Georgia Tech School of Computer Science

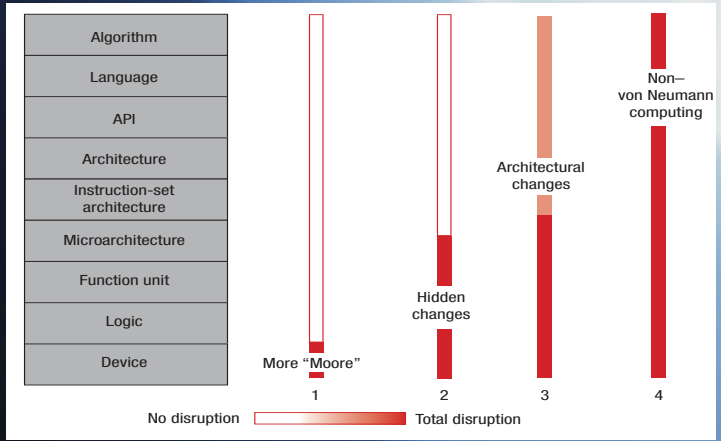
MCHPC'19 Panel: Software and Hardware Support for Programming Heterogeneous Memory

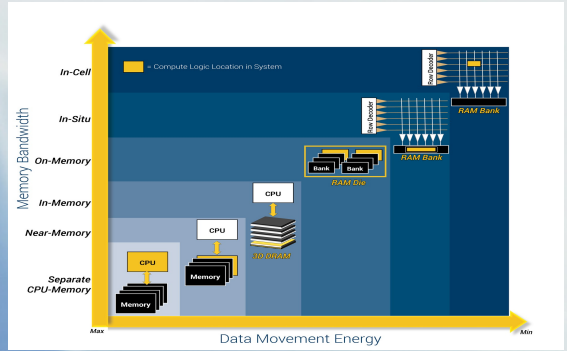
Vivek Sarkar

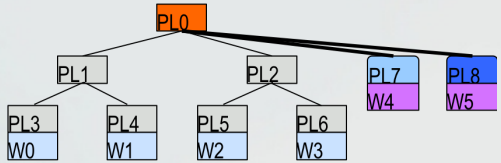
Professor, School of Computer Science

Stephen Fleming Chair for Telecommunications, College of Computing

