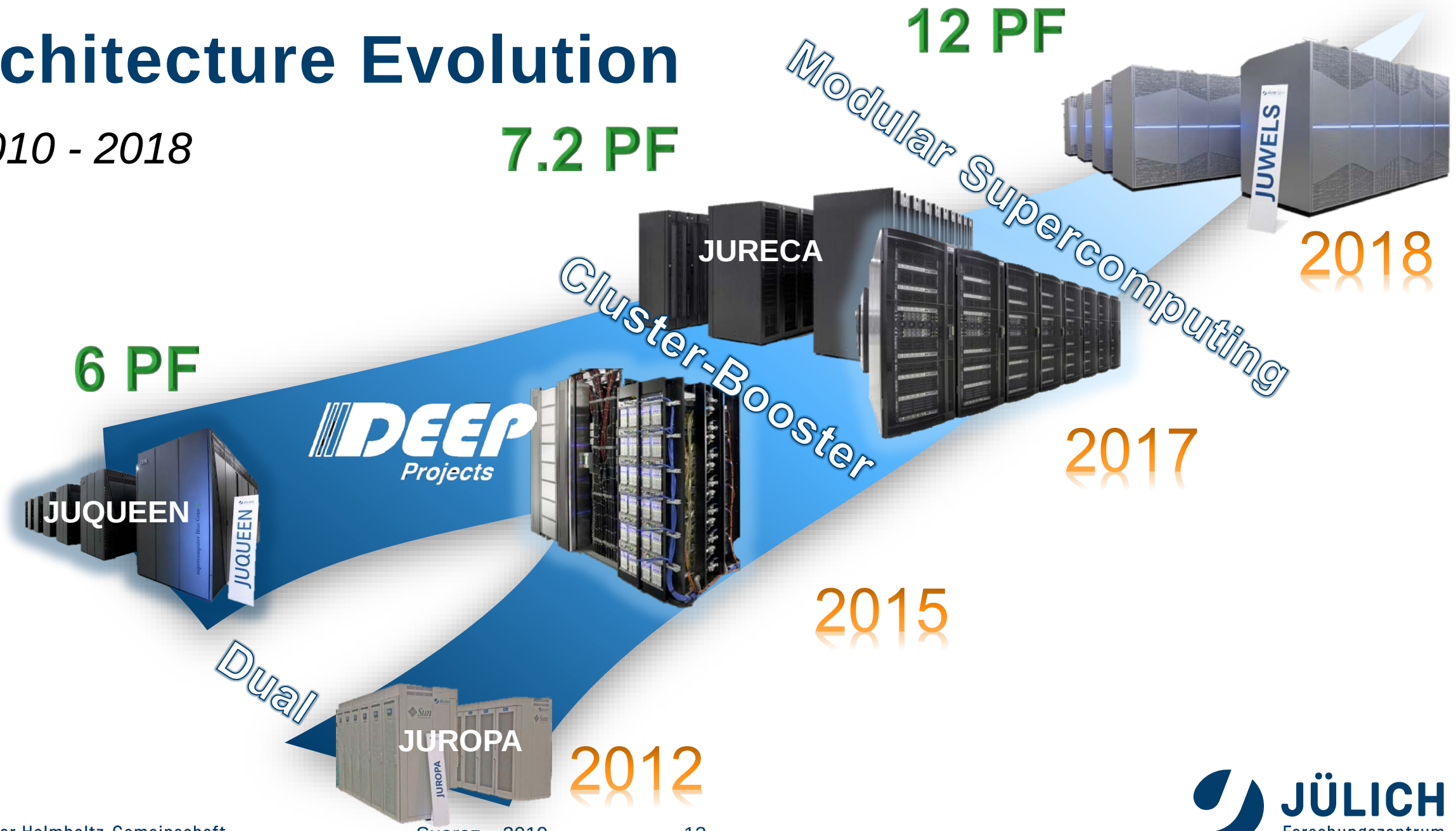
a) JURECA Cluster


b) JURECA Booster

| | Cluster | Booster |
|---|---|---|
| Processor | Intel Xeon (Haswell) | Xeon Phi (KNL) |
| Interconnect | InfiniBand EDR | OmniPath |
| Node count | 1,872 | 1,640 |
| Peak Perf. (PFlops) | 1,8 (CPU) + 0.4 (GPU) | 5 |

