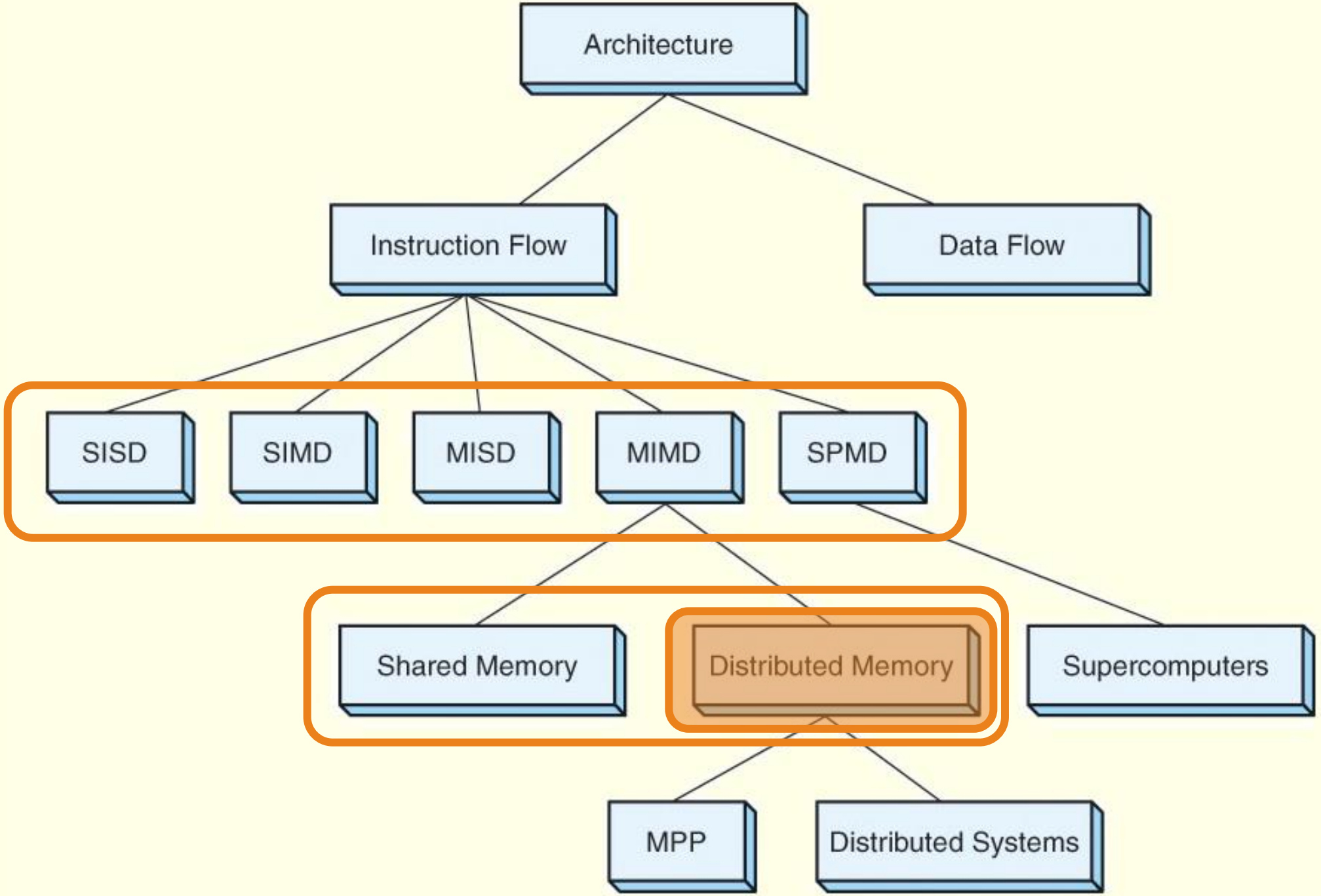
Parallel and Distributed Systems / 6

Pavel Krömer,
Dept. of Computer Science,
VSB – Technical University of Ostrava

Agenda

- Programming distributed memory systems
  - MPI and friends
- Literature
  - Peter Pacheco, An Introduction to Parallel Programming, Elsevier, 2011 (Ch. 3)
  - Alexander Supalov, Inside the message passing interface, DeGruyter

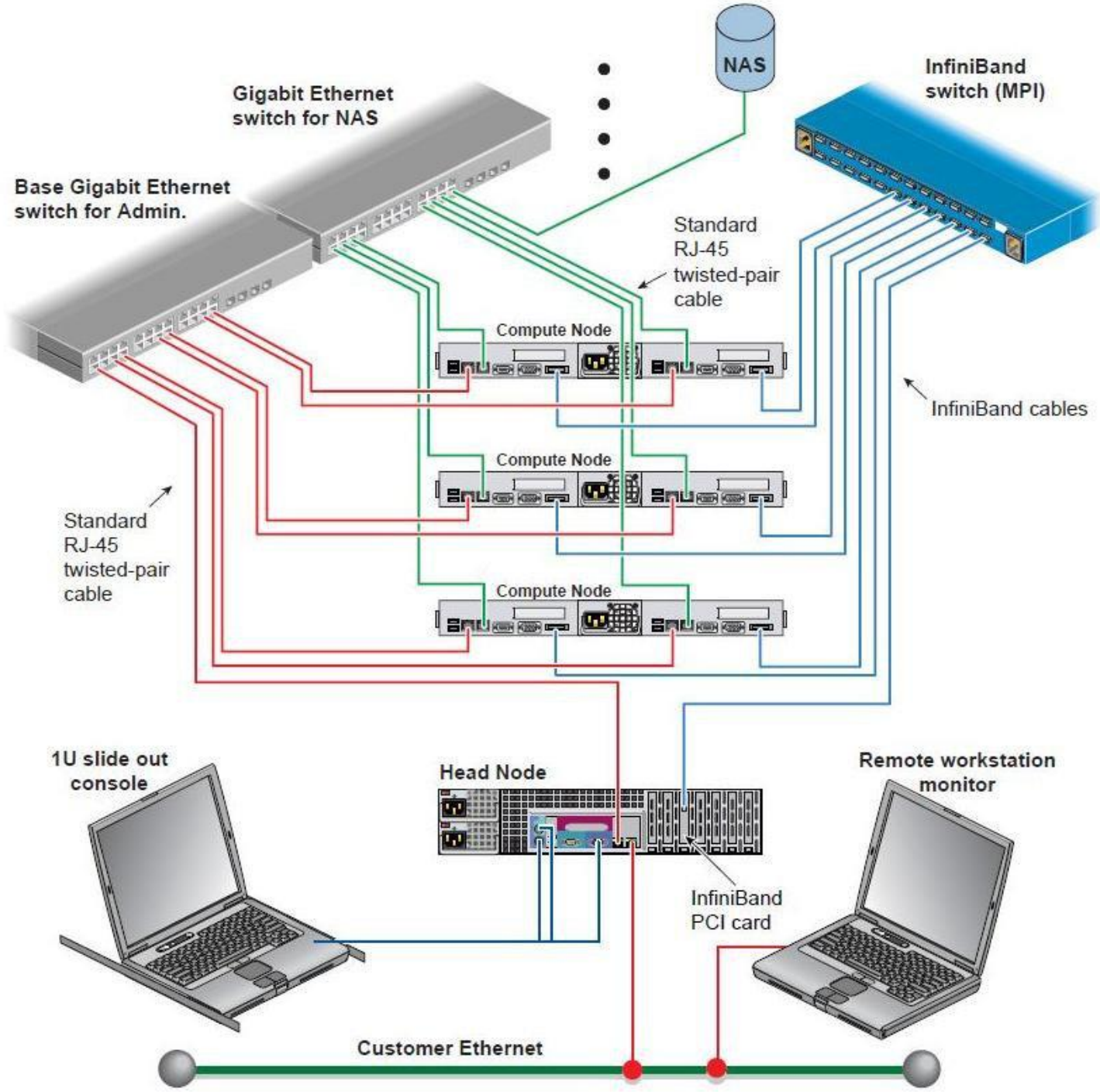
Distributed memory systems


Architecture
Instruction Flow
Data Flow
SISD
SIMD
MISD
MIMD
SPMD
Shared Memory
Distributed Memory
Supercomputers
MPP
Distributed Systems

Programming Distributed Memory systems

Need for a different model than shared memory systems

- loosely coupled systems
- no (physical) shared memory
- communication/coordination by hand or using a framework/tool



Message Passing Interface

A unified approach to the design and implementation of distributed applications

- an industry standard, stewarded by the MPI Forum (over 40 organizations)
- similar roles (different tools) as OpenMP for shared memory programming

A standard of distributed applications (not only) for HPC

- apps very often (but not exclusively) following the SPMD model
- syntax and semantics of a standard set of library functions covering common communication scenarios
- Interface and implementation de-coupled
  - Open-source vs. Commercial MPIs
- Bindings) for C/C++, Fortran 77/95
- M(VA)PICH(2), LAM/MPI, OpenMPI

MPI: Main features

Standardization

- the only message passing library that can be considered a standard. It is supported on virtually all HPC platforms, has replaced all previous message passing libraries.

Portability

- Little or no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.

Performance Opportunities

- Vendor implementations can exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.

Functionality

- over 430 routines defined in MPI-3 (superset of MPI-2 and MPI-1). Most MPI programs can be written using a dozen or less routines.

