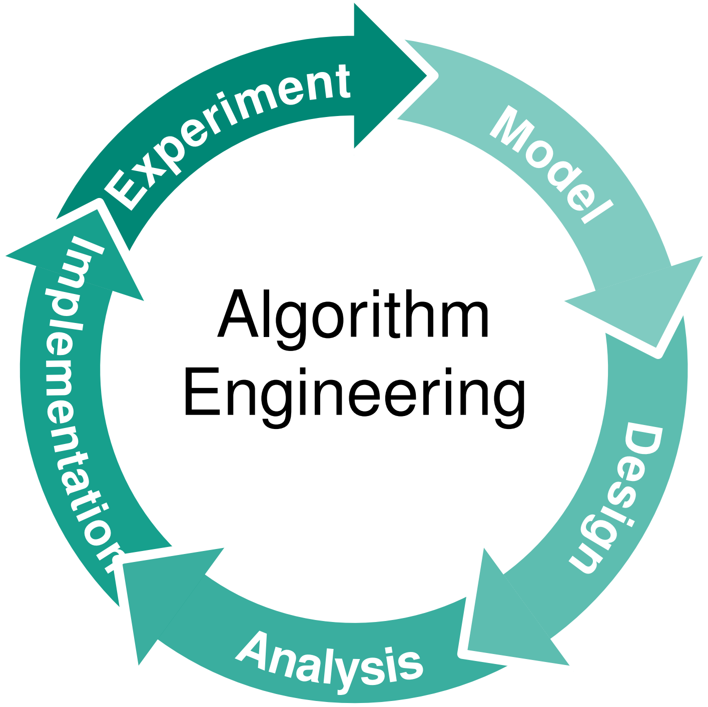
Karlsruhe Institute of Technology

KaMPIng: Flexible and (Near) Zero-Overhead C++ Bindings for MPI

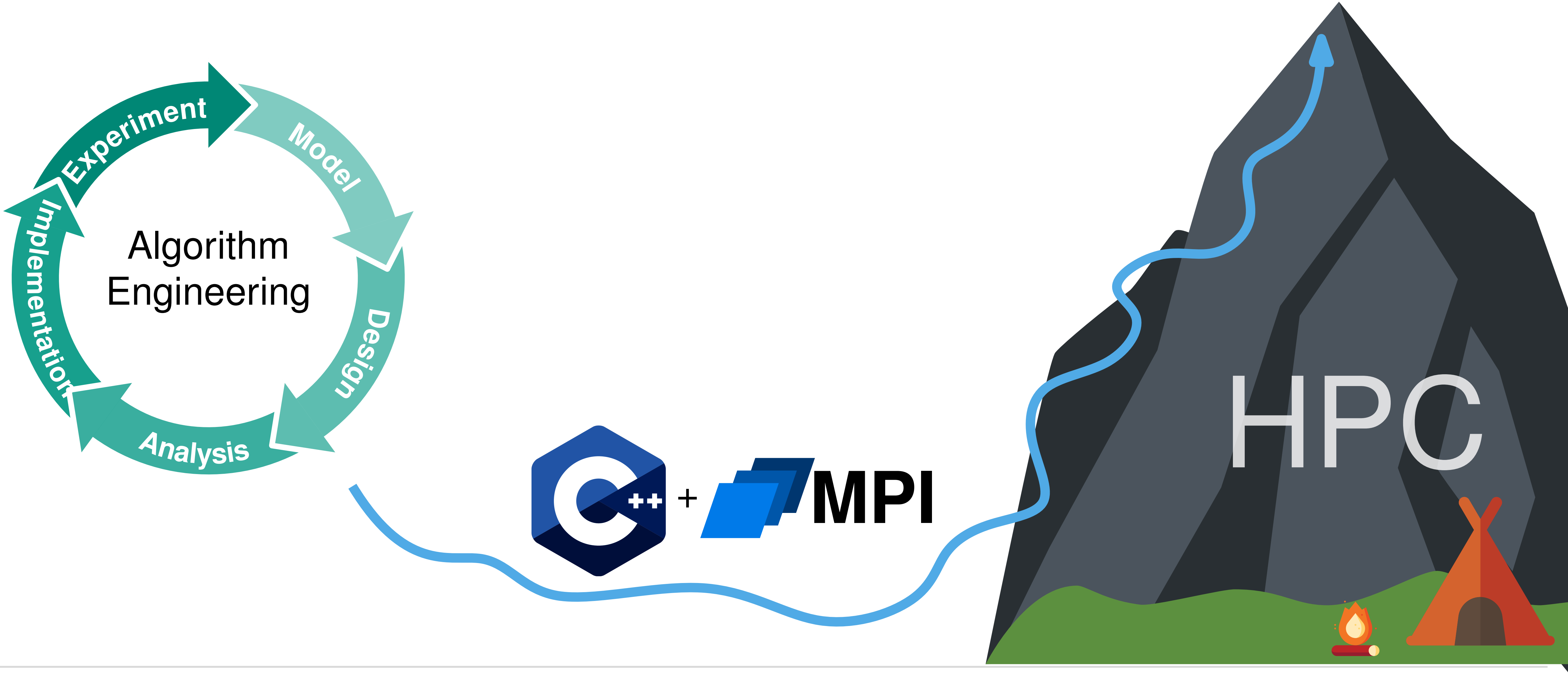
SC'24 · 2024-11-20

Tim Niklas Uhl, Matthias Schimek, Lukas Hübner, Demian Hespe,