# Architecture Evolution

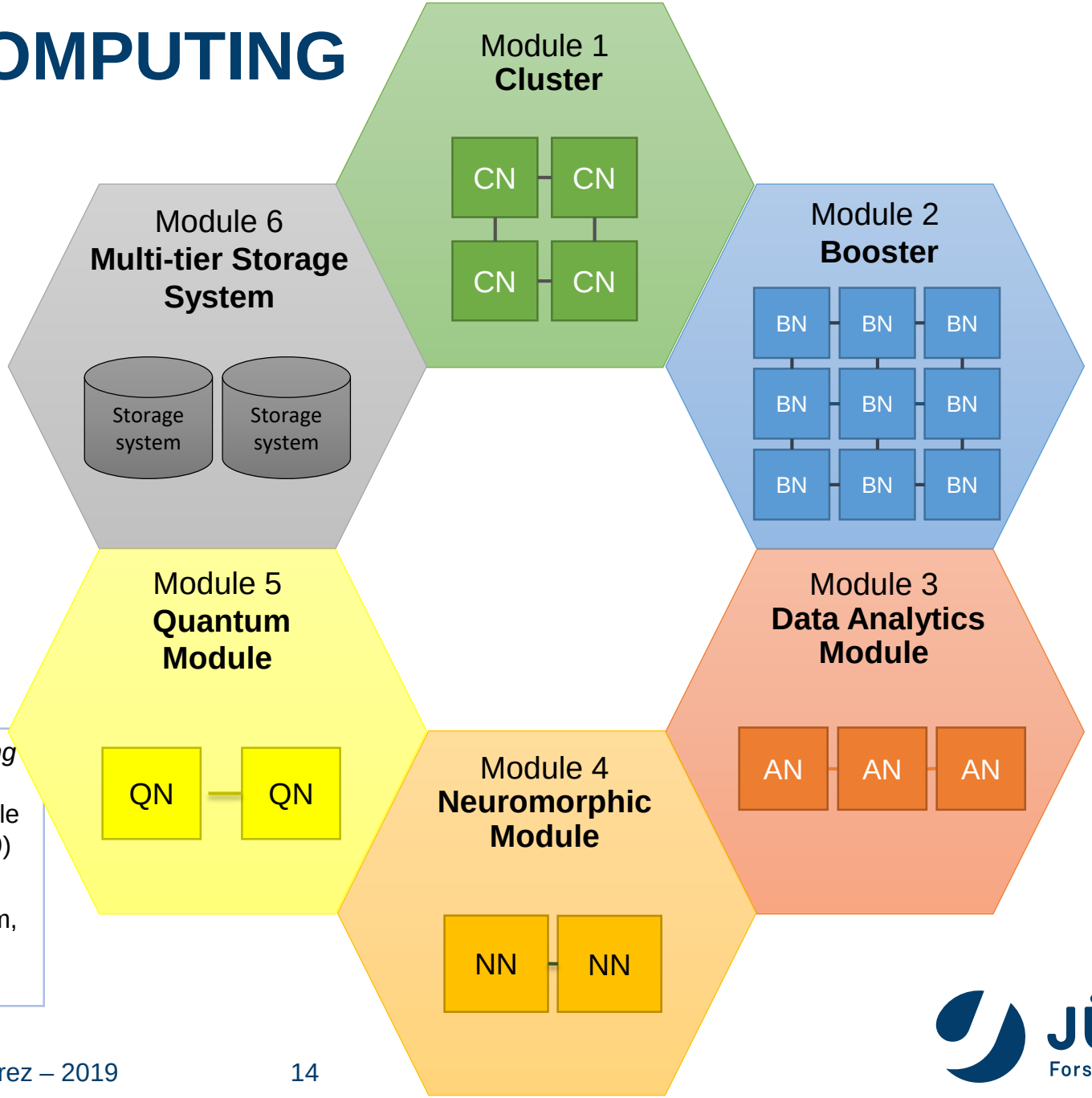
*2010 - 2018*



# MODULAR SUPERCOMPUTING

## Composability of heterogeneous resources

- Cost-efficient scaling
- Effective resource-sharing



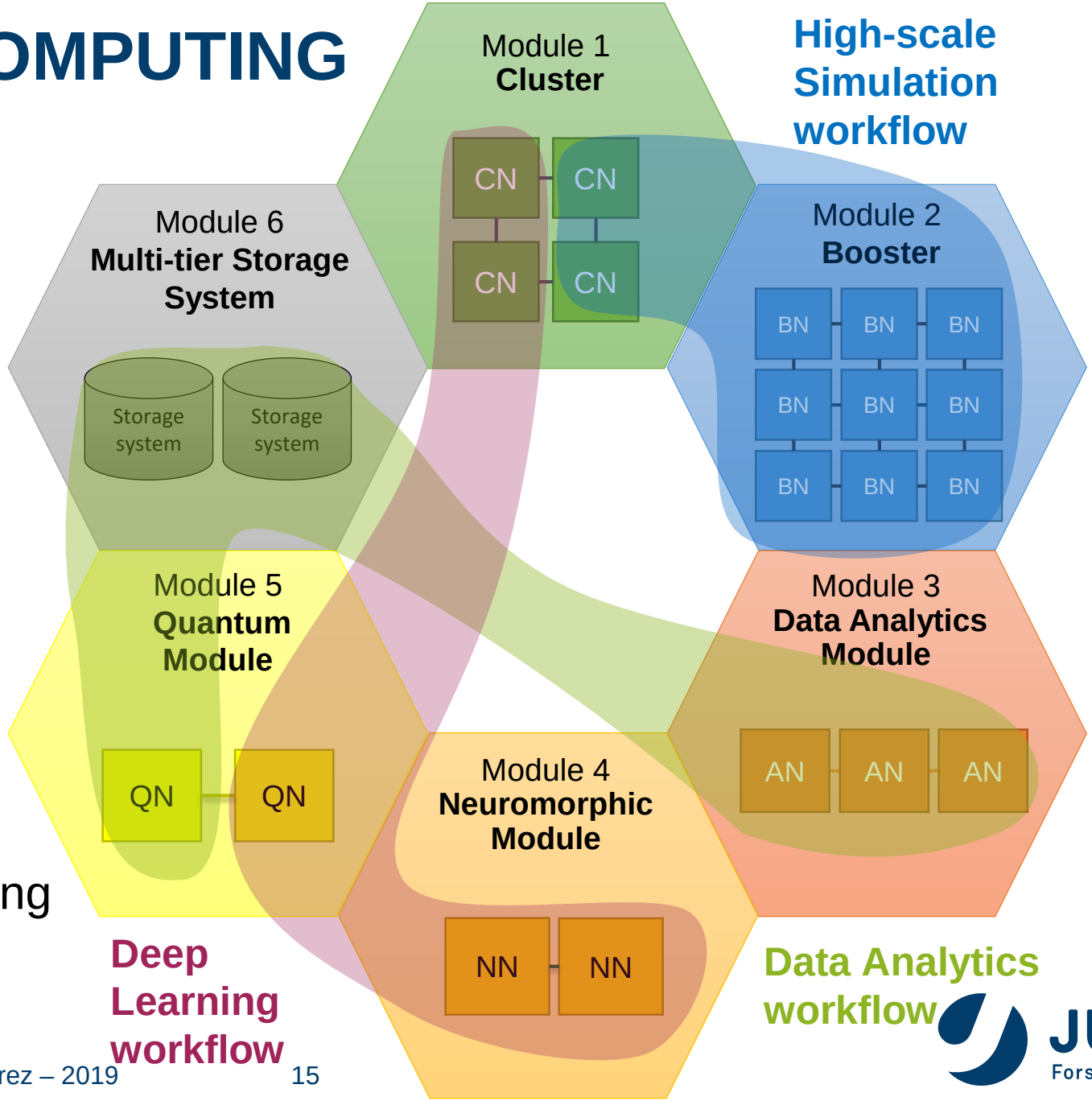


- **E. Suarez***, N. Eicker, Th. Lippert, "*Modular Supercomputing Architecture: from idea to production*", Chapter 9 in Contemporary High Performance Computing: from Petascale toward Exascale, Volume 3, pp 223-251, CRC Press. (2019)
- **E. Suarez***, N. Eicker, and Th. Lippert, "Supercomputer Evolution at JSC", Proceedings of the 2018 NIC Symposium, Vol.49, p.1-12, (2018) [online: http://juser.fz-juelich.de/record/844072].

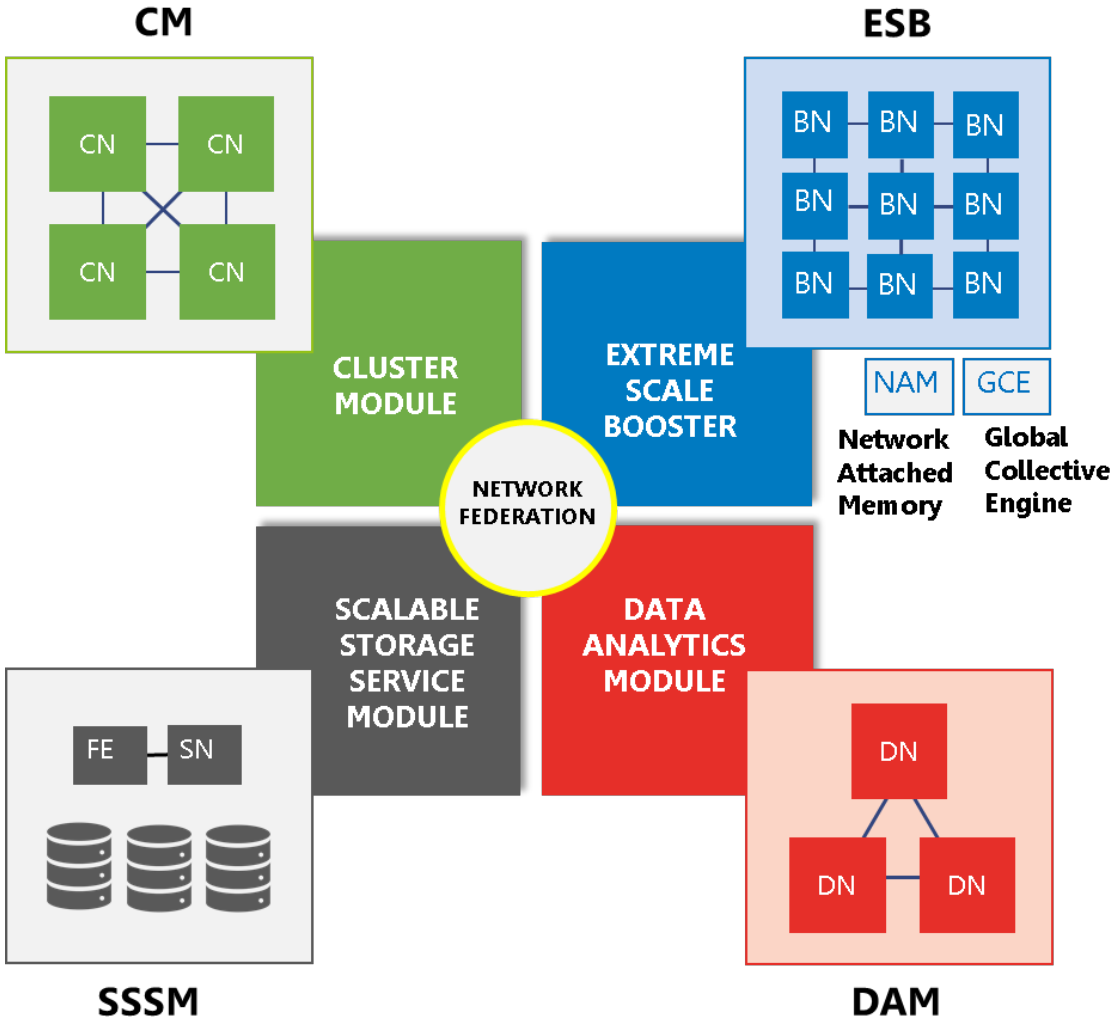
# MODULAR SUPERCOMPUTING

## Composability of heterogeneous resources

- Cost-efficient scaling
- Effective resource-sharing
- Fit application diversity
  - Large-scale simulations
  - Data analytics
  - Machine- and Deep Learning
  - Artificial Intelligence