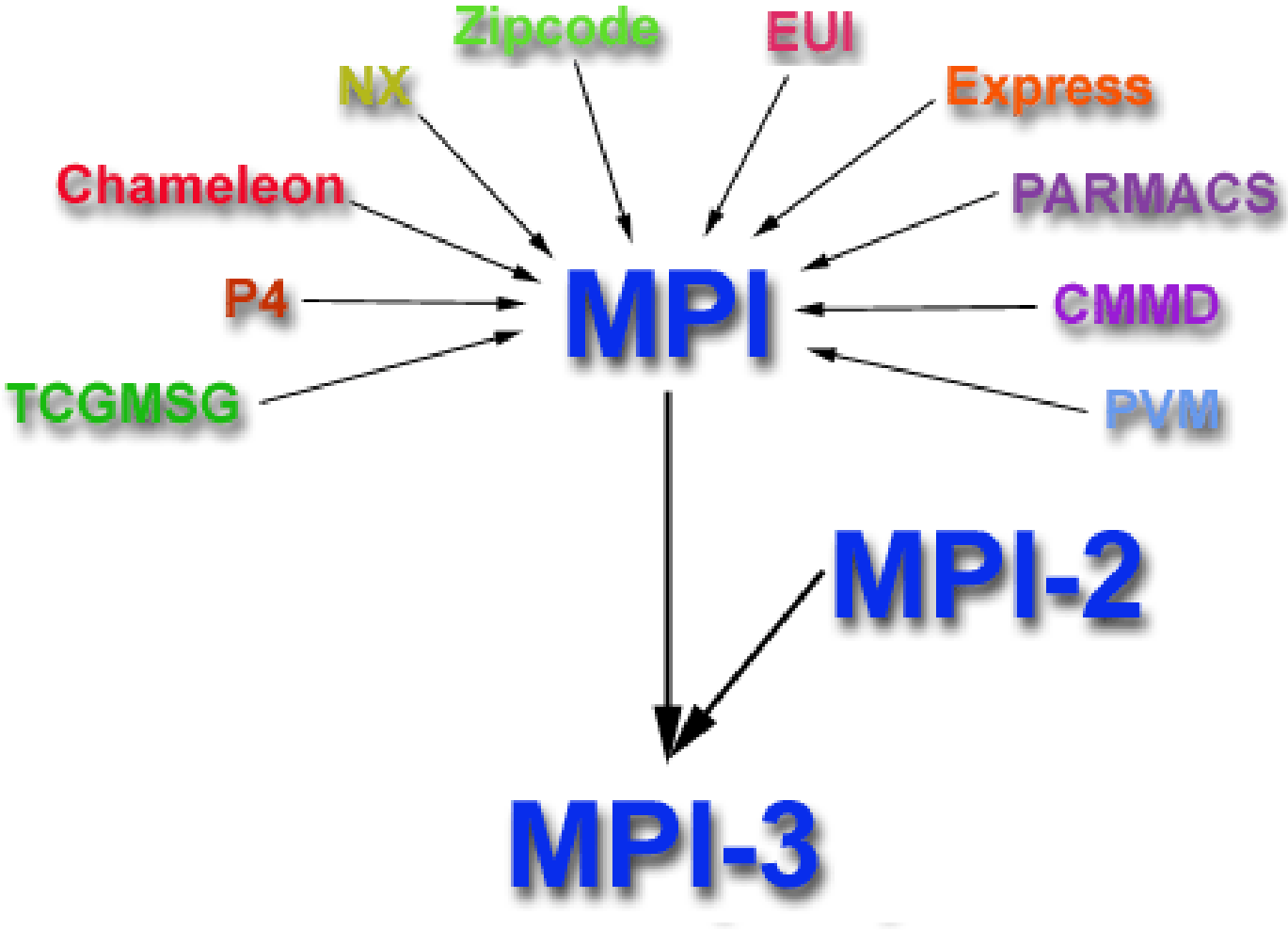
MPI: Historical perspective

Apr 1992: Workshop on Standards for Message Passing in a Distributed Memory Environment, sponsored by the Center for Research on Parallel Computing, Williamsburg, Virginia.

Nov 1992: Working group meets in Minneapolis. MPI draft proposal (MPI1), formation of the MPI Forum

Nov 1993: draft MPI standard presented at Supercomputing'93

- **May 1994:** Final version of MPI-1.0 released (1.1: 1995; 1.2: 1997; 1.3: 2008)
- **1998:** MPI-2 (2.1: 2008; 2.2: 2009)
- **Sep 2012:** The MPI-3.0 (3.1: 2015)



MPI: An application and its world

- collection of processes that exchange data in the form of messages
- (usually) a general-purpose SPMD application that can scale well
- coordination of IO operations
- the communicator and group concepts to define the hierarchy of communicating processes

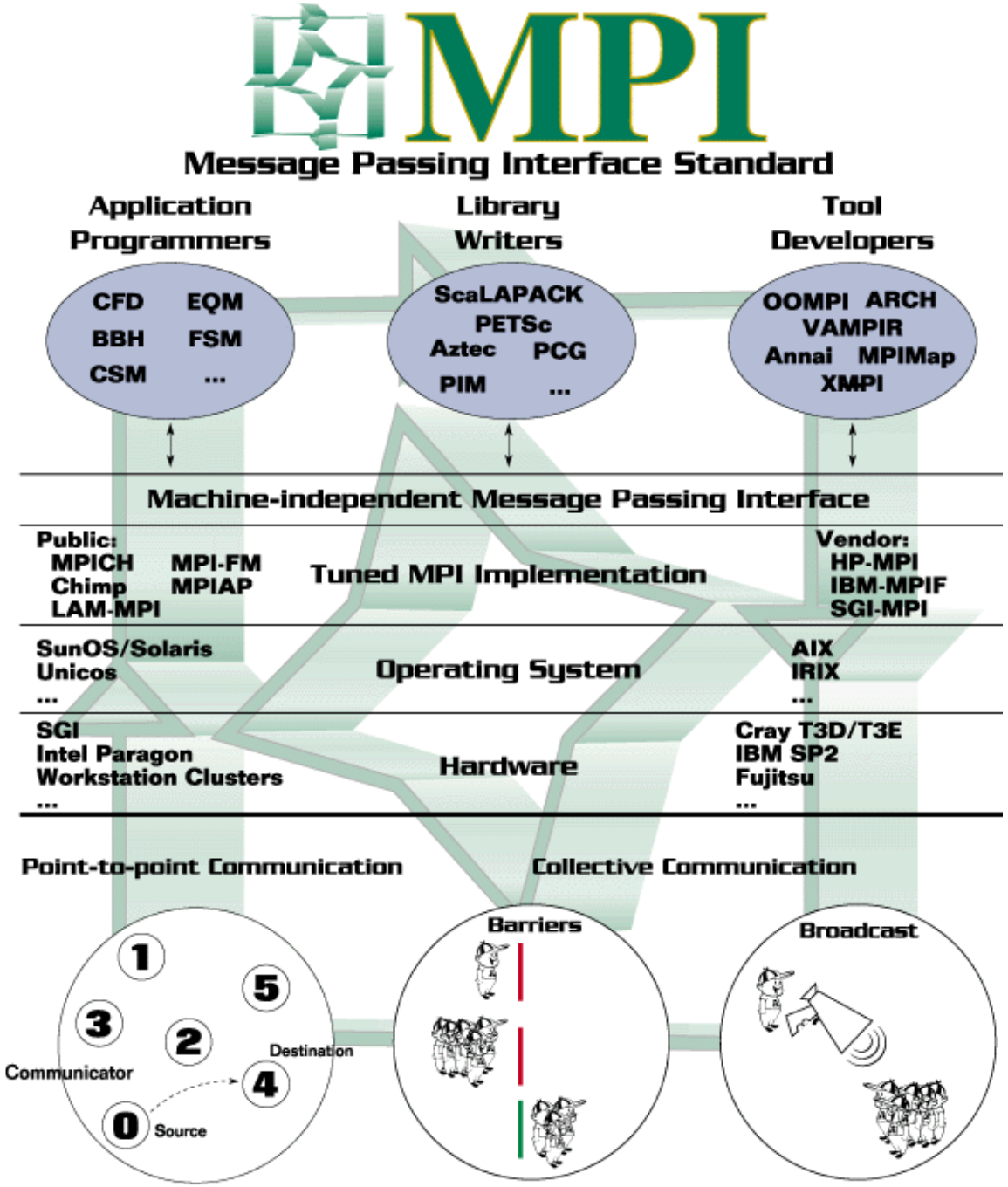
The MPI Ecosystem

Rank

- an ID of a process within a communicator (numerical, starting 0)
- source and destination ‘address’ of messages

Size

- no. of processes within a communicator



MPI operations and communication

blocking operations
- when they return, all resources are ready for another use
- all state changes are finished

non-blocking operations
- may return before all operations are finished
- call of a non-blocking operation initiates the operation

synchronous comm.
- sending is finished when receiving process gets the message

asynchronous comm.
- no synchronization of sending and receiving

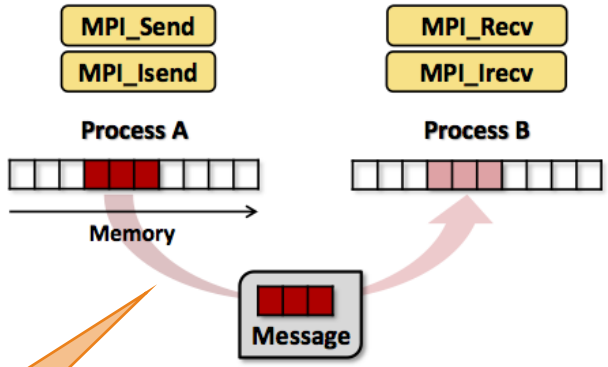
Point-2-point communication

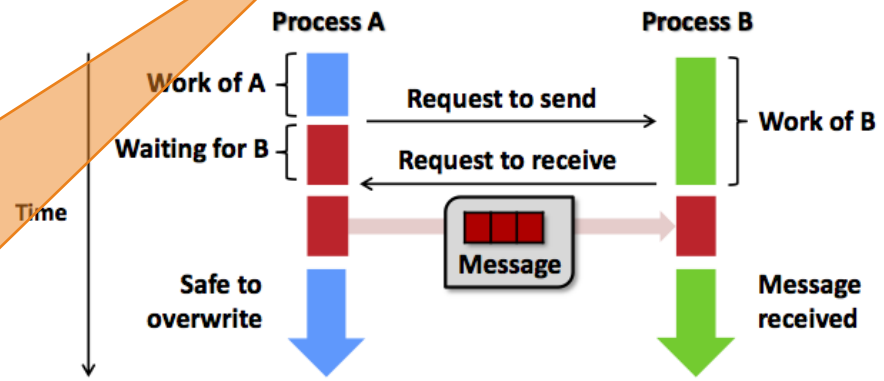
Information exchange between 2 MPI ranks