Florian Kurpicz, Daniel Seemaier, Christoph Stelz, Peter Sanders

The Trail to HPC



The Trail to HPC



The Trail to HPC

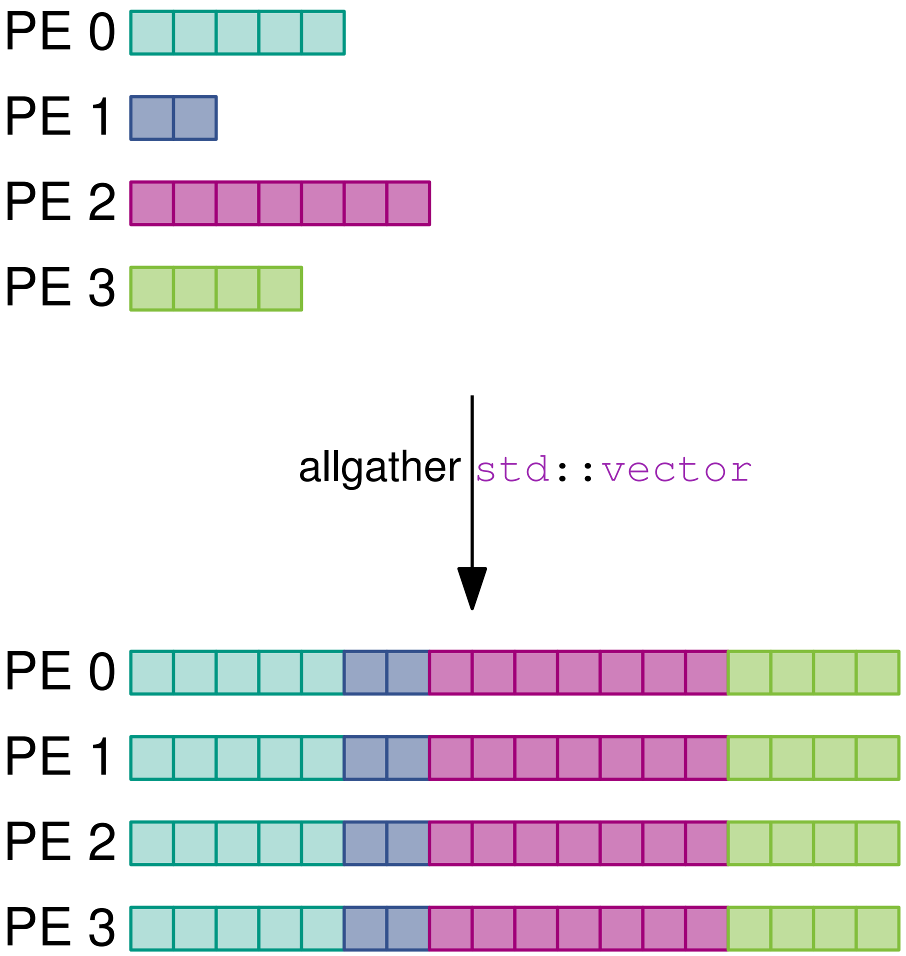
Experiment
Model
Design
Analysis
Implementation
Distributed
Algorithm
Engineering
C++
+
MPI
HPC