# DEEP-EST prototype



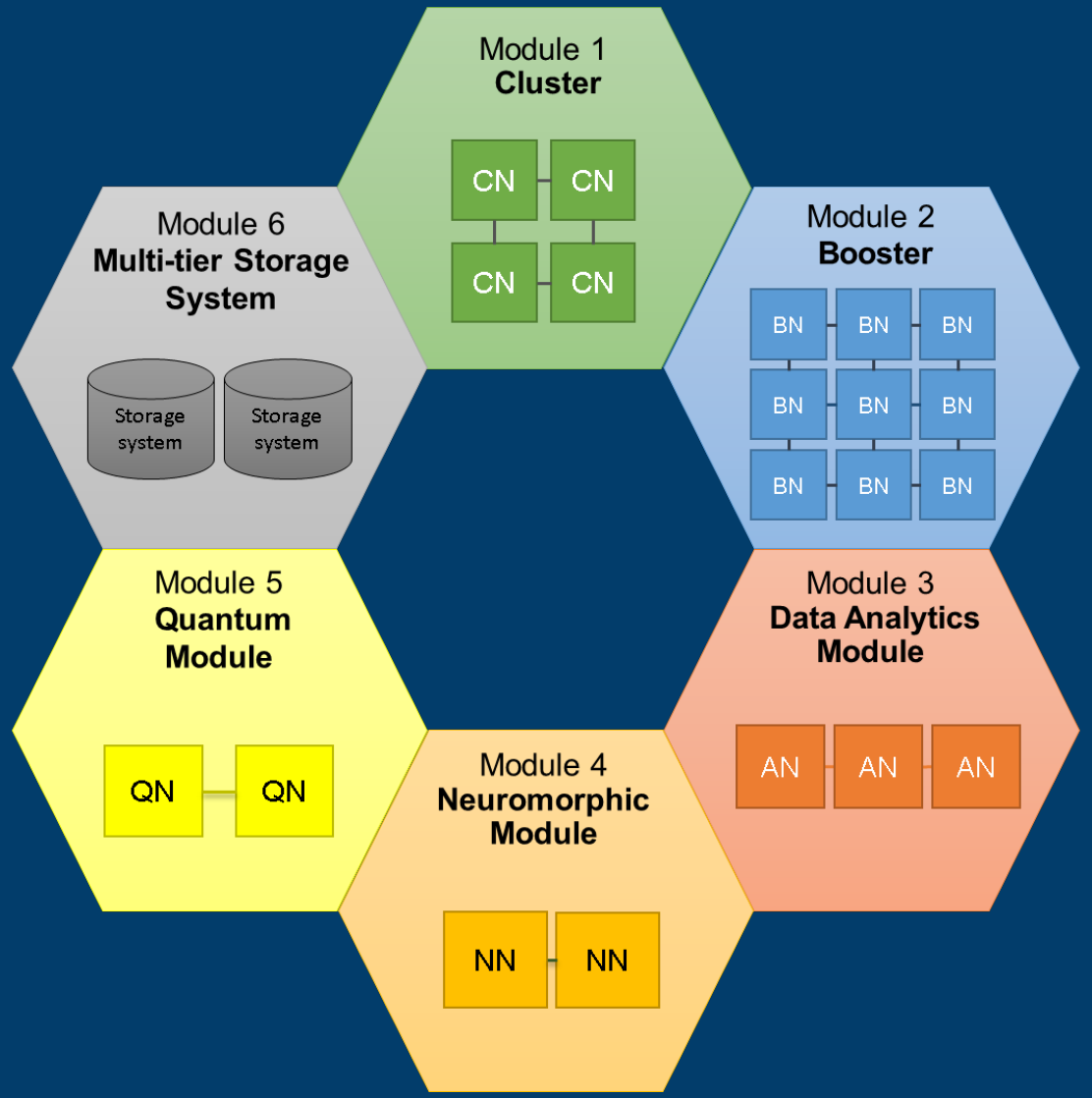

Source: FZJ

Prototype co-designed with Software and Applications

Early-Access program in 2020!

# OUTLINE

- **Evolution of HPC architectures**
  - Global historical evolution
  - Dual architecture at JSC
  - Cluster-Booster
  - Modular Supercomputing Architecture
- **Software**
  - Software stack
  - Network bridging
  - Programming environment
  - Scheduling and resource management
- **Application experience**
- **Conclusions and Next steps**



# SOFTWARE ENVIRONMENT

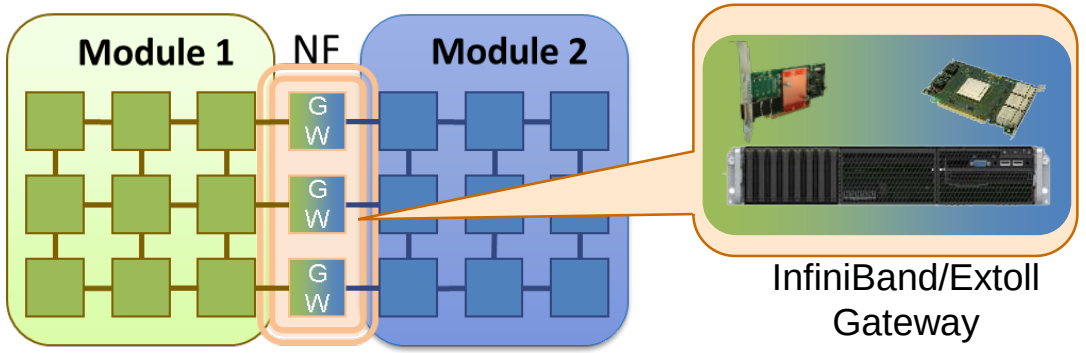


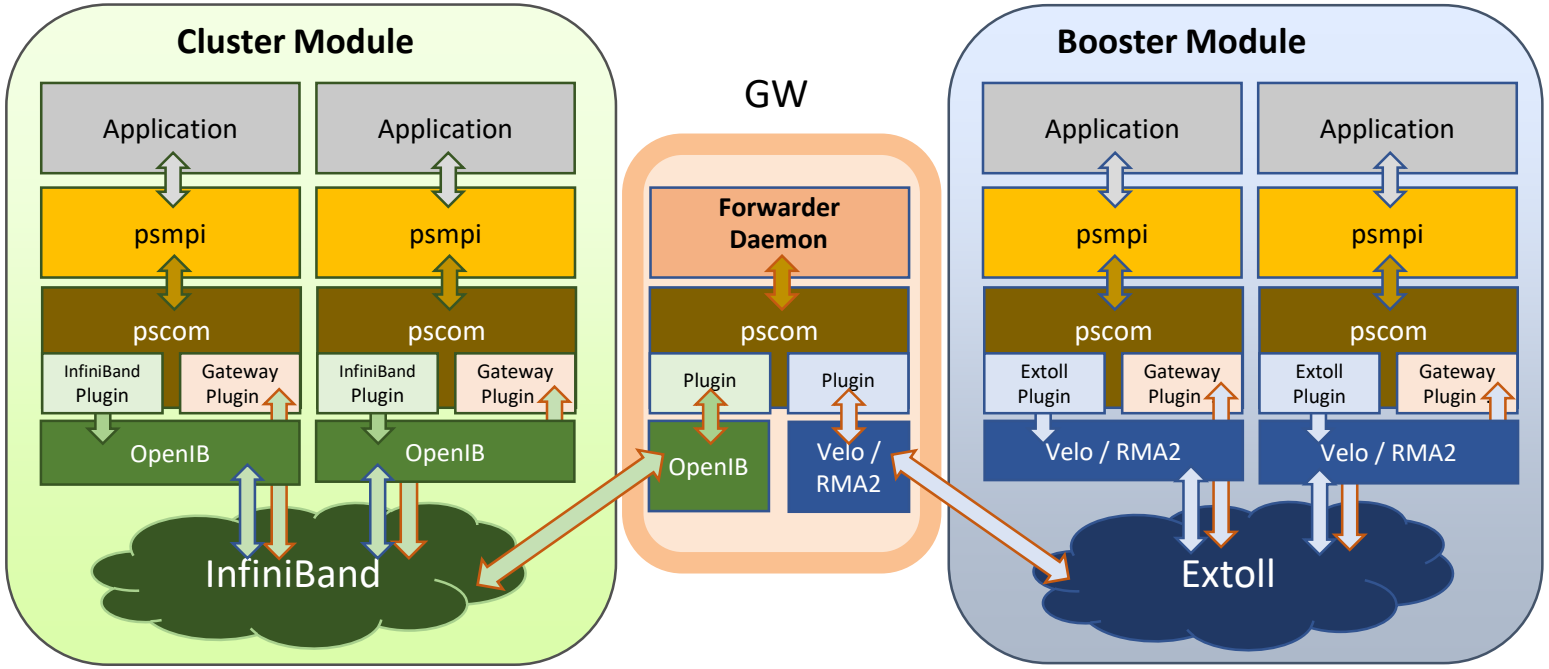



- **Low-level SW:** Inter-network bridging
- **Scheduler**: Torque/Maui → SLURM
- **Filesystem**: BeeGFS
- **Compilers**: Intel, gcc, PGI
- **Debuggers**: Intel Inspector, TotalView
- **Programming**: ParaStation MPI (mpich), OpenMP, OmpSs
- **Performance analysis tools**: Scalasca, Extrae/Paraver, Intel Advisor, VTune…
- **Benchmarking tools**: JUBE
- **Libraries**: SIONlib, SCR, HDF5…

# NETWORK BRIDGING

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





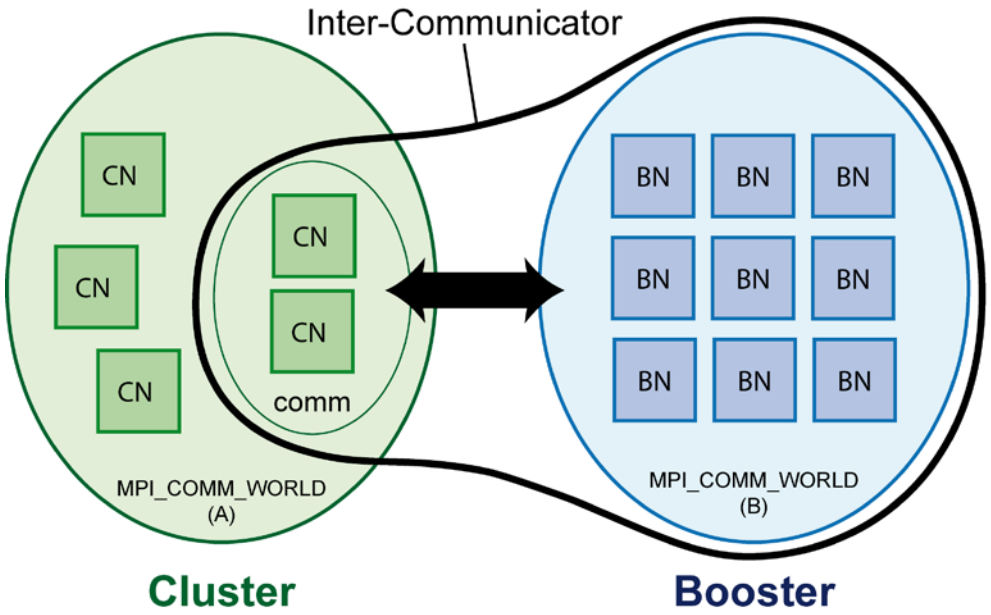
• **Eicker et al.**, *Bridging the DEEP Gap - Implementation of an Efficient Forwarding Protocol*, Intel European Exascale Labs - Report 2013 34-41, (2014)