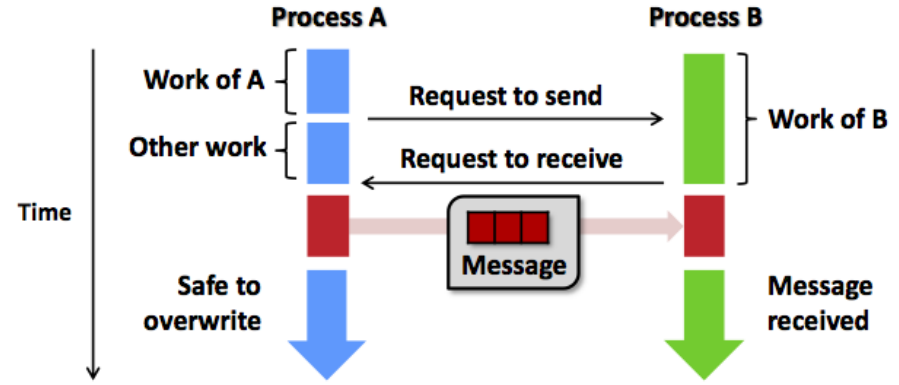
Workflow

- Initialization
  - *MPI_Init*, *MPI_Comm_rank*,
  - *MPI_Comms ize*
- Message transmission
  - *MPI_Send*, *MPI_Receive*,
  - *MPI_Sendrcv* - blocking
- Finalization and clean-up
- vs. non-blocking, sync.
- vs. async.

MPI Datentyp	C-Datentyp
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wide char
MPI_PACKED	special data type for packing
MPI_BYTE	single byte value



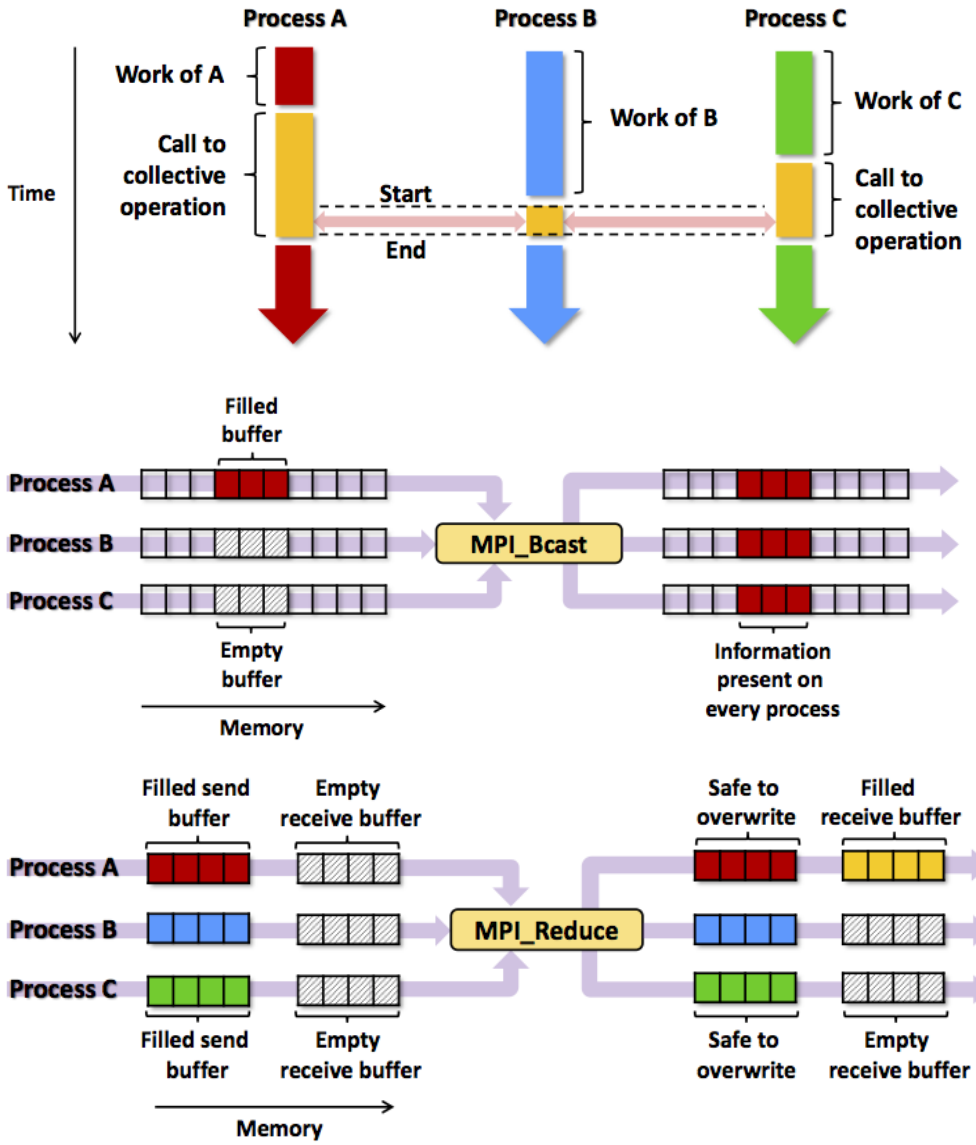




Collective and global communication

Information exchange between all ranks in a communicator

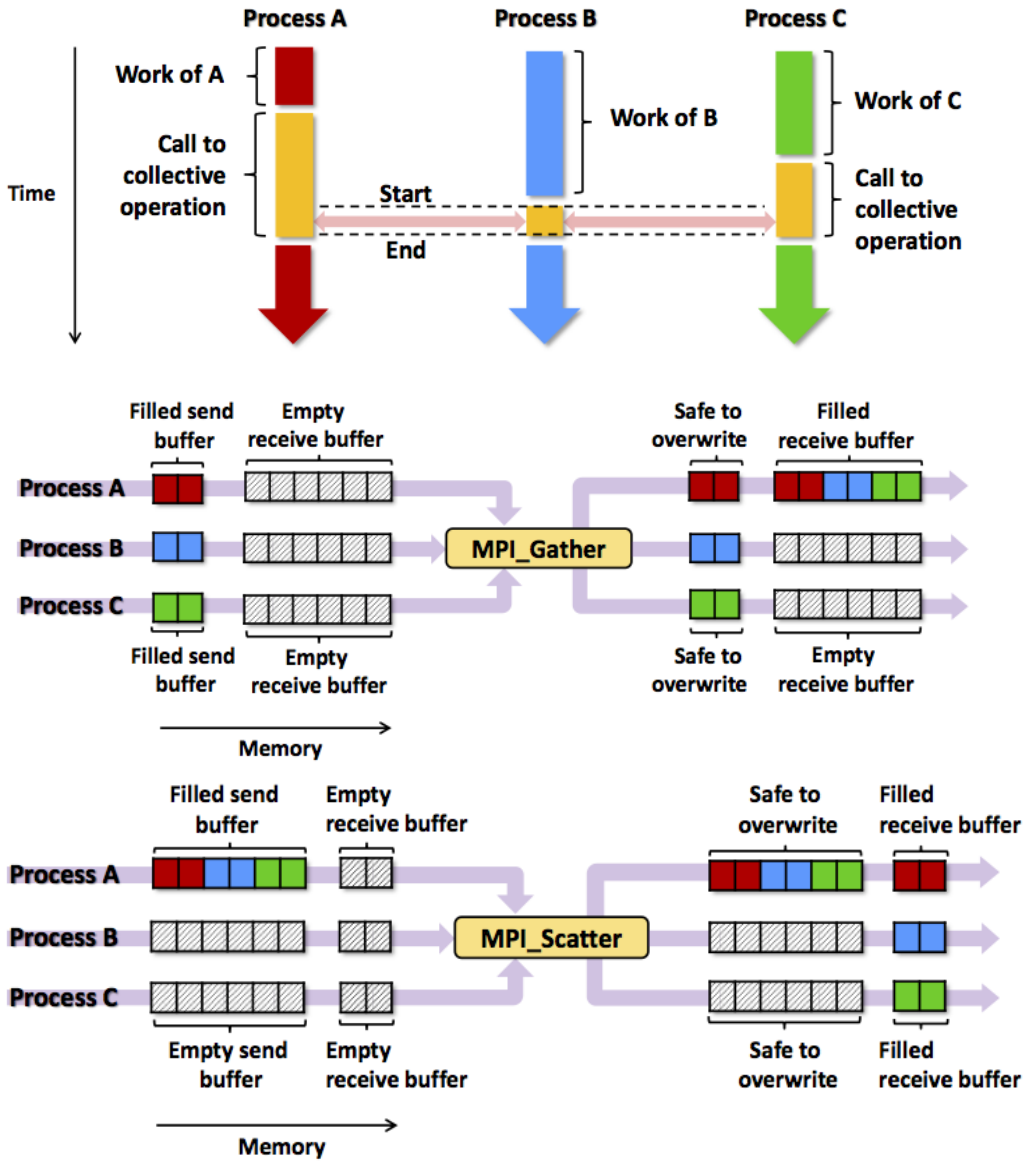
- broadcasts (*MPI_Bcast*())
- reductions (*MPI_Reduce*) - operations: max, min, sum, logic, bit ops.,
- user defined reductions
- gather (*MPI_Gather*)
- scatter (*MPI_Scatter*)



Collective and global communication

Information exchange between all ranks in a communicator

- broadcasts (*MPI_Bcast*())
- reductions (*MPI_Reduce*) - operations: max, min, sum, logic, bit ops.,
- user defined reductions
- gather (*MPI_Gather*)
- scatter (*MPI_Scatter*)