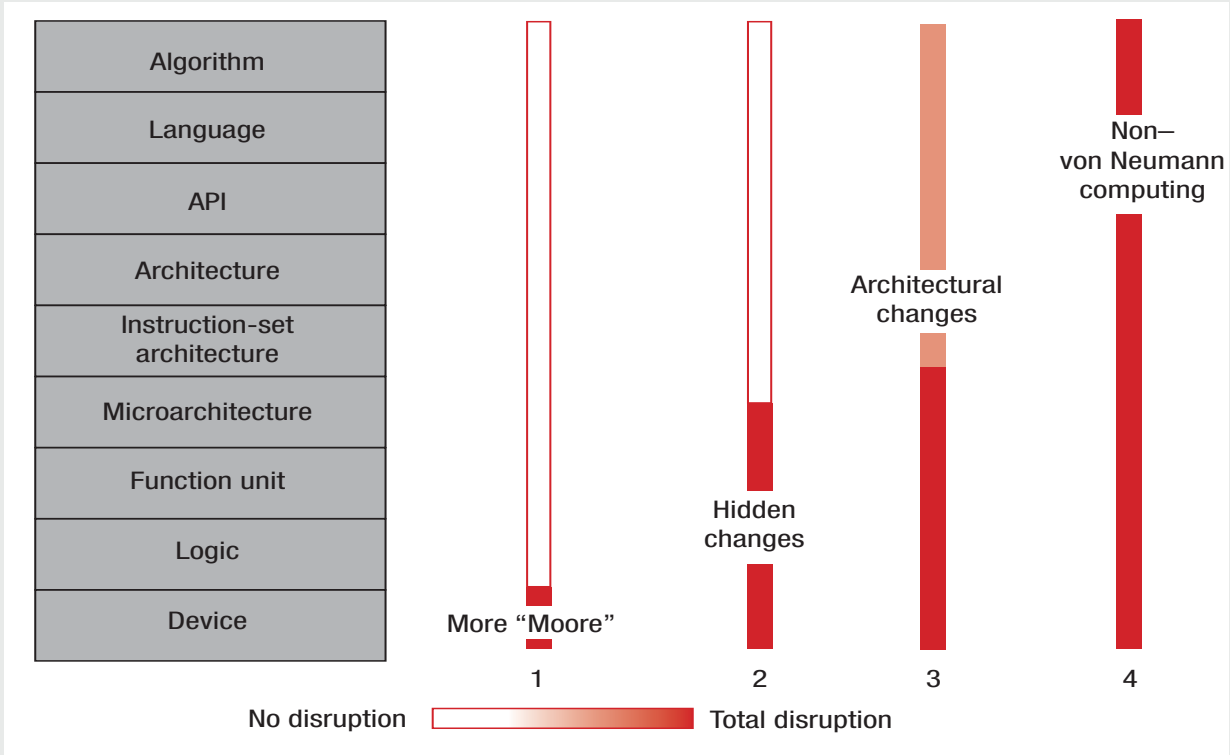
Georgia Institute of Technology







Levels of Disruption in Moore's Law End-Game and Post-Moore eras



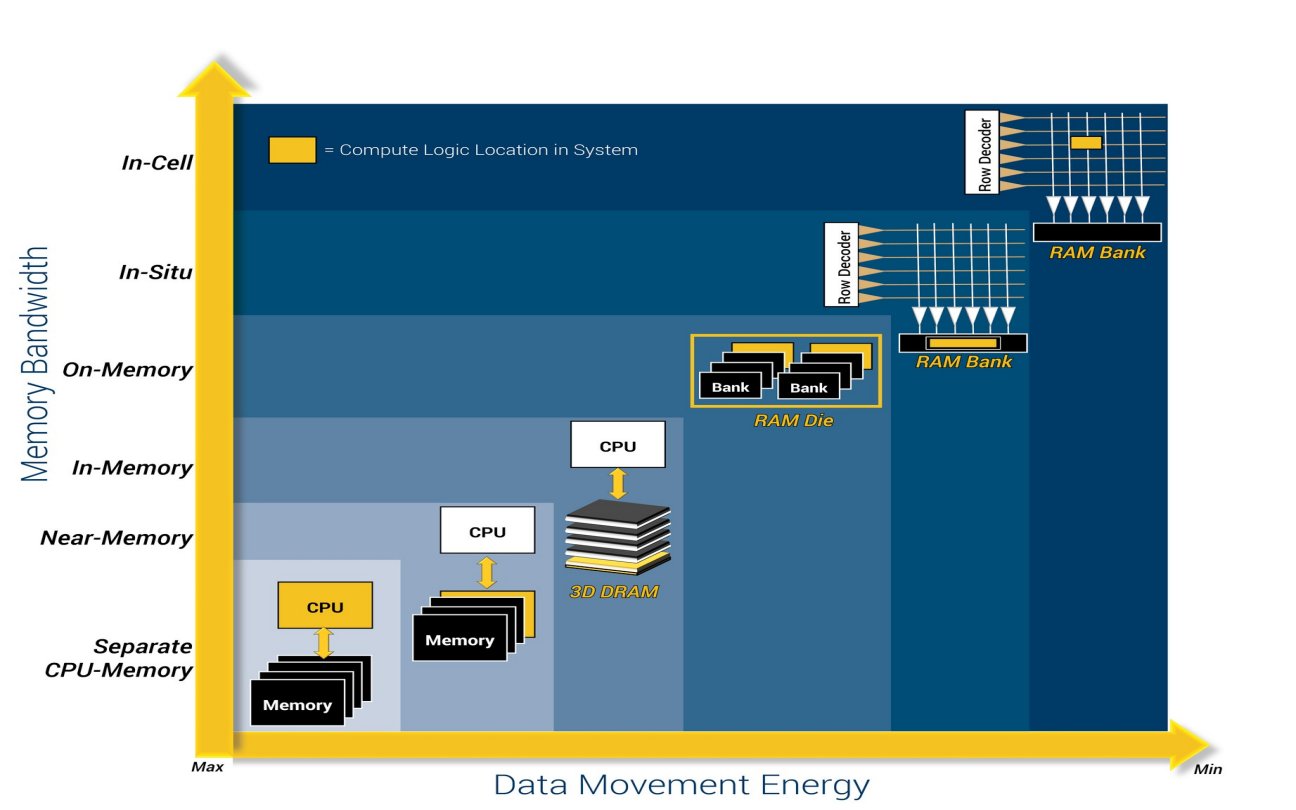
At the far right (level 4) are non–von Neumann architectures, which completely disrupt all stack levels, from device to algorithm.