# PROGRAMMING ENVIRONMENT

- One application can run:
  - Using only Cluster nodes
  - Using only Booster nodes
  - Distributed over Cluster and Booster
    - *In this case two executables are created*
    - *Collective offload process*

- ParaStation Global MPI
  - Enables distributing code
  - Uses **`MPI_Comm_spawn()`**
    - *Collective spawn groups of processes from Cluster to Booster (or vice-versa)*
  - Inter-communicator
    - *Connects the 2 MPI_Comm_worlds*

- One can also start two parts of a code and connect them via **`MPI_Connect()`**
- Or have one single common **`MPI_Comm_World()`** and split it into subcommunicators via **`MPI_Comm_Split()`**





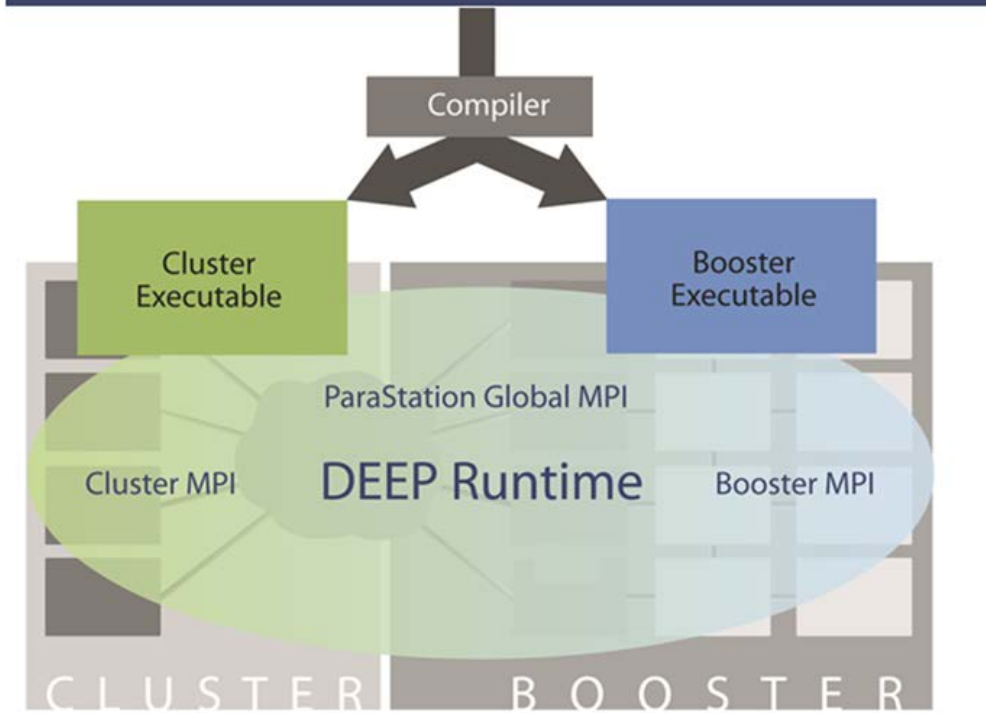
- **Clauss et al.**, *Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing*, COSH@HiPEAC, (2016)

# COMPILE AND RUN

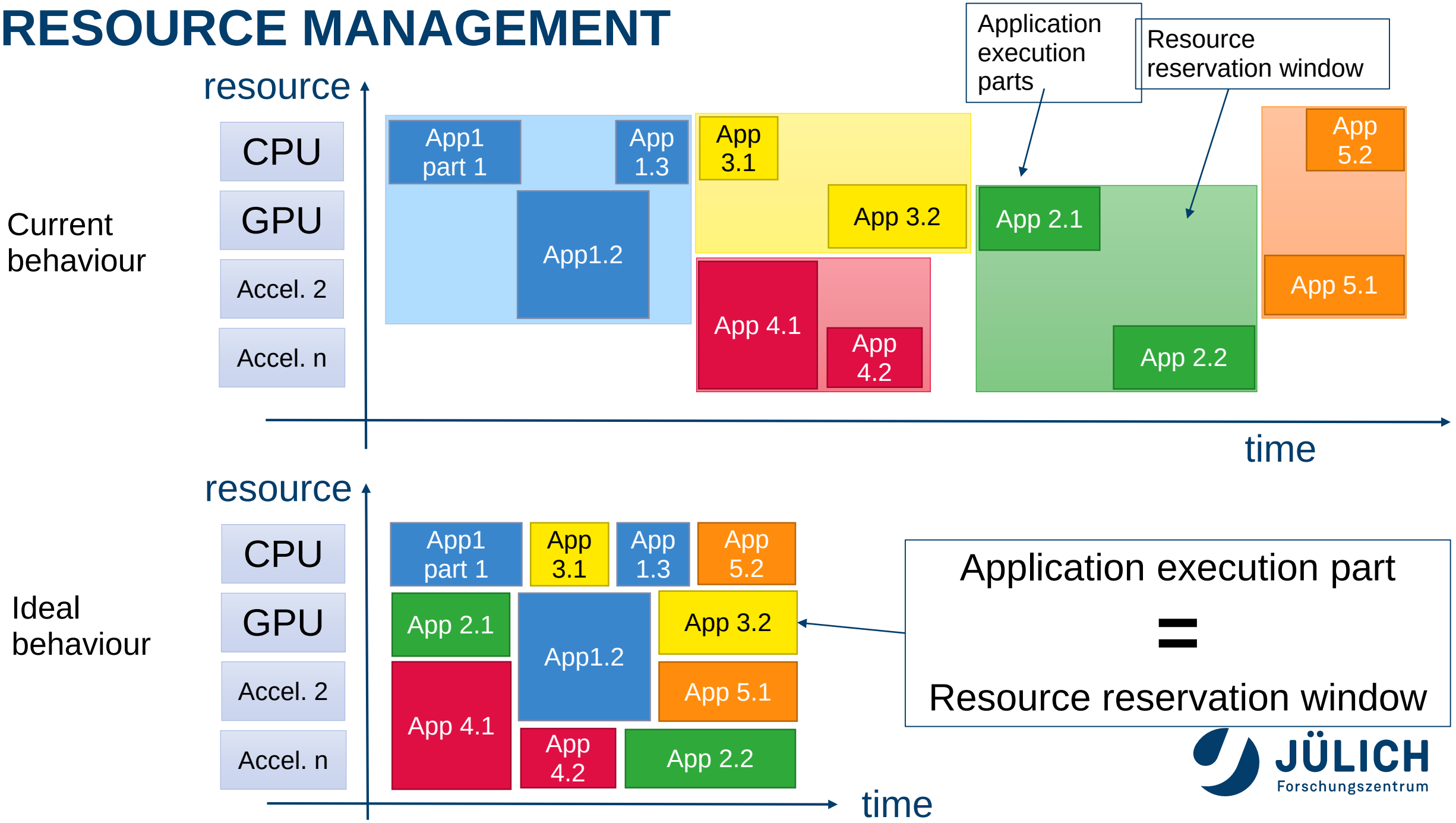
- **Compilation**
  - Creates two executables
    - One for `__CLUSTER__` code
    - One for `__BOOSTER__` code
- **Batch system**
  - Reserves required resources
- **Execution**
  - Script starts Booster code
  - This code calls `MPI_Comm_spawn()` with name of Cluster executable
- **Runtime + Scheduler + FS**
  - Detect ParaStation MPI calls
  - Distribute child binaries

```
salloc --partition=cluster -N 4
       : --partition=booster -N 12
srun --pack-group=0 -N 4 -n 8
       ./hi_booster
```

```
int main (int argc, char *argv[]){
   /* ... */
   MPI_Comm_spawn("./xPic.Cluster", &argv[1],
      nproc, MPI_INFO_NULL, 0, GRID_COMM_WORLD,
      INTERCOMM, MPI_ERRCODES_IGNORE);
   /* ... */
}
```

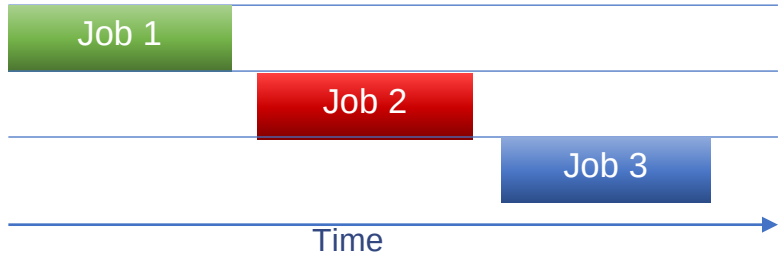


# RESOURCE MANAGEMENT



# IMPROVED WORKFLOW SUPPORT

- Simple workflows realizable using dependent jobs
  - Costly data buffering to secondary memory
- **Goal**: Overlapping job execution
  - Currently not supported by Slurm
    - Whole job pack either accepted or rejected
    - All jobs allocated and run in parallel
    - All jobs wait for allocation if any of the jobs can not be allocated at the moment
- New parameter `--delay` introduced in sbatch command for job packs
  - Amount of time, the next job should wait after start of the first job in a job pack