MPI Caveats

- order of messages (not guaranteed)
- deadlocks
- implementation and runtime agnostic programs

MPI Implementations

Open MPI

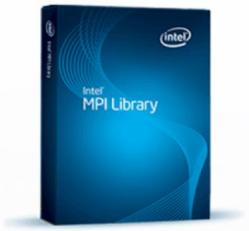
Roadrunner (USA)
K computer (Japan)

MPICH

Tianhe-2 (Chiny)

9/10 TOP 500

Stampede (USA)
Pleiades (USA)



Intel MPI

Amazon EC2 C3

IBM

IBM Platform MPI

Partitioned Global Address Space

Partitioned global address space

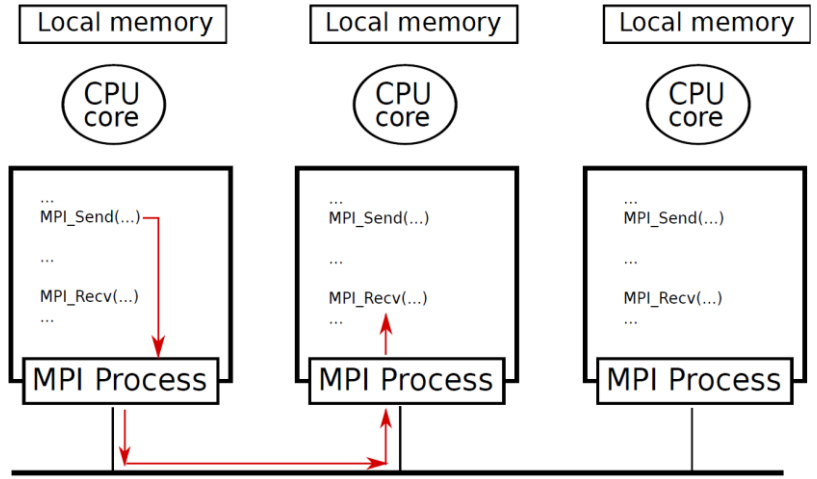
Motivation

- MPI provides an industry standard for programming of distributed systems
- its bare-bones nature makes the app development for MPI harder than dev. for shared memory systems (longer code = more space for errors)
- there are efforts to bring shared memory programming concepts to distributed memory programming
- data and computation locality must be considered

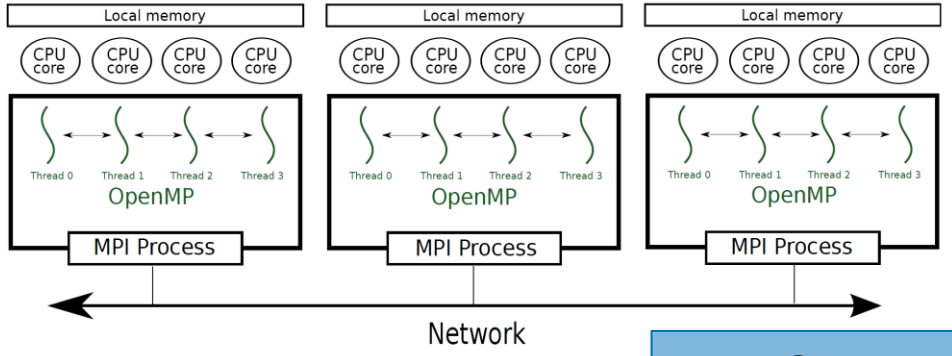
Partitioned global address space

Partitioned global address space (PGAS)

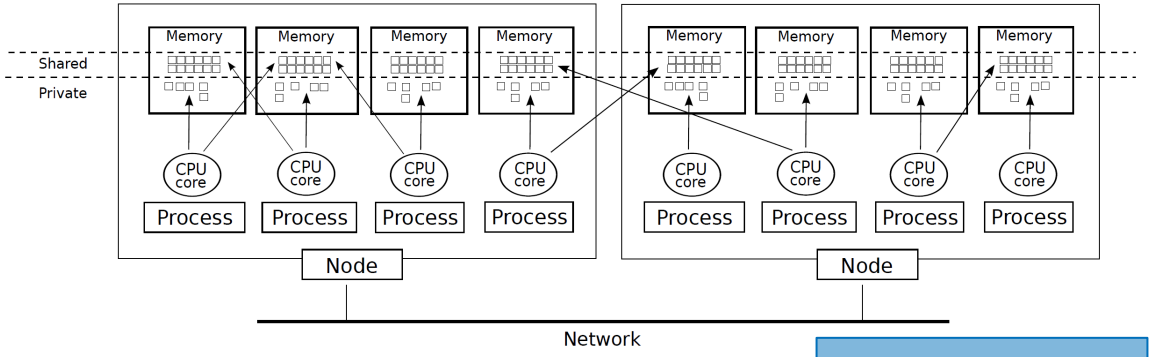
- an abstract parallel programming model implemented by several languages
- built around the concept of (virtual) distributed shared memory



MPI



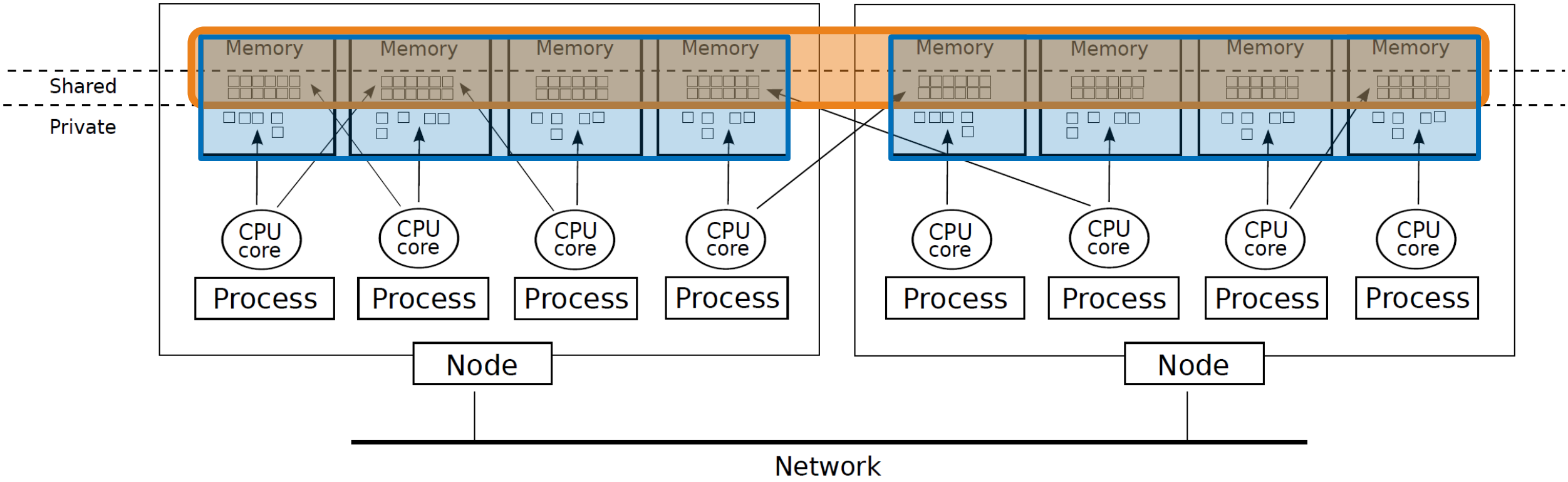
MPI+OpenMP



PGAS

PGAS
Partitioned Global Address Space

Partitioned global address space



PGAS

Partitioned Global Address Space

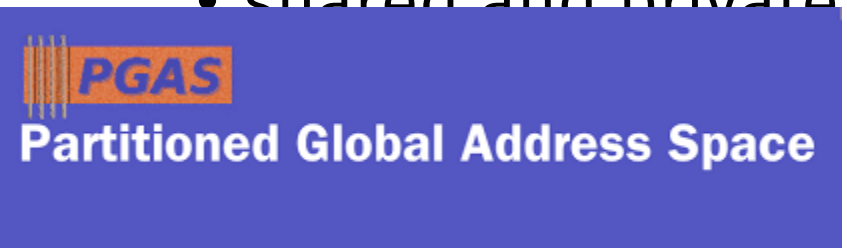
PGAS application

Structure

- each application is composed of multiple threads, each of them knows its identity
- barriers, loop work sharing, parallel control libraries

Memory

- private and shared memory
- each thread has its own private memory
- each thread can access any data in the shared memory; this, however, might be much more expensive
- shared and private pointers



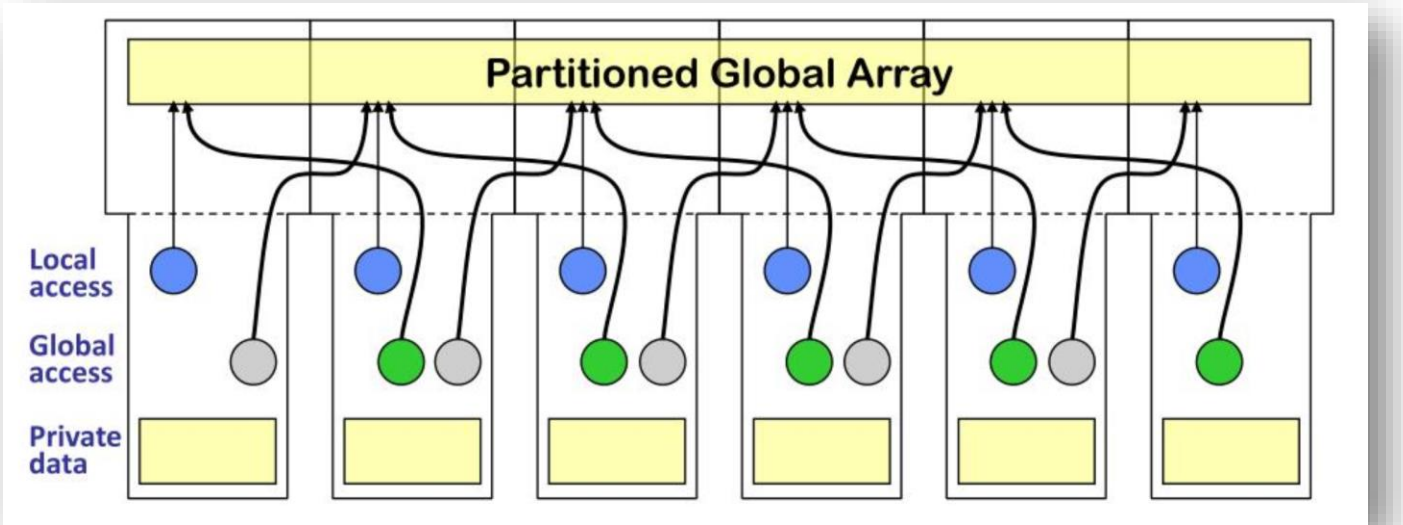
PGAS properties

Global address space memory model

- Each thread can write memory anywhere in the system (convenience of shared memory)