The Trail to HPC

The baggage of using C++ + MPI

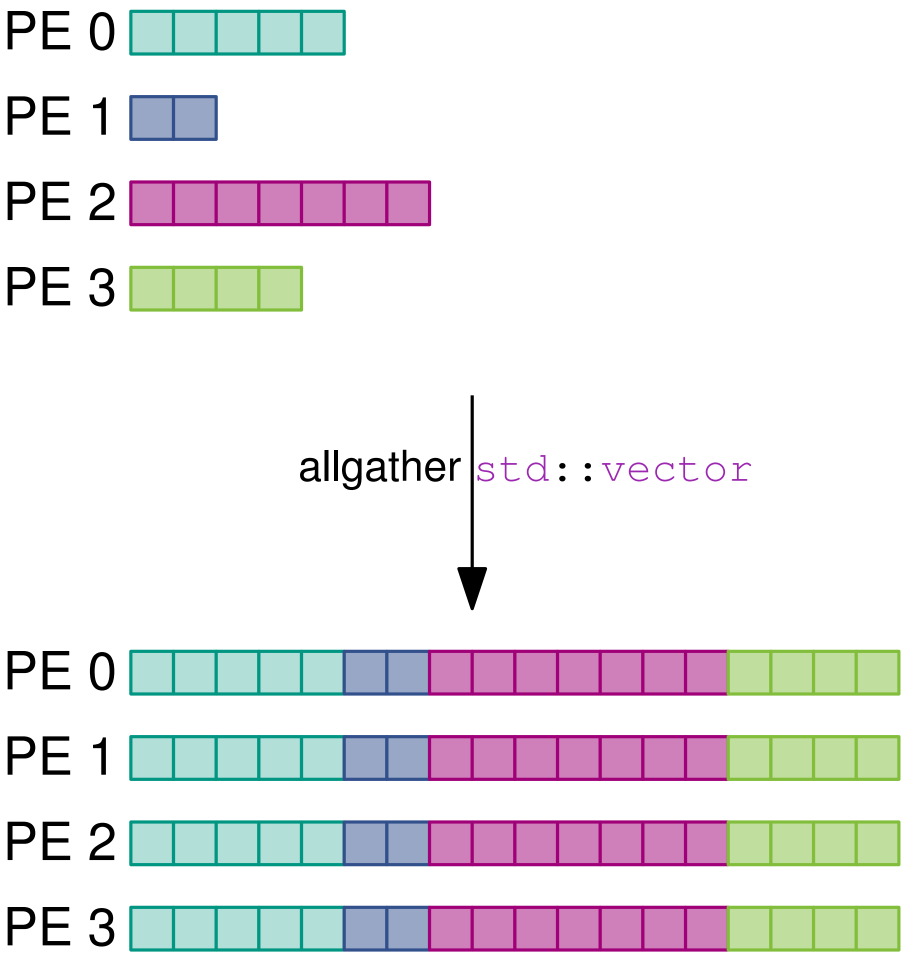
The Trail to HPC

The baggage of using C++ + MPI



The Trail to HPC

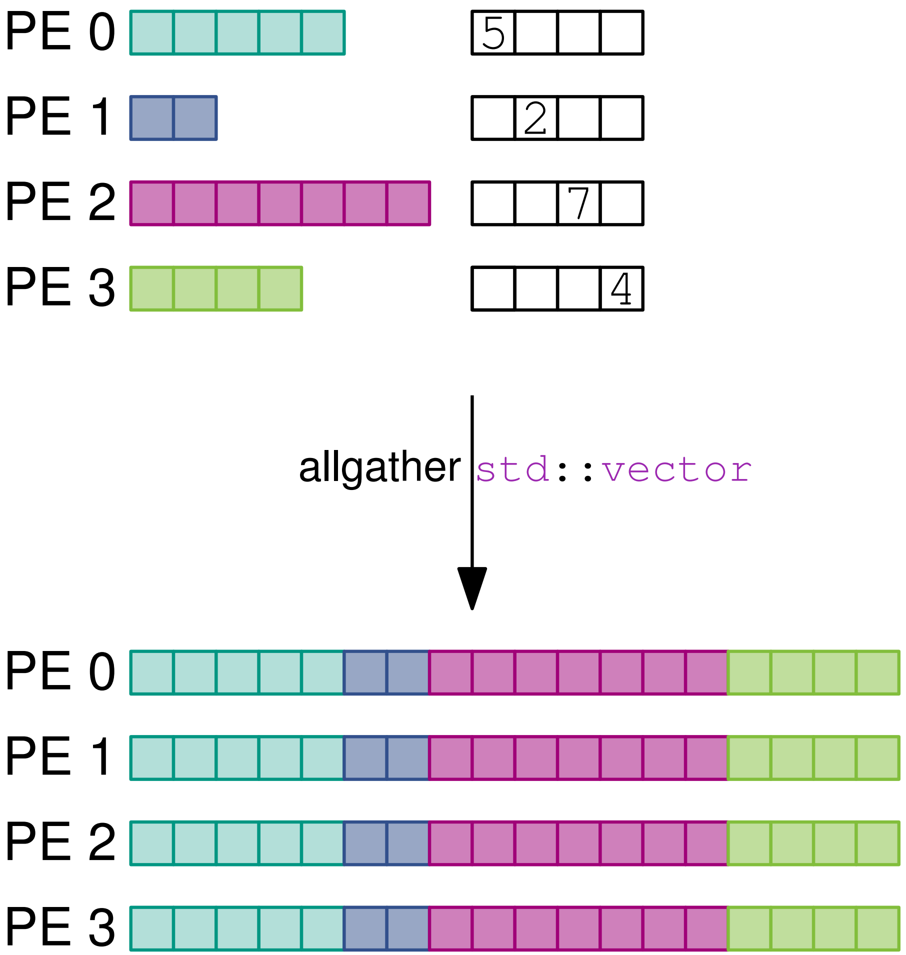
The baggage of using C++ + MPI



```
std::vector<double> get_whole_vector(std::vector<double> const& v_local,