At the least disruptive end (level 1) are more "Moore" approaches, such as new transistor technology and 3D circuits, which affect only the device and logic levels.

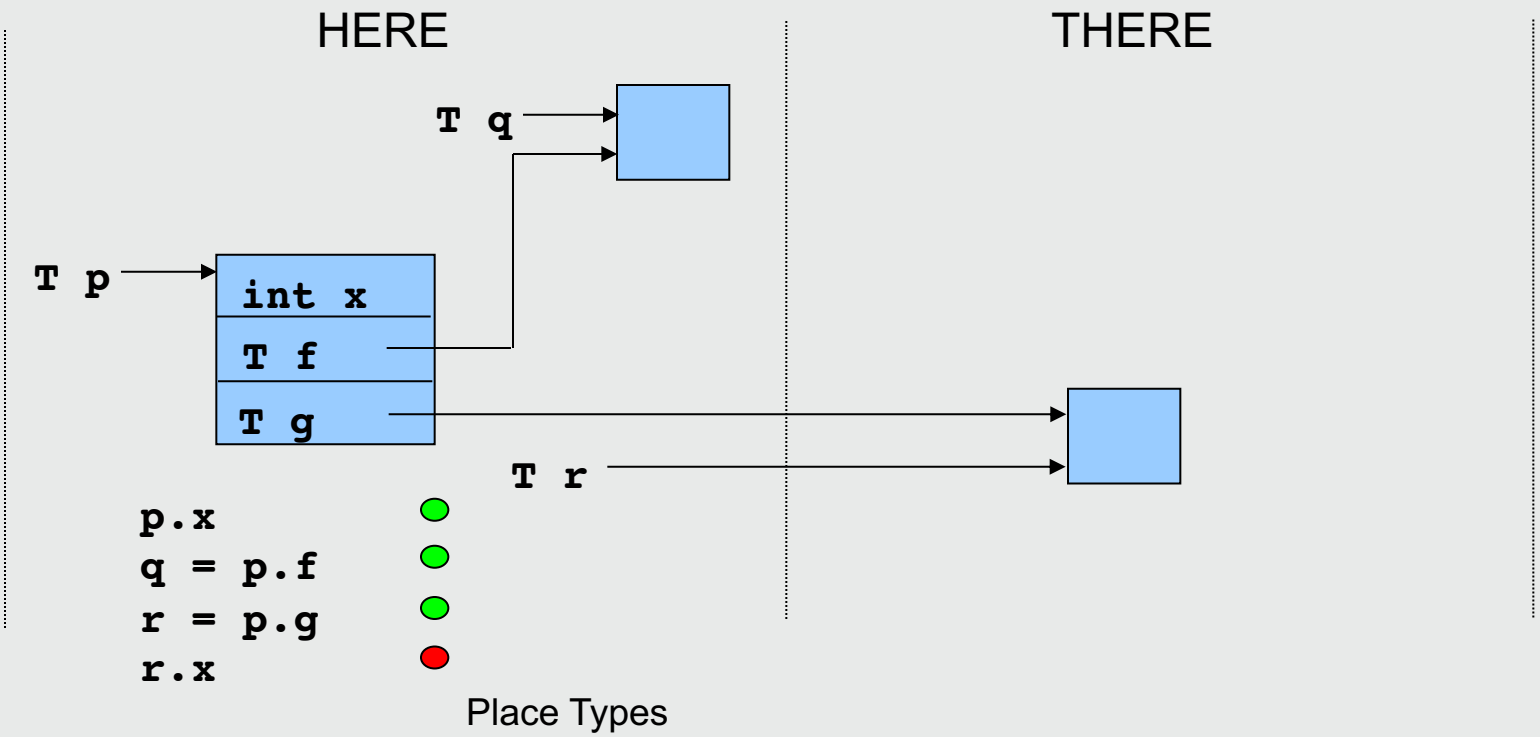
All future hardware directions are characterized by ***extreme heterogeneity***.