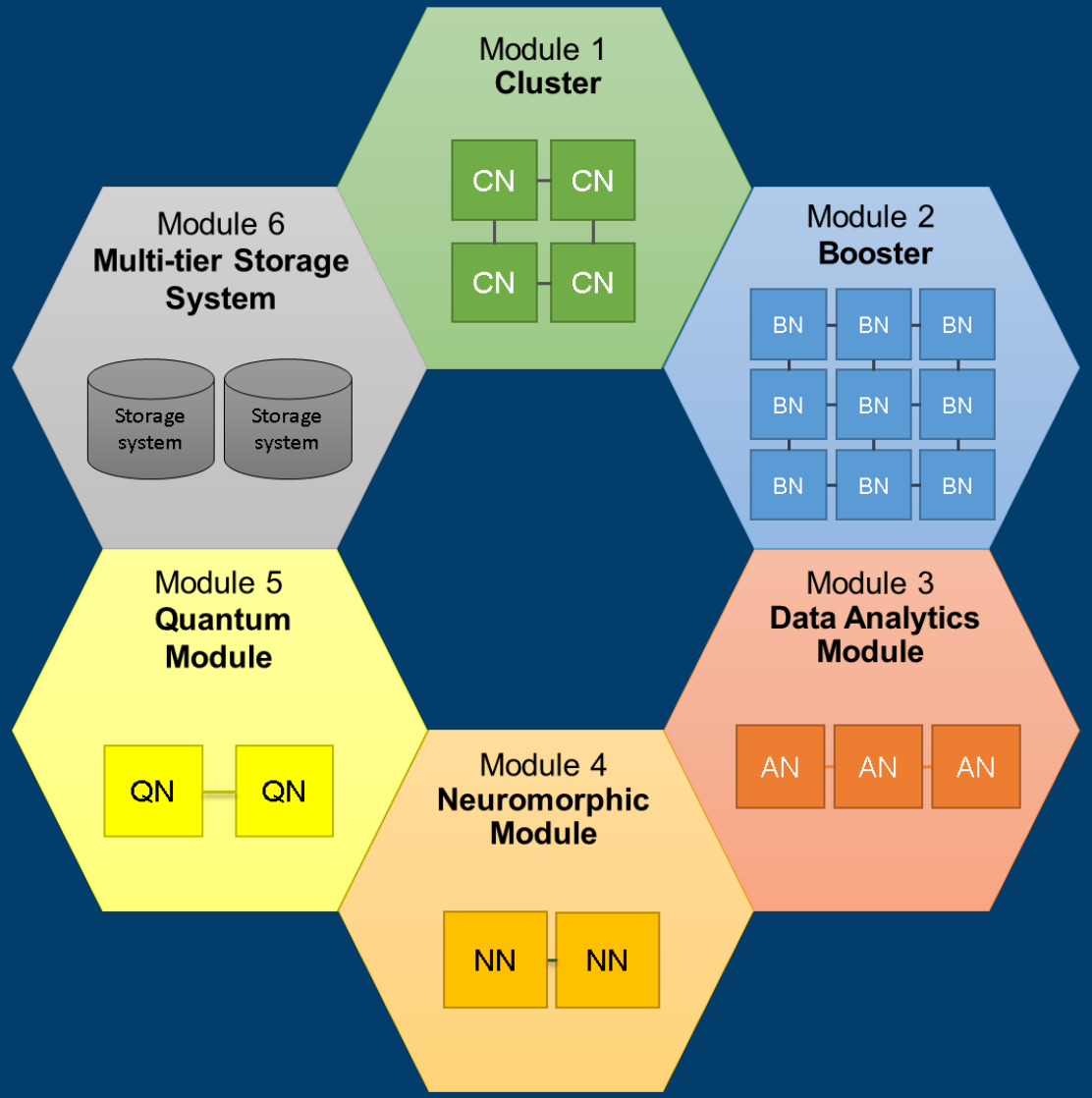



Typical Workflow supported by Slurm

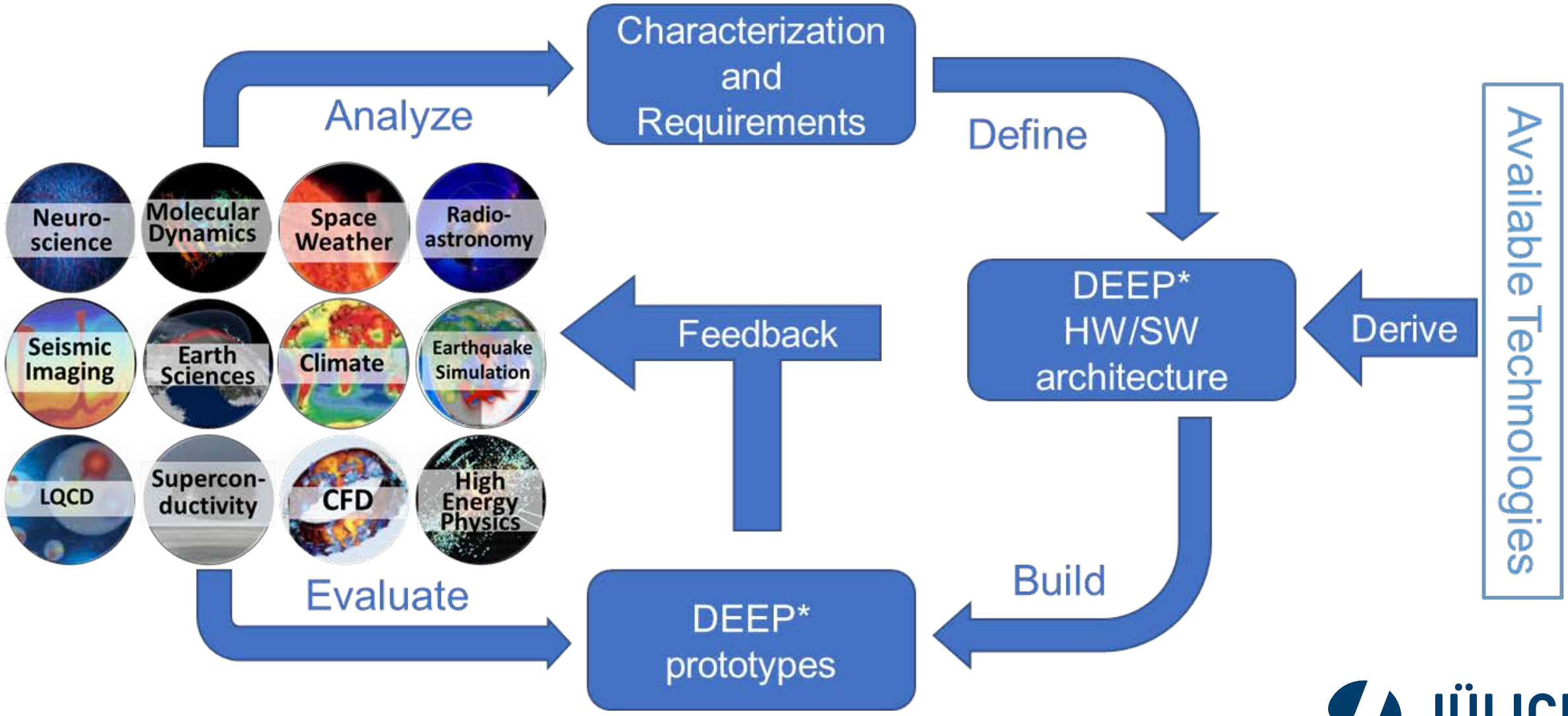

Workflow we are trying to achieve

# OUTLINE

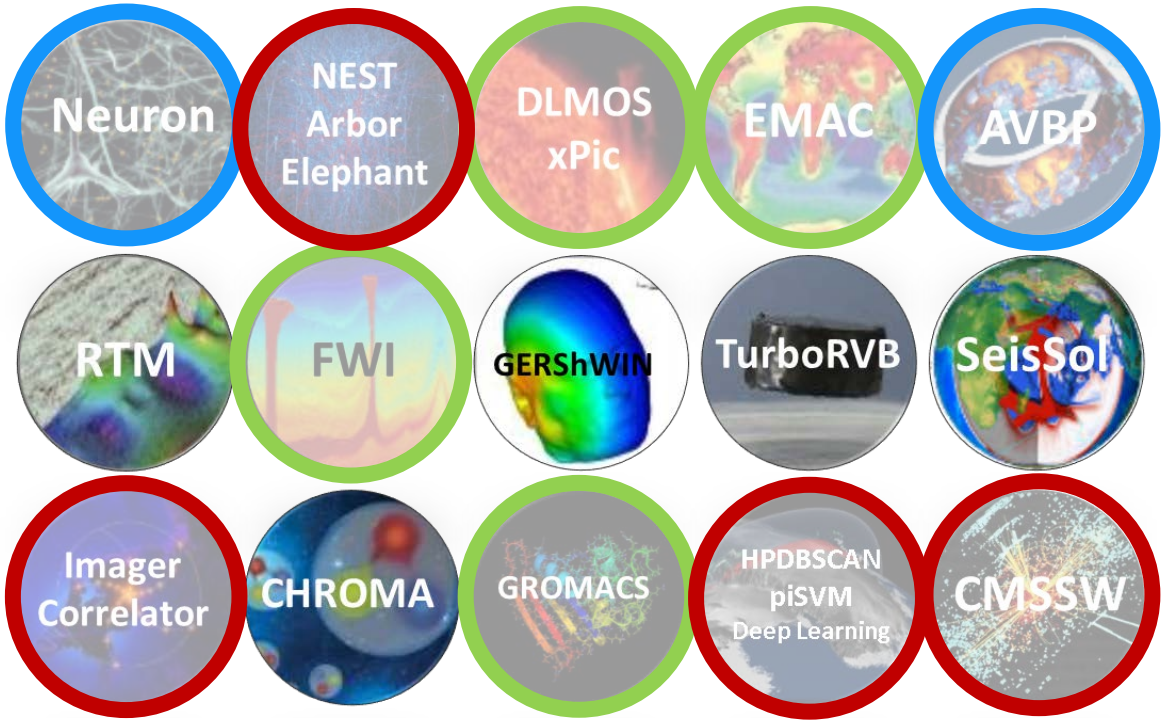
- **Evolution of HPC architectures**
  - Global historical evolution
  - Dual architecture at JSC
  - Cluster-Booster
  - Modular Supercomputing Architecture
- **Software**
  - Software stack
  - Network bridging
  - Programming environment
  - Scheduling and resource management
- **Application experience**
- **Conclusions and Next steps**



# Application-driven HW+SW developments



# Architecture Use-Modes




Cluster-Booster use mode

**Code partition**
**Workflow**
**I/O forward**

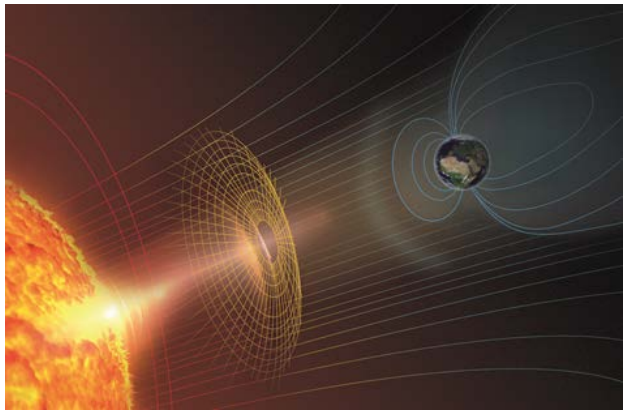

- **Kreuzer, et al.,** *Application Performance on a Cluster-Booster System.* IPDPSW – HCW (2018) [10.1109/IPDPSW.2018.00019]
- **Kreuzer et al.** *The DEEP-ER project: I/O and resiliency extensions for the Cluster-Booster architecture.* HPCC'18 proceedings (*2018)* [10.1109/HPCC/SmartCity/DSS.2018.00046]
- Wolf et al., *PIC algorithms on DEEP: The iPiC3D case study*. PARS-Mitteilungen 32, 38-48 (2015)
- Christou et al., *EMAC on DEEP*, Geoscientific model devel.(2016) [10.5194/gmd-9-3483-2016]
- Kumbhar et al., *Leveraging a Cluster-Booster Architecture for Brain-Scale Simulations*, Lecture Notes in Computer Science 9697 (2016) [10.1007/978-3-319-41321-1_19]
- Leger et al., *Adapting a Finite-Element Type Solver for Bioelectromagnetics to the DEEP-ER Platform*. ParCo 2015, Advances in Parallel Computing, 27 (2016) [10.3233/978-1-61499-621-7-349]

# Application use case: xPic

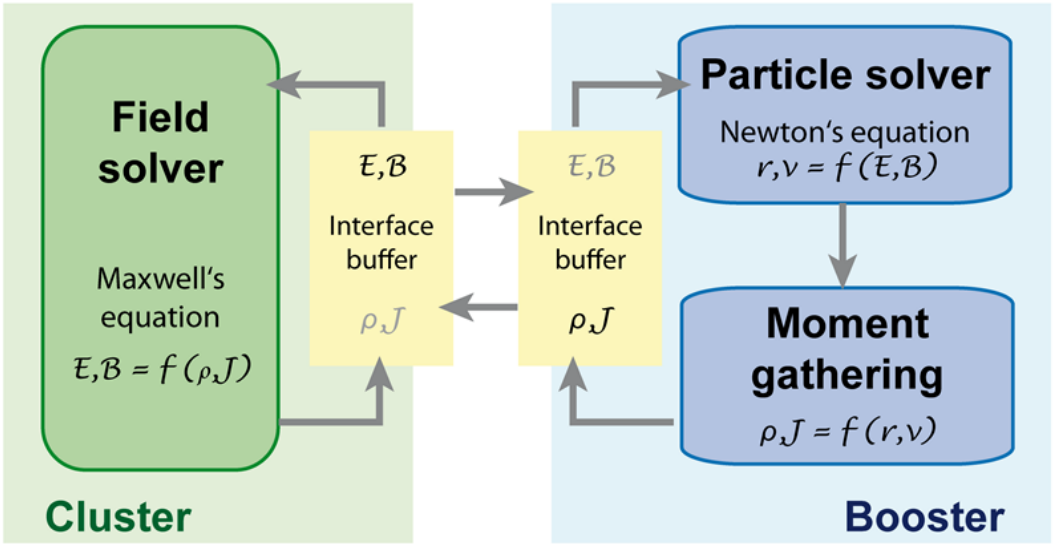
- **Space Weather simulation**
  - Simulates plasma produced in solar eruptions and its interaction with the Earth magnetosphere
  - Particle-in-Cell (PIC) code