                                     MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0,  MPI_DATATYPE_NULL
                rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

The Trail to HPC

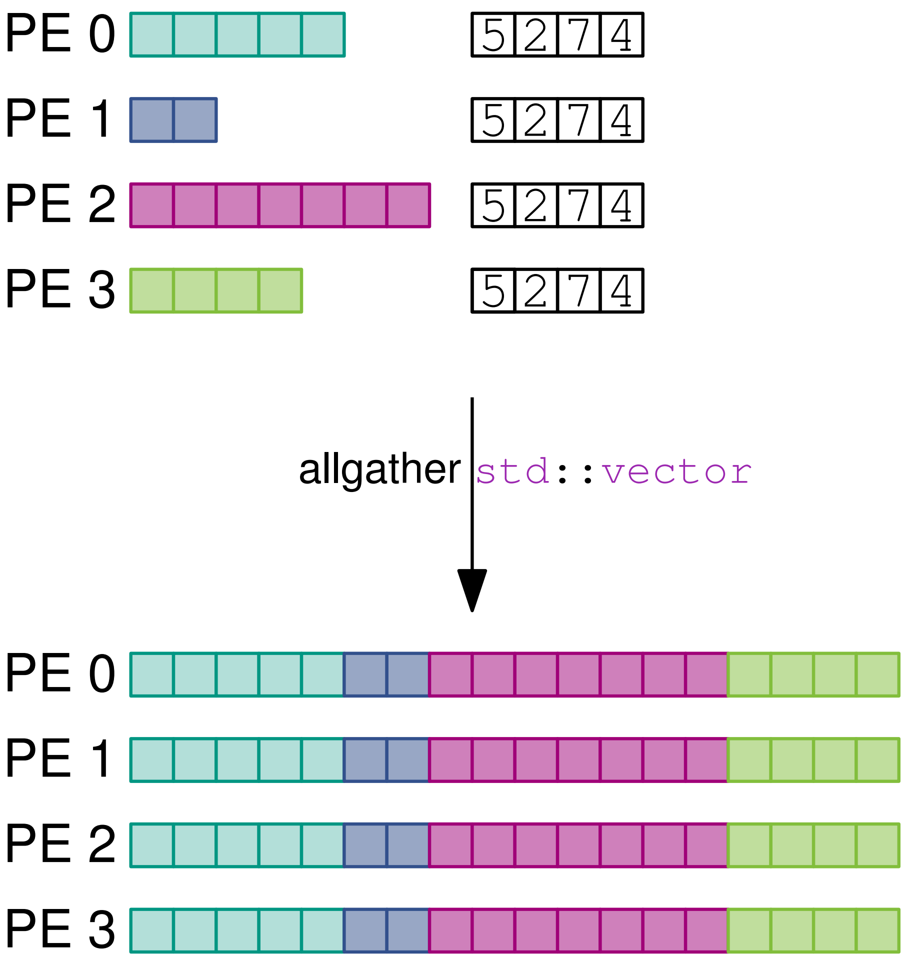
The baggage of using C++ + MPI