Source: "Rebooting Computing: The Road Ahead", T.M.Conte, E.P.DeBenedictis, P.A.Gargini, E.Track, IEEE Computer, 2017.

Range of Approaches (and disruptions!) for Memory-Centric Processing

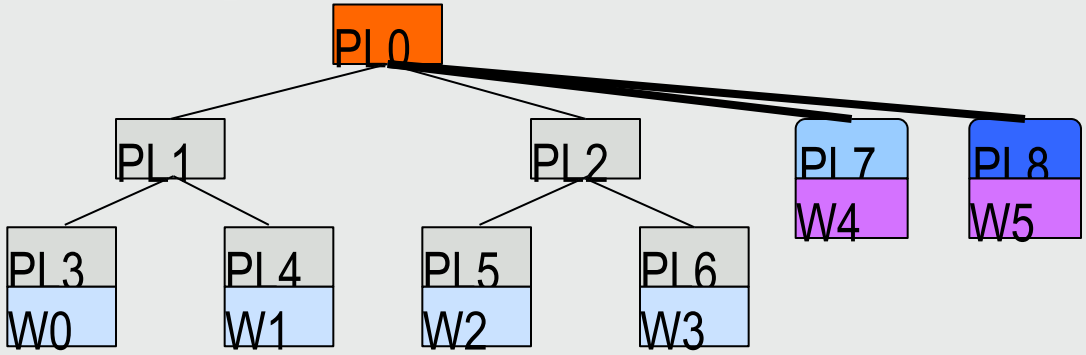


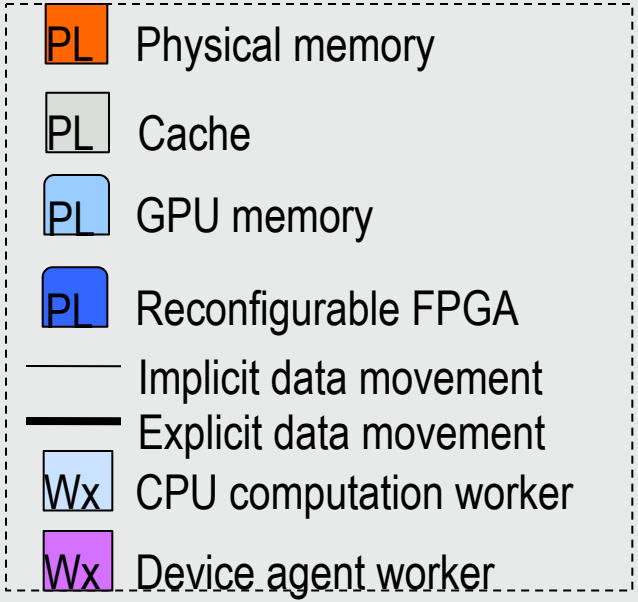
Memory-Centric Programming with Places in X10



Type inference for locality analysis of distributed data structures.
Satish Chandra, Vijay Saraswat, Vivek Sarkar, Ras Bodik. PPoPP 2008.

Hierarchical Place Tree (HPT) extension for memory hierarchies and heterogeneous architectures





- **Devices (GPU or FPGA) are represented as memory module places and agent workers**
  - **GPU memory configuration is fixed, while FPGA memory can be reconfigured at runtime**
- ***async at(P) S***
  - **Creates new task to execute statement S at place P**

Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Date Movement. Yonghong Yan, Jisheng Zhao, Yi Guo, Vivek Sarkar. LCPC 2009.