  - Authors: KU Leuven

KATHOLIEKE UNIVERSITEIT LEUVEN

xPIC



- **Two solvers:**
  - **Field solver**: Computes electromagnetic (EM) field evolution
    - Limited code scalability
    - Frequent, global communication
  - **Particle solver**: Calculates motion of charged particles in EM-fields
    - Highly parallel
    - Billions of particles
    - Long-range communication





A. Kreuzer, J. Amaya, N. Eicker, **E. Suarez***, *"Application performance on a Cluster-Booster system"*, 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), HCW (20th International Heterogeneity in Computing Workshop), Vancouver (2018), p: 69 - 78. [doi: 10.1109/IPDPSW.2018.00019]


JÜLICH Forschungszentrum

# xPic – ORIGINAL CONFIGURATION

```
1
2   for (auto i=beg+1; i<=end; i++){
3     fld.solver->calculateE();
4     fld.cpyToArr_F();
5
6
7
8     pcl.cpyFromArr_F();
9     for (auto is=0; is<nspec; is++) {
10      pcl.species[is].ParticlesMove();
11      pcl.species[is].ParticleMoments();
12    }
13    pcl.cpyToArr_M();
14
15
16
17    fld.solver->calculateB();
18    fld.cpyFromArr_M();
19  }
20
```

`fld`: Field Solver

Copy information between solvers

`plc`: Particle Solver

# xPic – CODE PARTITION

```
#ifdef __CLUSTER__
for (auto i=beg+1; i<=end; i++){
  fld.solver->calculateE();
  fld.cpyToArr_F();
  ClusterToBooster();
  // Auxiliary computations
  ClusterWait();









BoosterToCluster();

BoosterWait();
  fld.solver->calculateB();
  fld.cpyFromArr_M();
}
#endif
```

```
#ifdef __BOOSTER__
for (auto i=beg+1; i<=end; i++){



  ClusterToBooster();

  ClusterWait();
  pcl.cpyFromArr_F();
  for (auto is=0; is<nspec; is++) {
    pcl.species[is].ParticlesMove();
    pcl.species[is].ParticleMoments();
  }
  pcl.cpyToArr_M();
  BoosterToCluster();
  // I/O and auxiliary computations
  BoosterWait();



}
#endif
```