```
std::vector<double> get_whole_vector(std::vector<double> const& v_local,
                                     MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0,  MPI_DATATYPE_NULL
                rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

The Trail to HPC

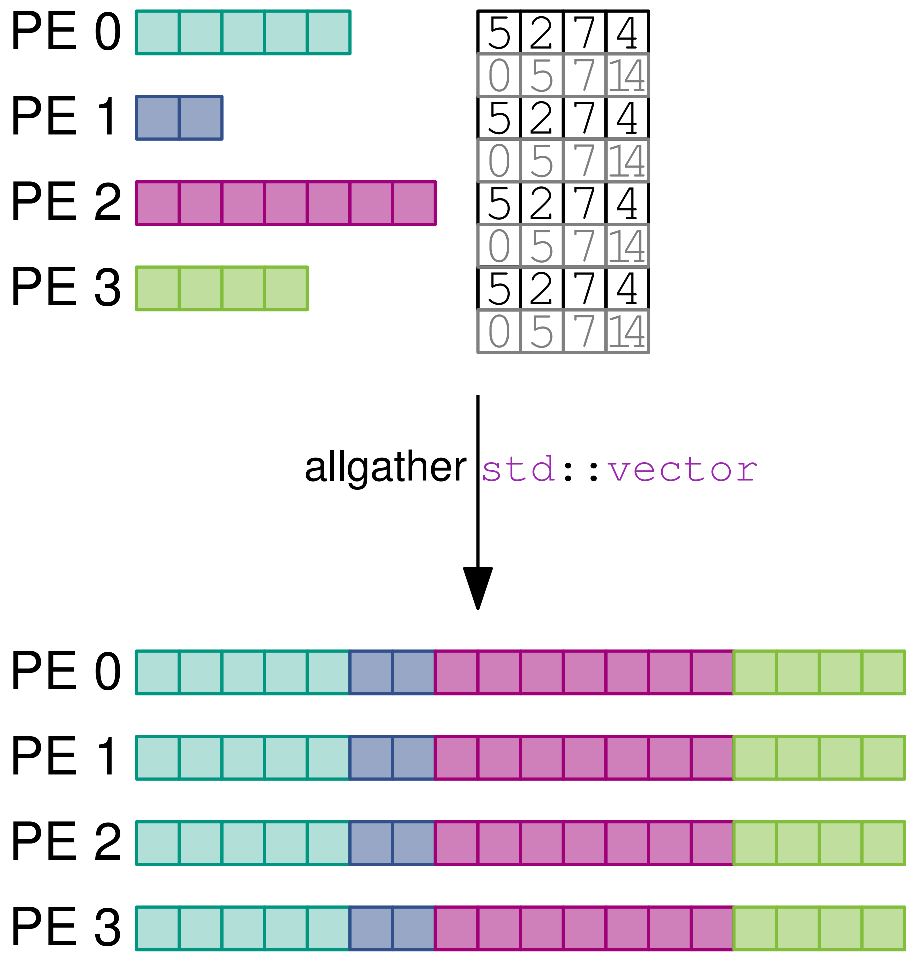
The baggage of using C++ + MPI



```