"Mapping a Data-Flow Programming Model onto Heterogeneous Platforms." Alina Sbirlea, Yi Zou, Zoran Budimlic, Jason Cong, Vivek Sarkar. LCTES 2012

Data Layout Transformations (Example)

```
// Input
  double C[NI][NJ];
  double A[NI][NK];
  double B[NK][NJ];
  …
  for (k = 0; k < nk; k++)
    for (i = 0; i < ni; i++)
      for (j = 0; j < nj; j++)
        C[i][j] += alpha * A[i][k]
                        * B[k][j];
```

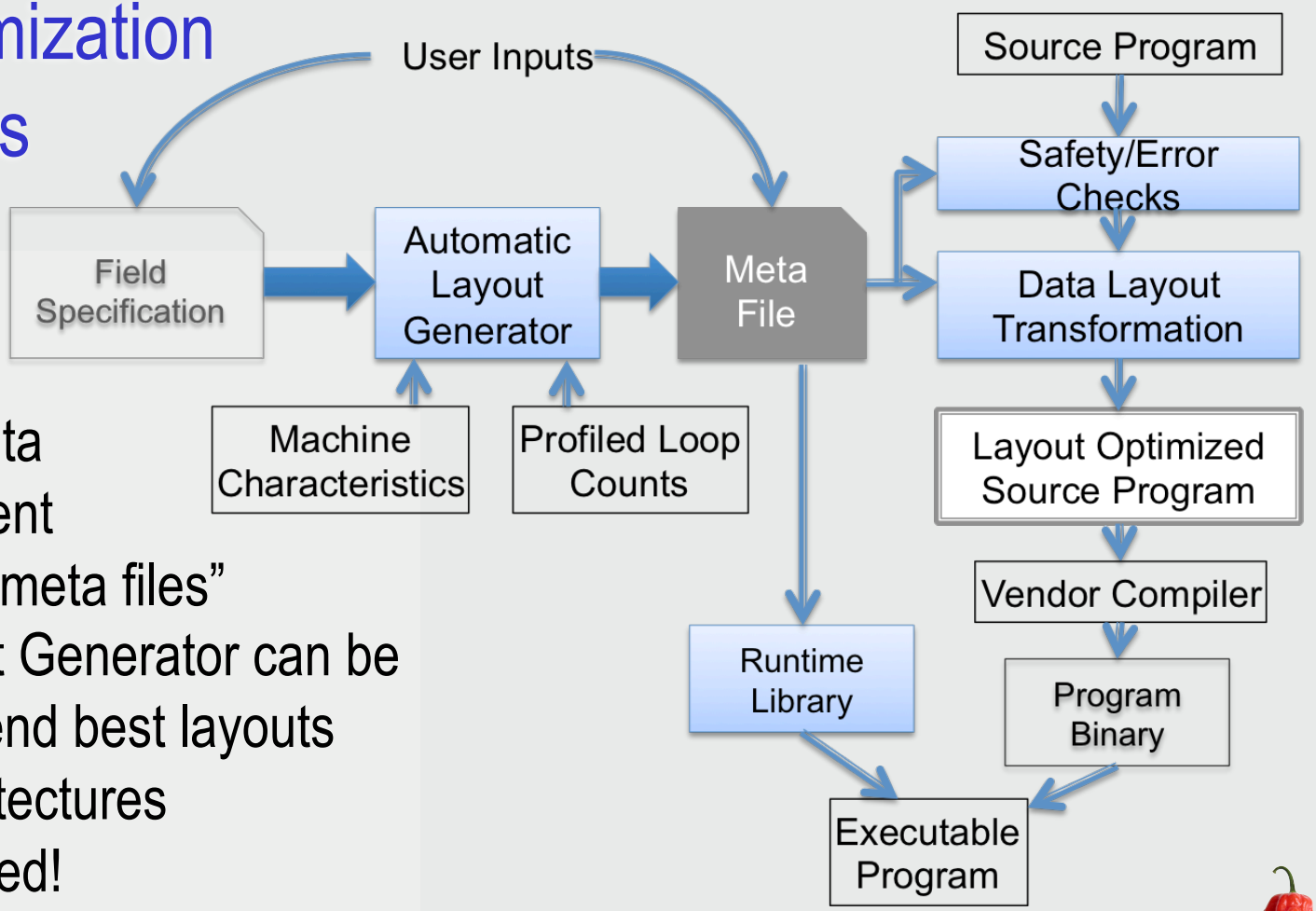
```
// Array dimensional permutation
  double C[NI][NJ];
  double A[NK][NI];
  double B[NK][NJ];
  …
  for (k = 0; k < nk; k++)
    for (i = 0; i < ni; i++)
      for (j = 0; j < nj; j++)
        C[i][j] += alpha * A[k][i]
                        * B[k][j];
```

```
// Conversion to Struct-of-Array
  double C[NI][NJ];
  struct Struct_of_AB {
    double A[NI];
    double B[NJ];
  };
  Struct_of_AB SoAB[NK];
  …
  for (k = 0; k < nk; k++)
    for (i = 0; i < ni; i++)
      for (j = 0; j < nj; j++)
        C[i][j] += alpha * SoAB[k].A[i]
                        * SoAB[k].B[j];
```