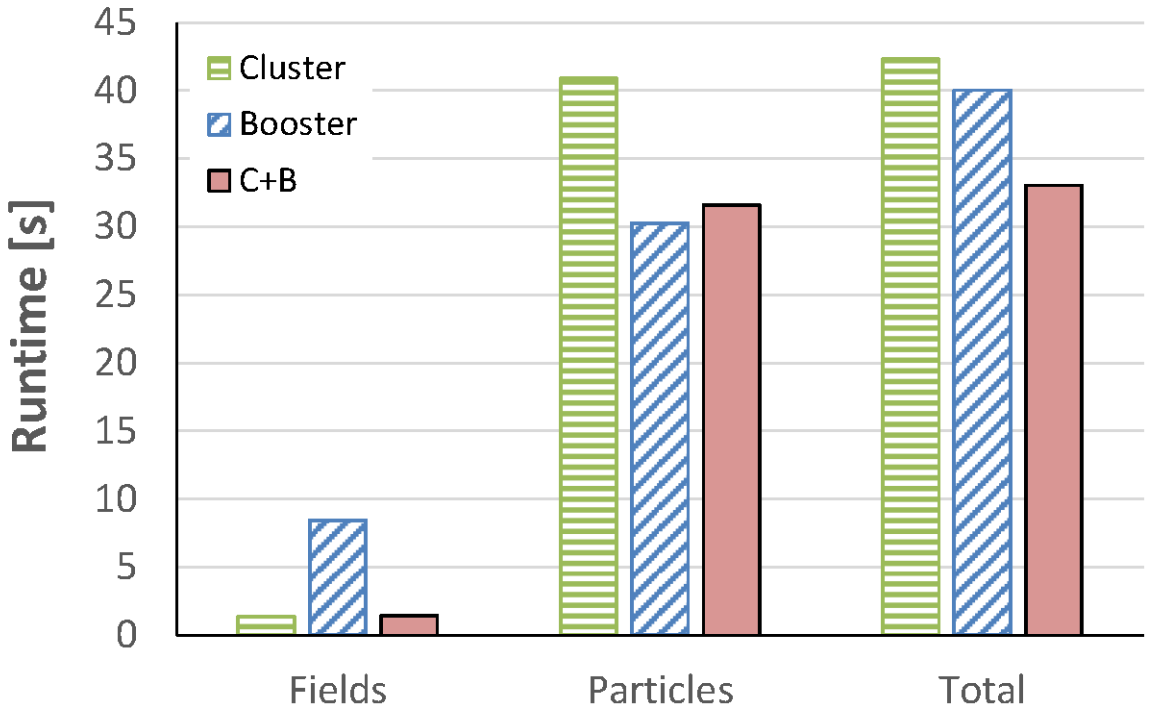
# xPic – (1-NODE) PERFORMANCE RESULTS

- **Field solver**: 6× faster on Cluster
- **Particle solver**: 1.35 × faster on Booster
- **Overall performance gain:**

1× node
**28% × gain** compared to Cluster alone
**21% × gain** compared to Booster alone

8× nodes
**38% × gain** compared to Cluster only
**34% × gain** compared to Booster only

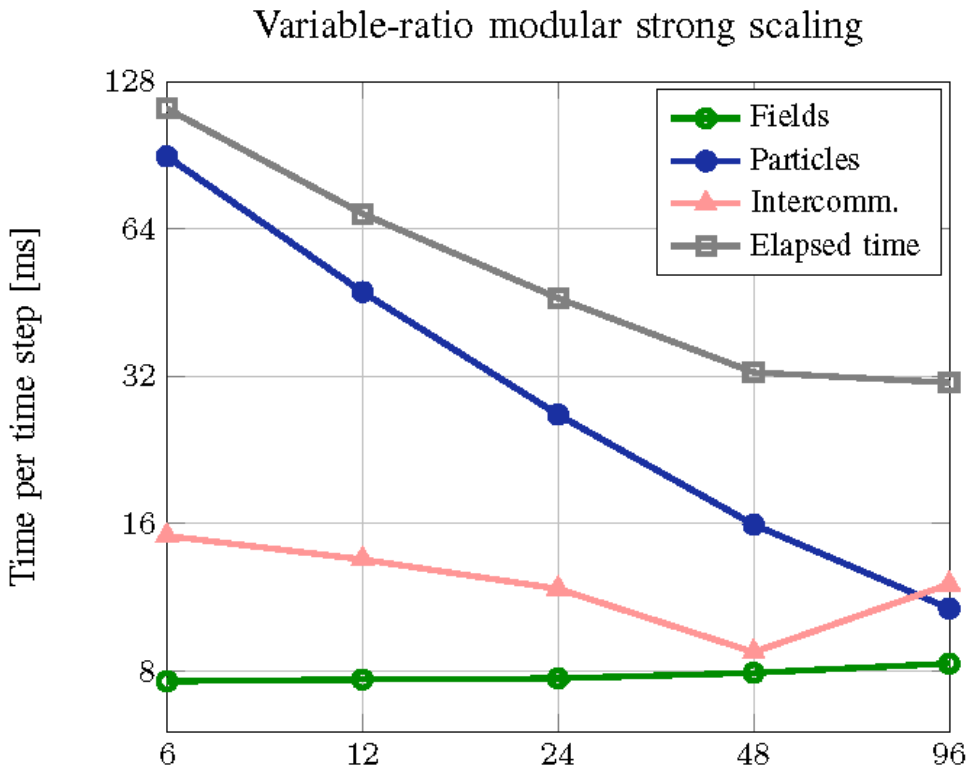
  - 3%-4% overhead per solver for C+B communication (point to point)




| | |
|---|---|
| #cells per node | 4096 |
| #particles per cell | 2048 |
| Compilation flags | -openmp, -mavx (Cluster)<br>-xMIC-AVX512 (Booster) |


**A. Kreuzer et al**. "*Application Performance on a Cluster-Booster System*", 2018 IEEE IPDPS Workshops (IPDPSW), Vancouver, Canada, p 69 - 78 (2018) [10.1109/IPDPSW.2018.00019]

# xPic – STRONG SCALING on JURECA



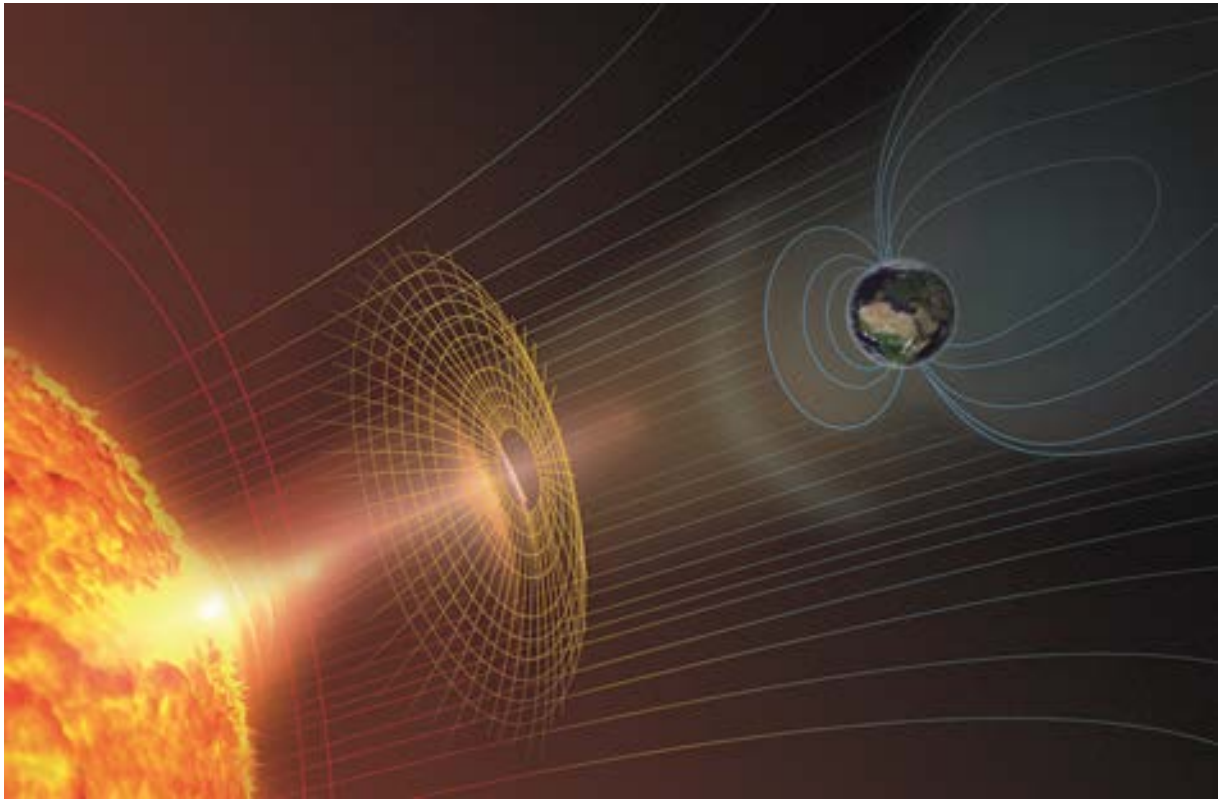

(4 Cluster nodes)

| | |
|---|---|
| #cells per node | 36864 |
| #particles per cell | 1024 |
| #blocks per MPI process | 12, 32 or 64 |
| Compilation flags | -mavx (Cluster)<br>-openmp, xMIC-AVX512 (Booster) |