std::vector<double> get_whole_vector(std::vector<double> const& v_local,
                                     MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0,  MPI_DATATYPE_NULL
                rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

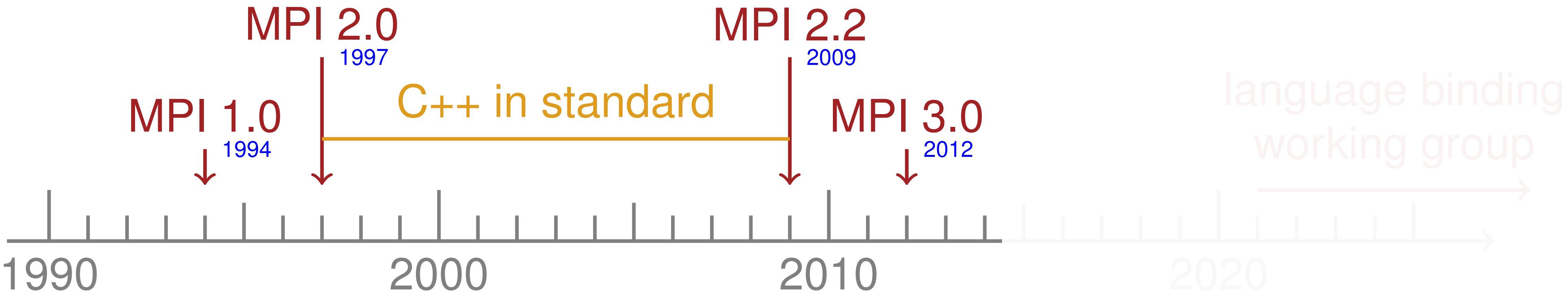
The Trail to HPC

The baggage of using C++ + MPI

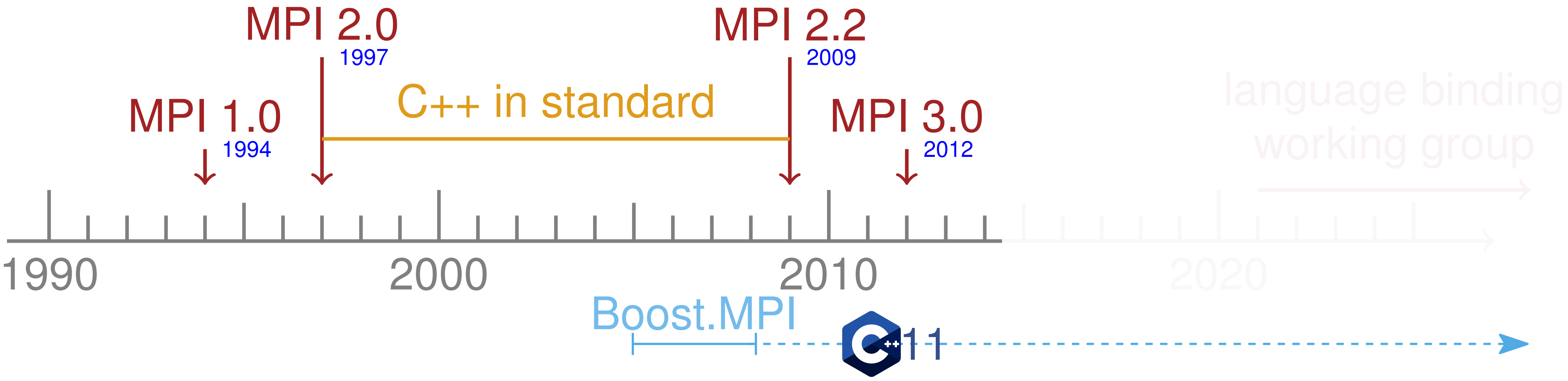