Information about the locality of data

- Some data is (guaranteed) local, some is global, potentially further away (locality and scalability of message passing)



PGAS
Partitioned Global Address Space

PGAS design goals

Application

- convenient distributed programming of irregular codes
  - graphs, Hash tables, Sparse matrices, Adaptive (hierarchical) meshes

Hardware

- expose the best available performance on a given machine
  - low latency for small messages
  - high bandwidth even for medium sized messages
  - high injection bandwidth

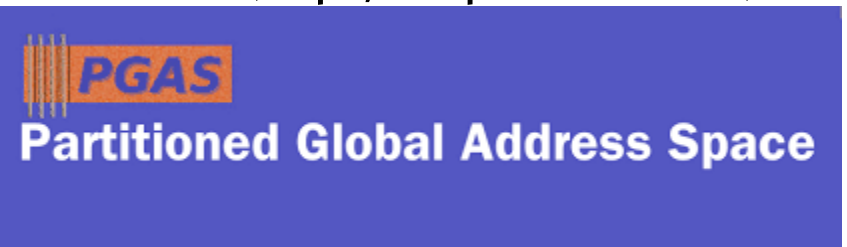
PGAS communication backbone(s)

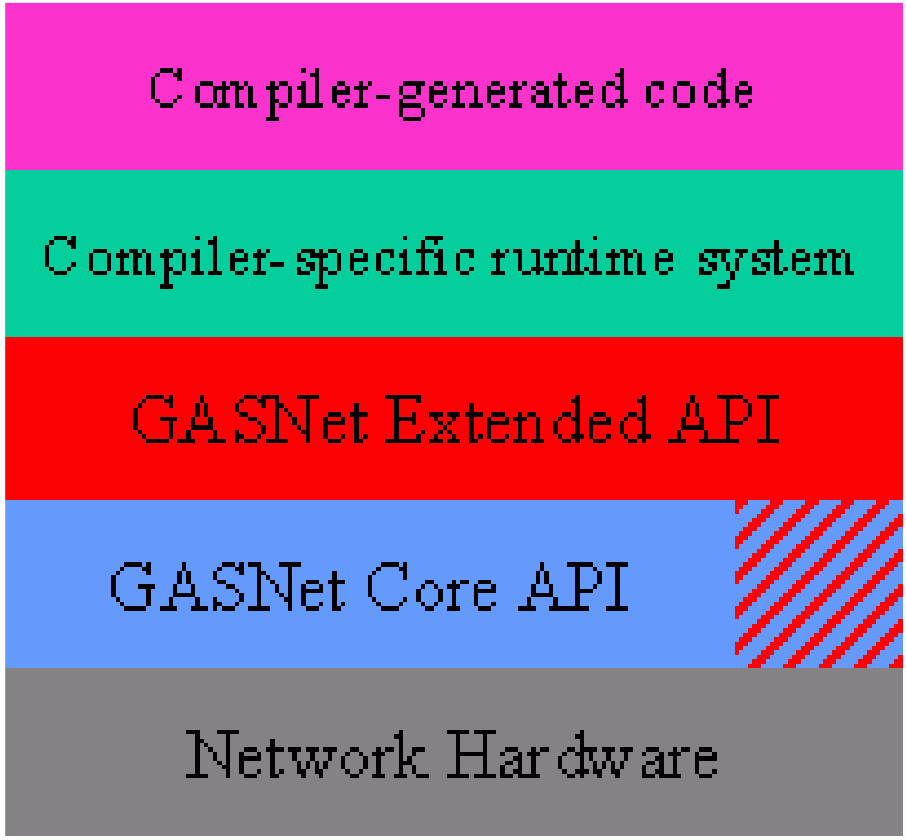
PGAS languages are built on top of several low-level communication libraries

- Primitives that implement Remote Memory Access (for PGAS)
- MPI, OpenSH-MEM, ARMCI, GASNet

GASNet

- language independent, low-level networking
- network-independent high-performance communication primitives
- support of HPC networking hardware and standards (IVB verbs, Cray Gemini interconnects)
- support for portable networking (mpi, udp conduits)





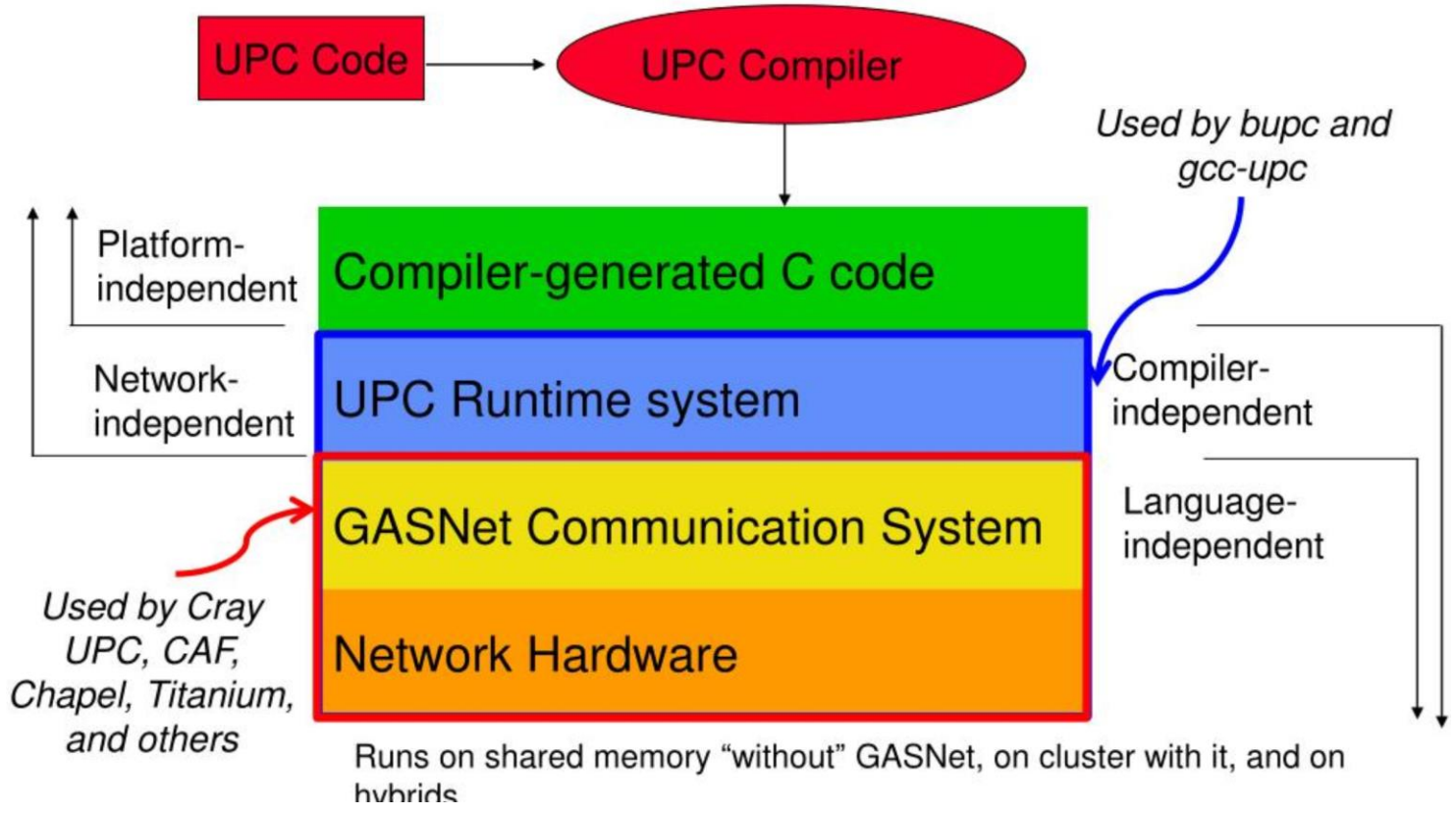
PGAS languages

Language	Parallel Execution	Topology	Data Distribution	Distributed Data	Remote Access	Array Indexing
Original PGAS languages						
CAF	SPMD	User defined mesh	Implicit	Regular	Explicit	Local
Titanium	SPMD	Flat ordered set	Explicit	Irregular	Expl. + Impl.	not applicable
UPC	SPMD	Flat ordered set	Explicit	Reg. + Irreg.	Implicit	Global
HPCS PGAS languages						
Chapel	APGAS + Impl.	User defined mesh	Explicit	Reg. + Irreg.	Expl. + Impl.	Global
X10	APGAS	Flat ordered set	Explicit	Reg. + Irreg.	Explicit	Global
Fortress	APGAS + Impl.	Hierarchical	Explicit	Reg. + Irreg.	Expl. + Impl.	Global
Retrospective PGAS languages						
HPF	Implicit	User defined mesh	Explicit	Regular	Implicit	Global
ZPL	Implicit	User defined mesh	Implicit	Regular	Explicit	Global
GA	SPMD	Flat ordered set	Explicit	Regular	Explicit	Global
Recent PGAS languages						
XCalableMP	SPMD	Flat ordered set	Explicit	Regular	Explicit	Global