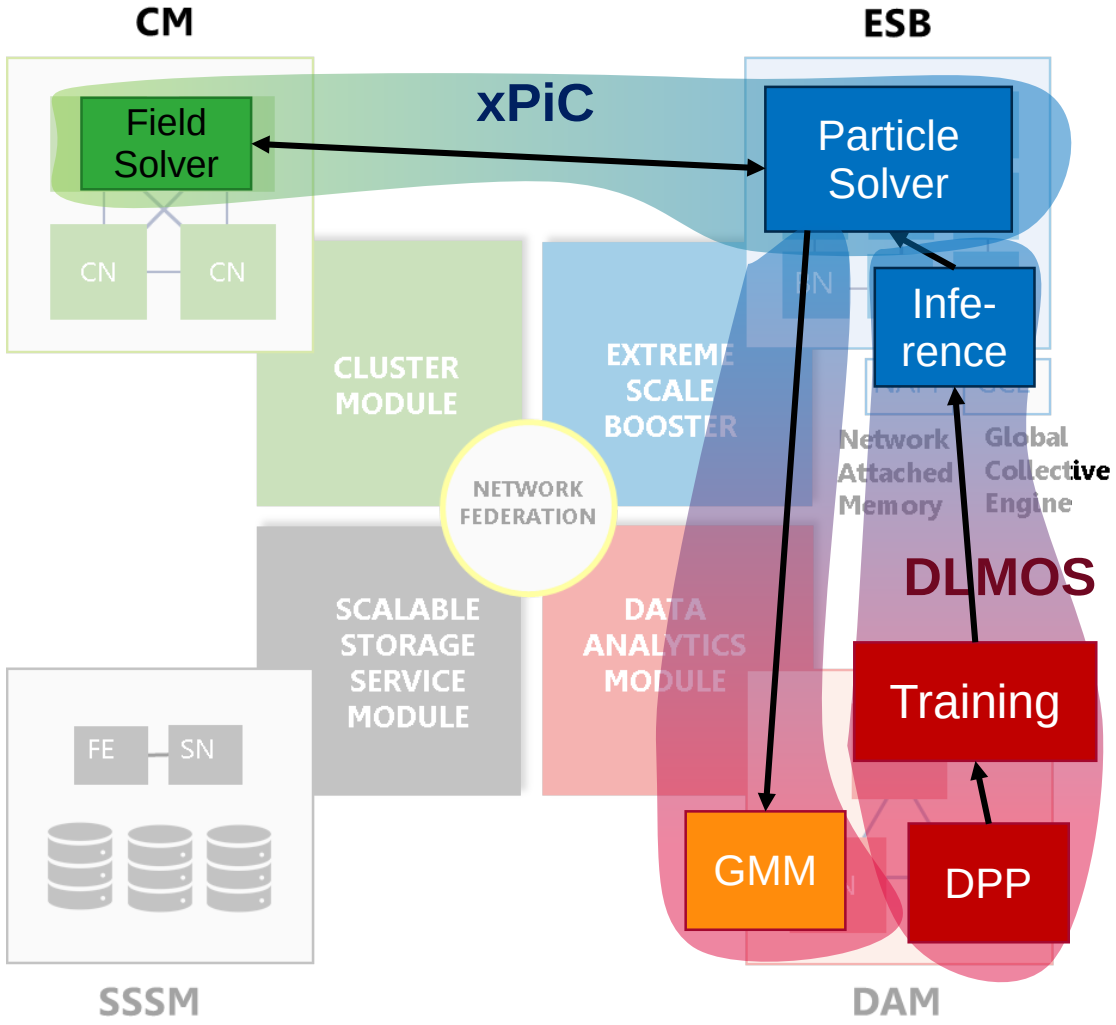
- Code portions can be scaled-up independently
  - Particles scale almost linearly on Booster
  - Fields kept constant on the Cluster (4CNs)
- A configuration is reached where same time is spent on Cluster and Booster
  - Additional 2× time-saving is enabled via overlapping

J. De Amicis*, **E.Suarez***, J. Amaya, N. Eicker, G.Lapenta, Th.Lippert, *"Assessing the scalability of the xPic code on a large-scale modular supercomputer",* In preparation

# xPic worfklow – mapping on DEEP-EST

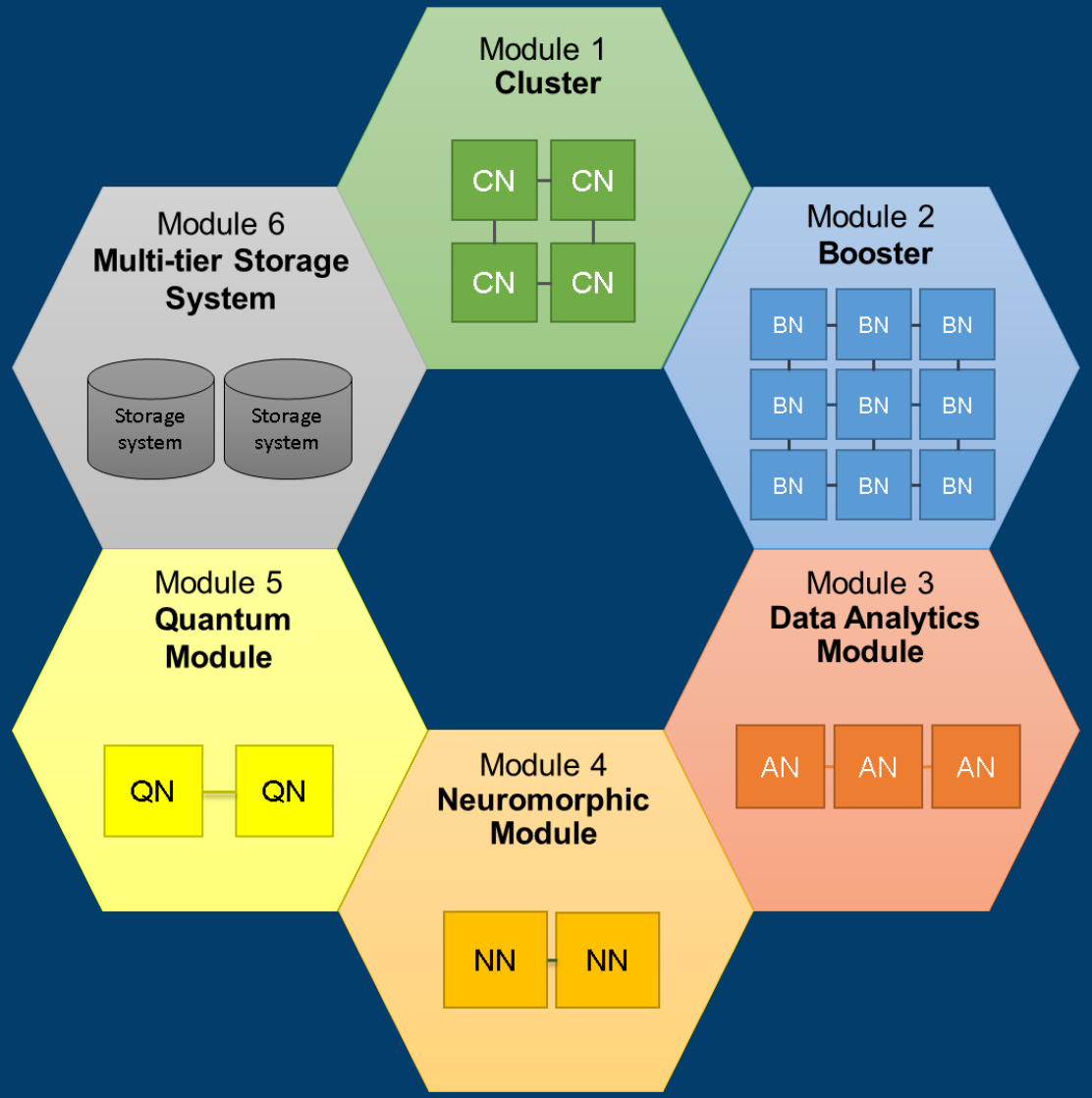


**DLMOS** ⟶ **xPiC** ↔ **GMM**



# OUTLINE

- **Evolution of HPC architectures**
  - Global historical evolution
  - Dual architecture at JSC
  - Cluster-Booster
  - Modular Supercomputing Architecture
- **Software**
  - Software stack
  - Network bridging
  - Programming environment
  - Scheduling and resource management
- **Application experience**
- **Conclusions and Next steps**



# CONCLUSIONS

- **The Modular Supercomputing Architecture (MSA)**
  - Orchestrates heterogeneity at system level
  - Allows scaling hardware in economical way (Booster → Exascale)
  - Serves very diverse application profiles
    - Maximum flexibility for users, without taking anything away (still can use individual modules)
- **Distribute applications on the MSA give each code-part a suitable hardware**
  - Straight-forward implementation for workflows
  - Partition at MPI-level interesting for multi-physics / multi-scale codes
  - Monolithic codes do not need to be divided
- **Current / Upcoming implementations of MSA**
  - DEEP prototypes, JURECA, JUWELS (in 2020)
  - MELUXINA (Luxembourgh EuroHPC Petascale system)
  - Tianhe-3 (heterogeneous flexible architecture)
    - https://www.r-ccs.riken.jp/R-CCS-Symposium/2019/slides/Wang.pdf

# NEXT STEPS

- **Hardware deployments**
  - DEEP-EST Booster (January 2020)
  - JUWELS Booster (Mid 2020)
    - Integrating later JUNIQ (Quantum Annealer)
  - And if everything goes well, then…. Exascale!
- **Software development**
  - Develop tools to map applications to hardware
  - Improve scheduling of heterogeneous jobs/workflows
  - Facilitate exploitation of new memory technologies
  - Modularize more codes

# THANK YOU!

www.deep-projects.eu

@DEEPprojects

The DEEP projects have received funding from the European Union's Seventh Framework Programme (FP7) for research, technological development and demonstration and the Horion2020 (H2020) funding framework under grant agreement no. FP7-ICT-287530 (DEEP), FP7-ICT-610476 (DEEP-ER) and H2020-FETHPC-754304 (DEEP-EST).