Compiler Optimization of Data Layouts

Separation of Concerns:

- User specifies data layouts for different architectures in “meta files”
- Automatic Layout Generator can be used to recommend best layouts for different architectures
- Code is unchanged!



“Data Layout Optimization for Portable Performance.” K. Sharma, I. Karlin, J. Keasler, J. McGraw, V.Sarkar, EuroPar ‘15.

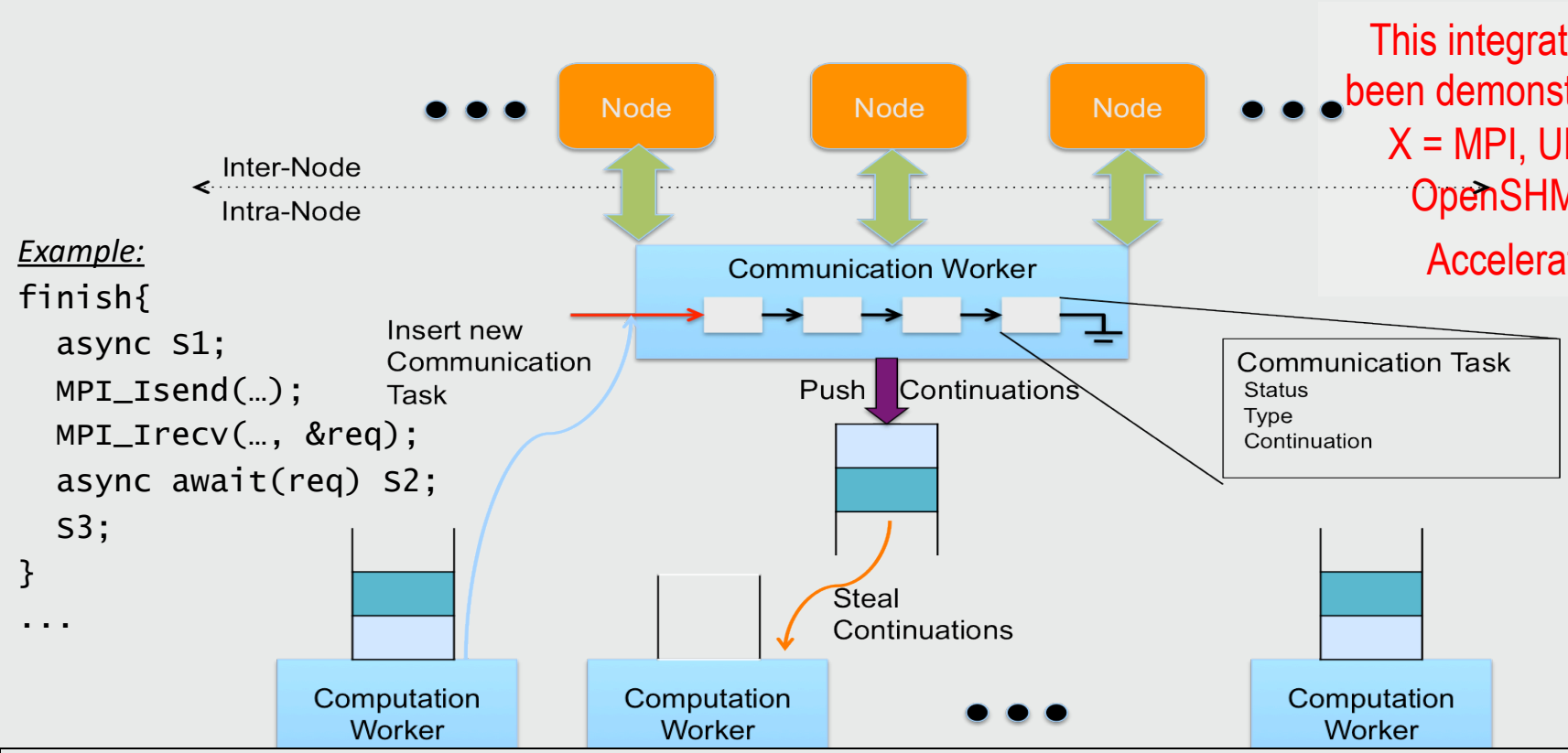
Automatic Data Layout Problem for CPU+GPU architectures