- Coarray Fortran
- UPC
- UPC++

An example based on UPC

- a minimalistic parallel extension to ANSI C implementing PGAS
- shared arrays sliced by blocks (default block size = 1)



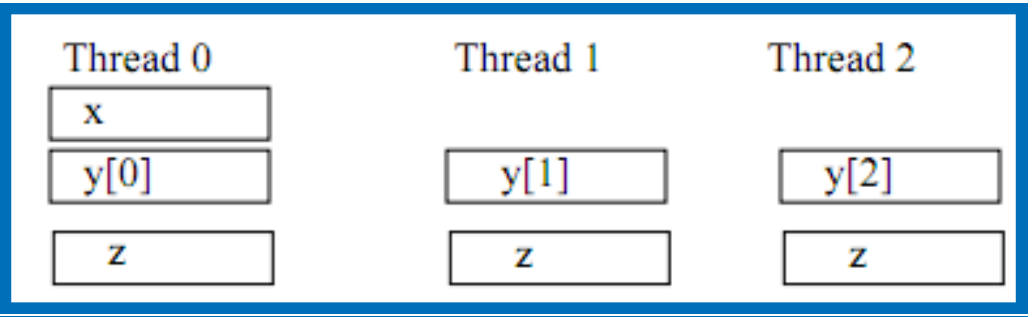
PGAS
Partitioned Global Address Space

An example based on UPC

- a minimalistic parallel extension to ANSI C implementing PGAS
- shared arrays sliced by blocks (default block size = 1)

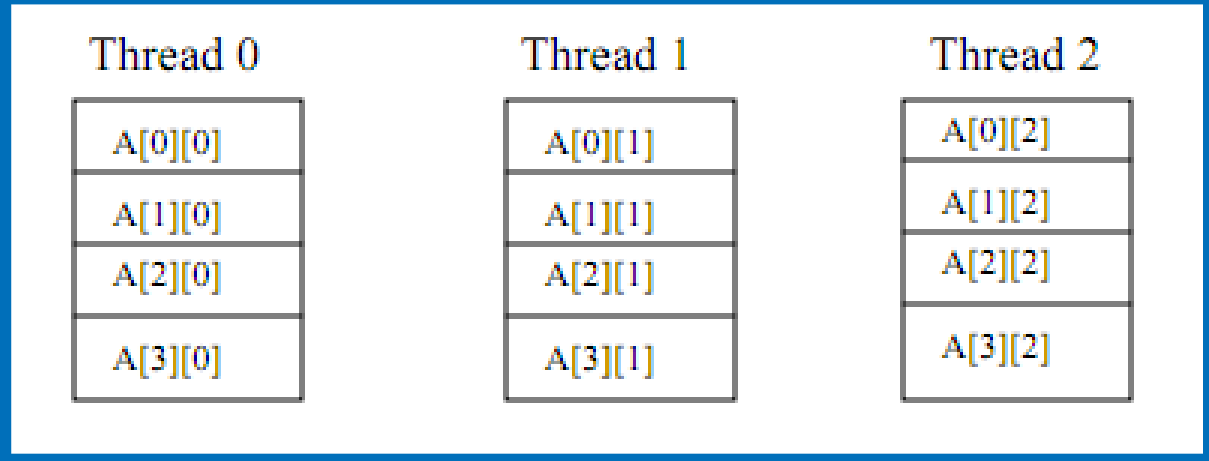
Example

```
shared int x;
shared int y[THREADS];
int z;
```



Example

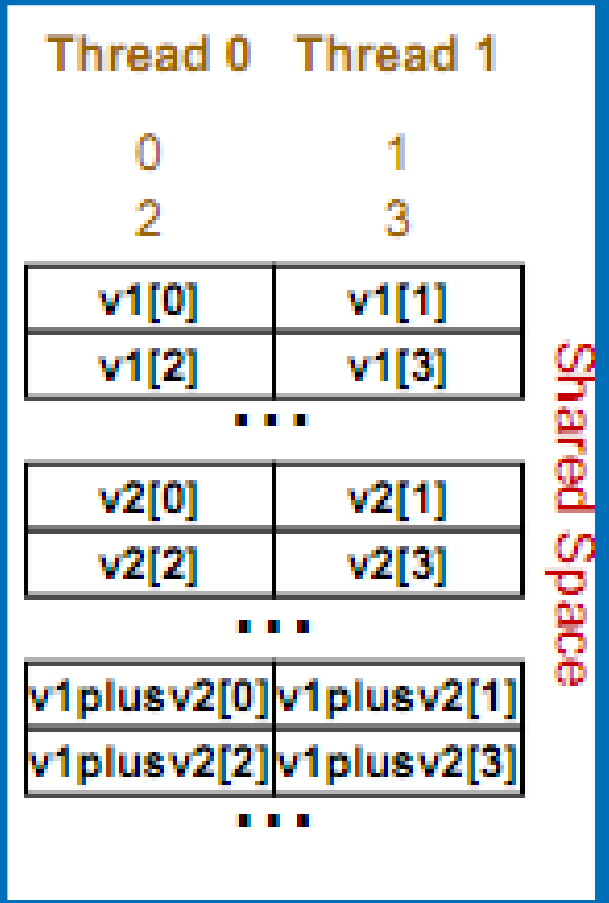
```
shared int A[4][THREADS];
```



An example based on UPC

Example

```
#include <upc_relaxed.h>
#define N 100* THREADS
shared int v1[N], v2[N], v1v2[N];
void main()
{
  int i;
  for(i=THREAD; i < N; i+=THREADS)
    v1v2[i] = v1[i] + v2[i];
}
```

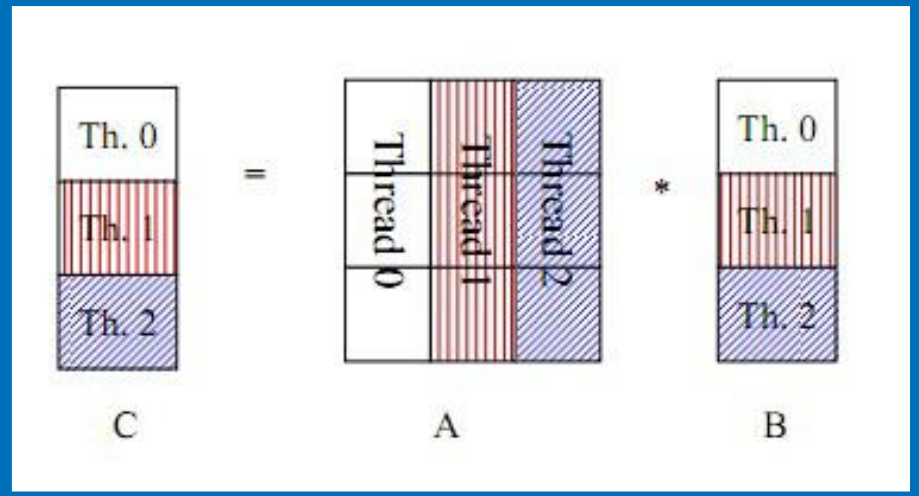


An example based on UPC

Example

```
#include <upc_relaxed.h>
#define N 100* THREADS

shared int a [THREADS] [THREADS];
shared int b [THREADS], c [THREADS];
void main()
{
  int i, j;
  upc_forall(i=0; i < N; i++; i)
  {
     c[i] = 0;
     for (j = 0; j< THREADS; j++)
       c[i] += a[i][j] * b[j];
  }
}
```

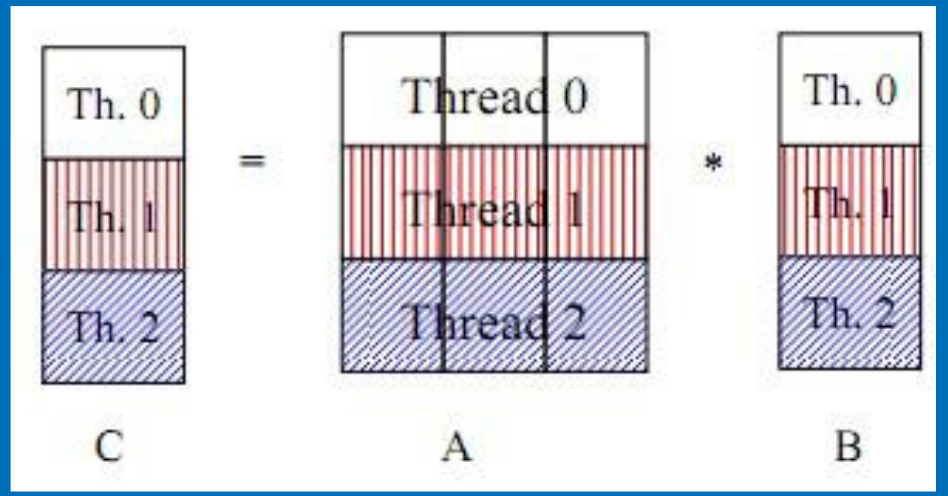


An example based on UPC

Example

```
#include <upc_relaxed.h>
#define N 100* THREADS

shared [THREADS] int a [THREADS] [THREADS];
shared int b [THREADS], c [THREADS];
void main()
{
  int i, j;
  upc_forall(i=0; i < N; i++; i)
  {
     c[i] = 0;
     for (j = 0; j< THREADS; j++)
       c[i] += a[i][j] * b[j];
  }
}
```



Recent development in PGAS

- UPC++
  - a template-based approach, “compiler-free” PGAS
- XCalableMP
  - a directive-based approach; omni-compiler
- Parallel Computing in Java
  - Java style PGAS

PCJ Parallel Computing in Java

PGAS
Partitioned Global Address Space

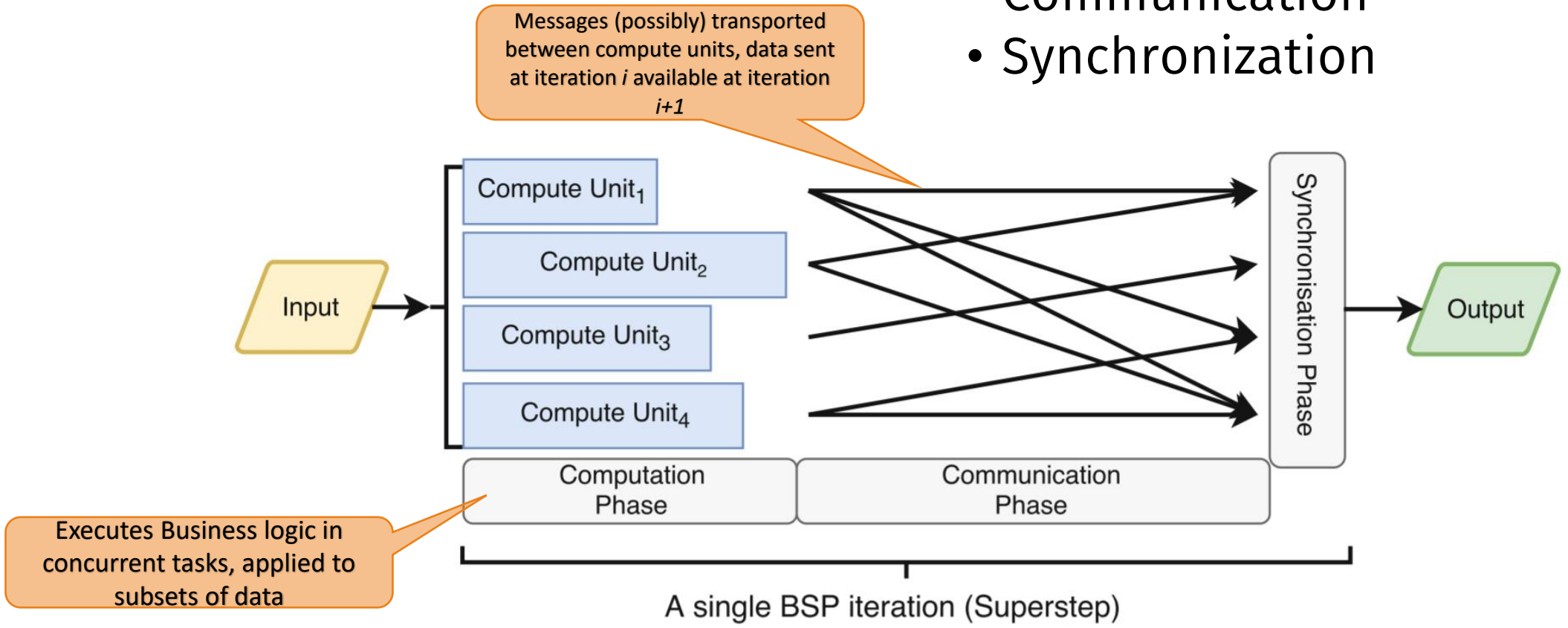
Bulk Synchronous Parallel

Bulk Synchronous Parallel

An iterative share-nothing parallel computing model

Three stages of parallel computation, repeated until completion:

- Computation
- Communication
- Synchronization



Bulk Synchronous Parallel

BSP advantages

- simplifies parallel computation
- automated balancing and scaling
- suitable for heterogeneous environments

BSP disadvantages

- restrictive model (code must be autonomous and independent)
- explicit synchronization hard to achieve
- slow computing units slow the whole computation

Sounds familiar?

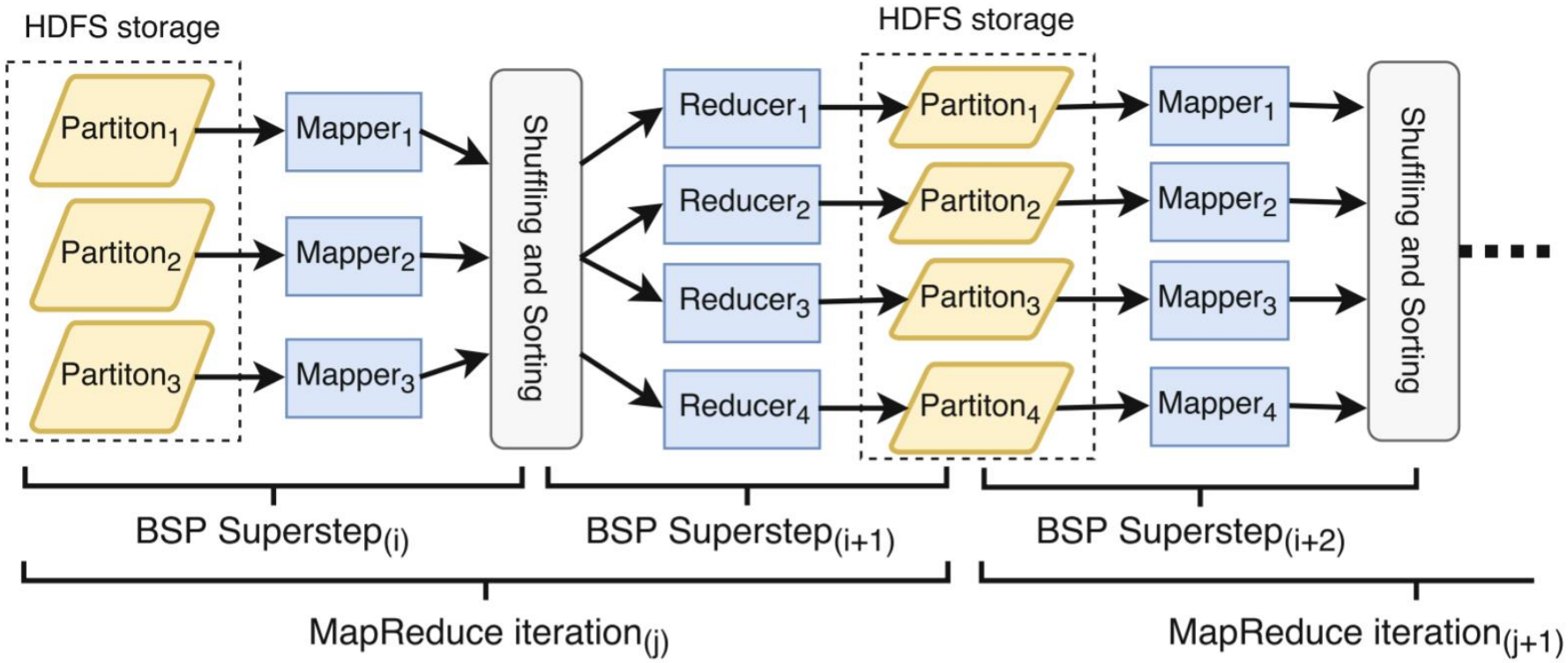
MAPREDUCE . . .

But also Pregel, SPARK etc.

MapReduce

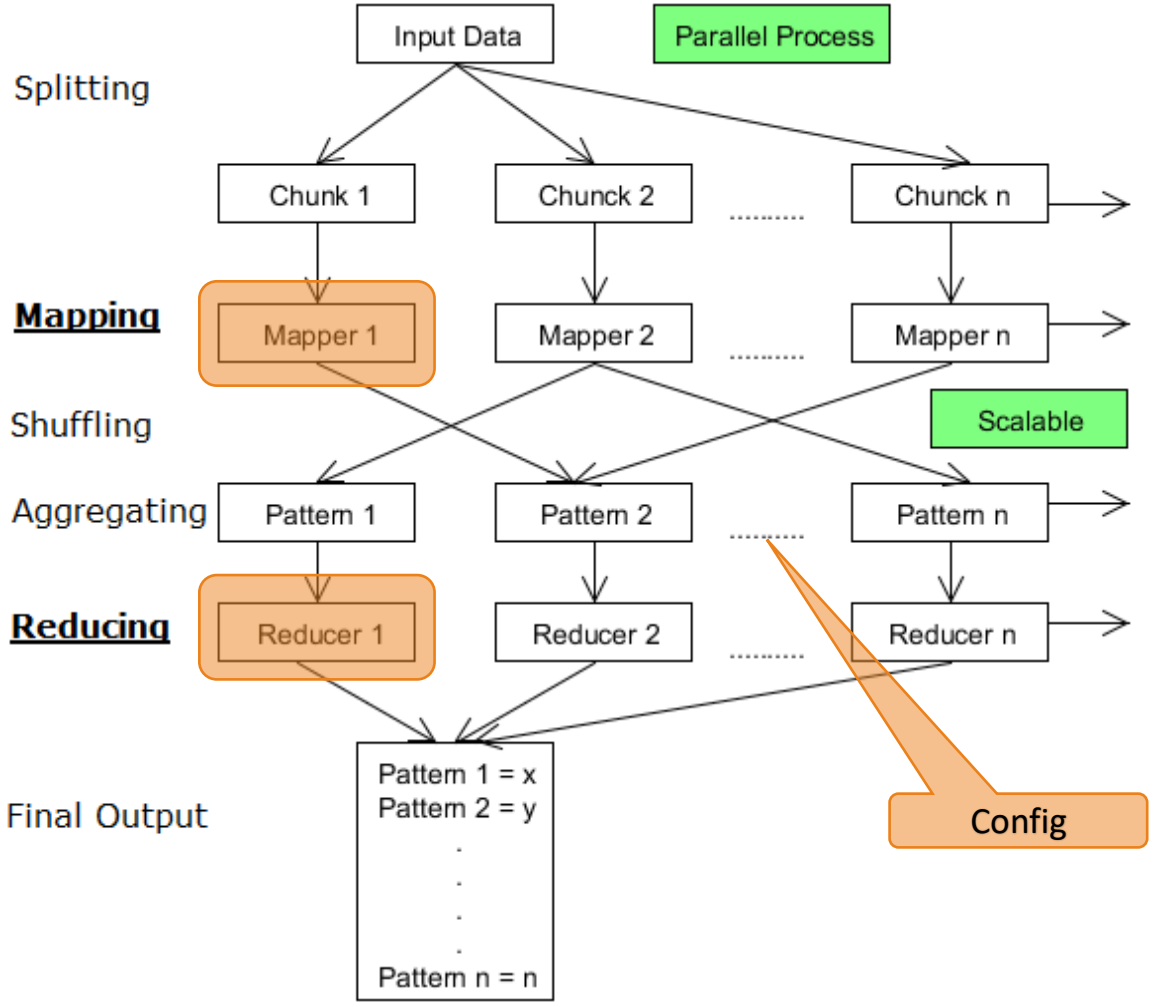
A general-purpose distributed system with automatic scalability and fault tolerance