- Previous framework limited to a single data-layout for the entire program

- Large programs with multiple kernels might require different layouts for different parts of the program
  - The best data layout could be a single layout for the entire program or multiple layouts for each kernel with data-remapping in between

"Automatic Data Layout Generation and Kernel Mapping for CPU+GPU Architectures."
Deepak Majeti, Kuldeep Meel, Raj Barik and Vivek Sarkar. CC 2016.

Integrating Data Movement with Tasking (X + Habanero)



Example:

```
finish{
  async S1;
  MPI_Isend(…);
  MPI_Irecv(…, &req);
  async await(req) S2;
  S3;
}
...
```

This integration has been demonstrated for X = MPI, UPC++, OpenSHMEM, Accelerators

"Integrating Asynchronous Task Parallelism with MPI." Sanjay Chatterjee, Sagnak Tasirlar, Zoran Budimlic, Vincent Cave, Milind Chabbi, Max Grossman, Yonghong Yan, Vivek Sarkar. IPDPS 2013.

Compiler/Runtime Analysis can also help with debugging memory mappings, e.g., OpenMP's map clause

DRACC File 22

```
int init(){
  for(int i=0; i<C; i++){
    for(int j=0; j<C; j++) {
L18:  b[j+i*C]=1;    }
    a[i]=1;
    c[i]=0;
  }}
int Mult(){
#pragma omp target map(to:a[0:C]) map(tofrom:c[0:C])
                                  map(alloc:b[0:C*C])
#pragma omp teams distribute parallel for
L32:  for(int i=0; i<C; i++){
       for(int j=0; j<C; j++)
L34:     c[i]+=b[j+i*C]*a[j];
    }
   }
```

OMPSan: Reported Error

ERROR Definition of :***b*** on Line:18 is not reachable to Line:34,
Missing Clause:to:Line:32

OMPSan: Static Verification of OpenMP's Data Mapping Constructs.
Prithayan Barua, Shirako Jun, Tsang Whitney, Paudel Jeeva, Chen Wang IWOMP 2019

Towards Performance, Programmability, and Portability for future Memory Systems

- **Programming models**
  - Use of virtualized abstractions of heterogeneous memories, such as hierarchical places/locales
- **Compilers**
  - New optimizations that combine code transformations with data placement, layout and movement
- **Runtime Syste**ms
  - Integrated scheduling of asynchronous tasks and data movement
- **Debugging**
  - Co-design of prog model, compiler, runtime to simplify debugging