Achieved through 2 user-defined operators, mappers and reducers



Structure of a MapReduce job

- processing of data divided into two phases, mapping and reducing
- map function processes a key/value pair to generate a set of intermediate key/value pairs
- reduce function merges all intermediate values associated with the same intermediate key
- embraces distribution of data and parallel processing as well as aggregation of similar patterns



An Apache Hadoop -based example

Hadoop filesystem

- a write-only filesystem-like application with data distributed across multiple nodes
- supports replication, is fault-tolerant, and highly scalable
- can deal with outages and is optimized for throughput
- implemented using data nodes (store data), namenode (control of metadata) , and secondary namenode (not a backup but bookkeeping)

MapReduce layer

- An API for writing MapReduce workflows in Java
- A set of services for managing the execution of these workflows

Example

```
public void map(Object key, Text value, Context context )
throws IOException, InterruptedException {
  StringTokenizer itr = new
  StringTokenizer(value.toString());
  while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
  }
}
...
job.setCombinerClass(IntSumReducer.class);
...
public void reduce(Text key, Iterable<IntWritable>
values, Context context)
throws IOException, InterruptedException {
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
}
```

Python: DASK

Python: disturbed distribution

The strengths
- de-facto language of data science
- fast and optimized (native code, accelerators)
- an existing ecosystem

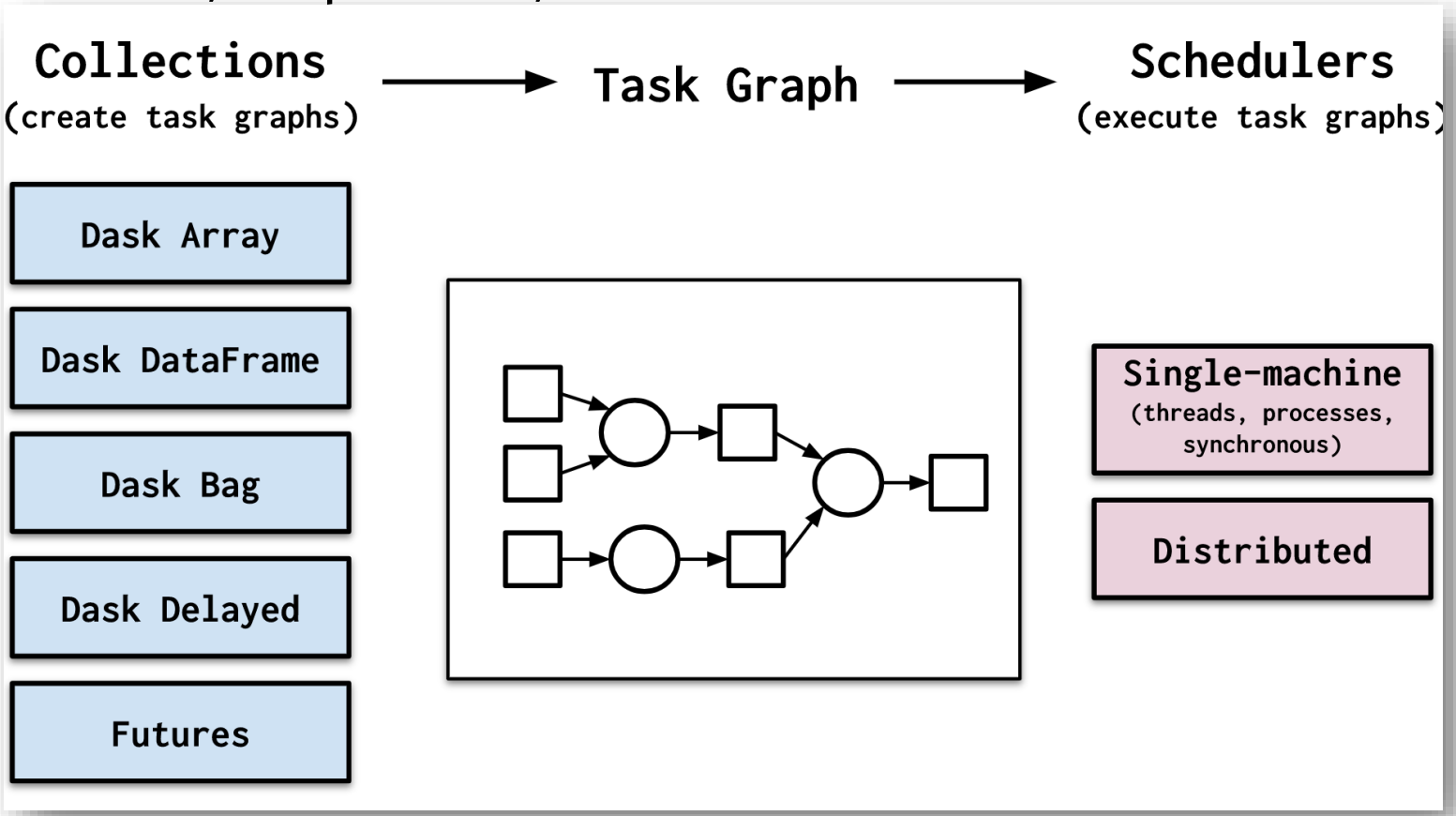
The headaches
- limited to single thread
- limited to in-memory data
- an existing ecosystem

DASK is a flexible parallel computing paradigm for Python

- Scales up and down well, resilient, responsive, realtime

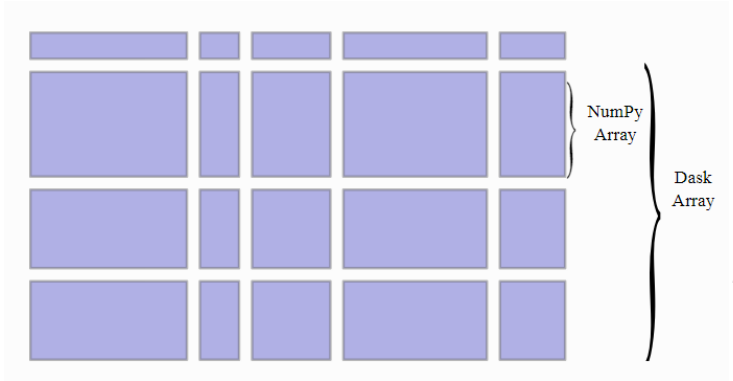
Two essential parts

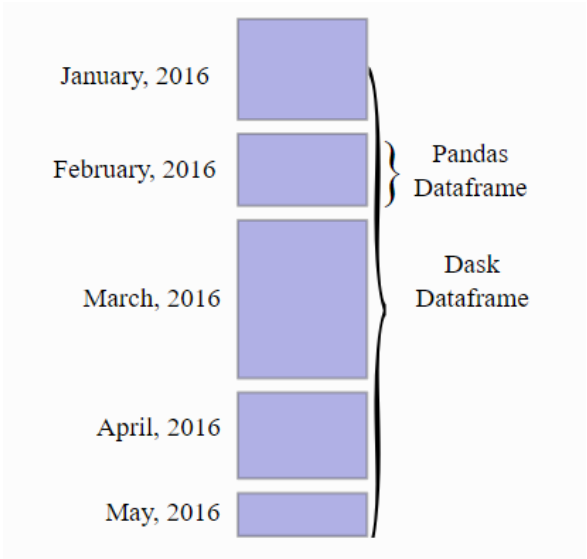
- Dynamic task scheduling
  – optimization of computation
- Big Data Collections
  – parallel containers



DASK

High-level API





DASK Arrays
[*Parallel NumPy*]

DASK Bag
[Parallel Lists]

DASK DataFrames
[Parallel Pandas]

DASK-ML
[Parallel Scikit-learn]

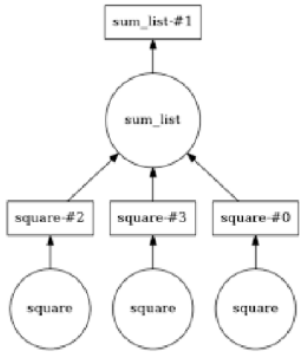
Custom
Algorithms

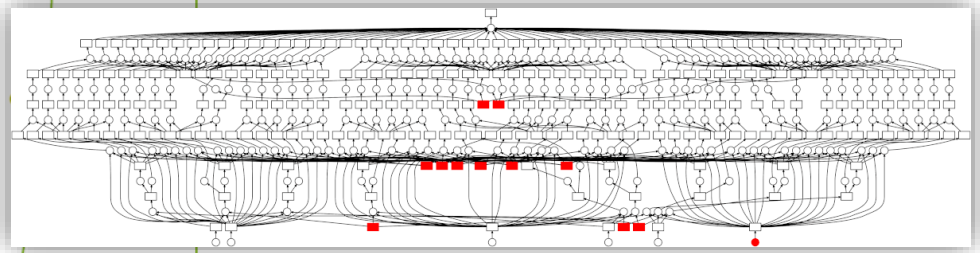
Custom
Graphs

DASK Delayed

DASK Futures

Low(er)-level
API





Task
scheduling

Local Scheduler

Multiprocessing

Synchronous

Threaded

Distributed

Dask.distributed

Centrally managed, distributed, dynamic task scheduler

Works on moderate-sized clusters

Low latency, complex scheduling, data locality

Warning: a relatively new project ...