```
std::vector<double> get_whole_vector(std::vector<double> const& v_local,
                                     MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0,  MPI_DATATYPE_NULL
                rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

A Walk Through the History of MPI and C++


MPI 1.0
1994
MPI 2.0
1997
C++ in standard
MPI 2.2
2009
MPI 3.0
2012
language binding working group
1990
2000
2010
2020

A Walk Through the History of MPI and C++



MPI 1.0 1994

MPI 2.0 1997

C++ in standard

MPI 2.2 2009

MPI 3.0 2012

language binding working group

1990

2000

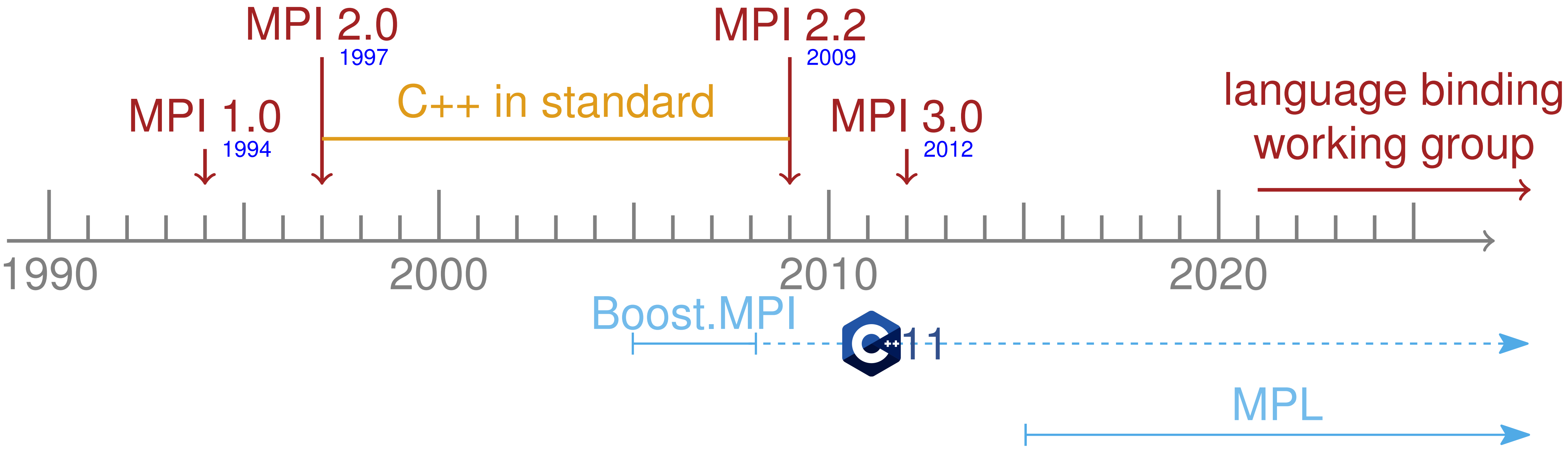
2010

2020

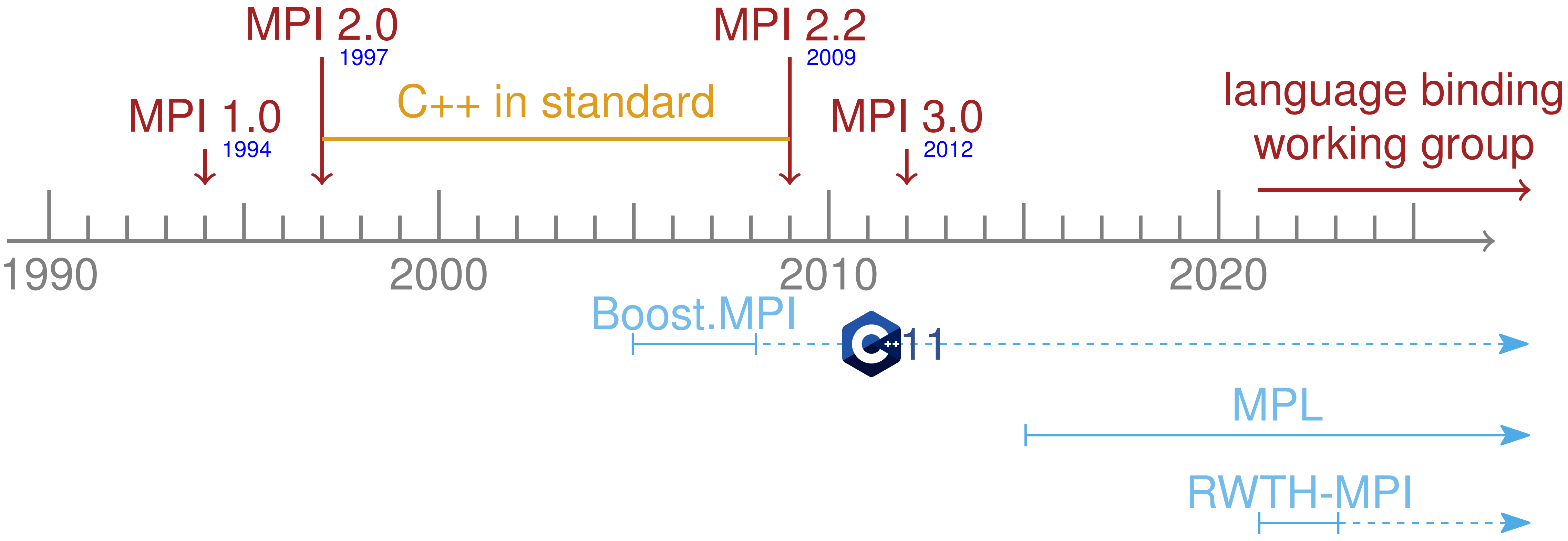
Boost.MPI

C++11

A Walk Through the History of MPI and C++



A Walk Through the History of MPI and C++



MPI 1.0 1994

MPI 2.0 1997

C++ in standard

MPI 2.2 2009

MPI 3.0 2012

language binding working group

1990

2000

2010

2020

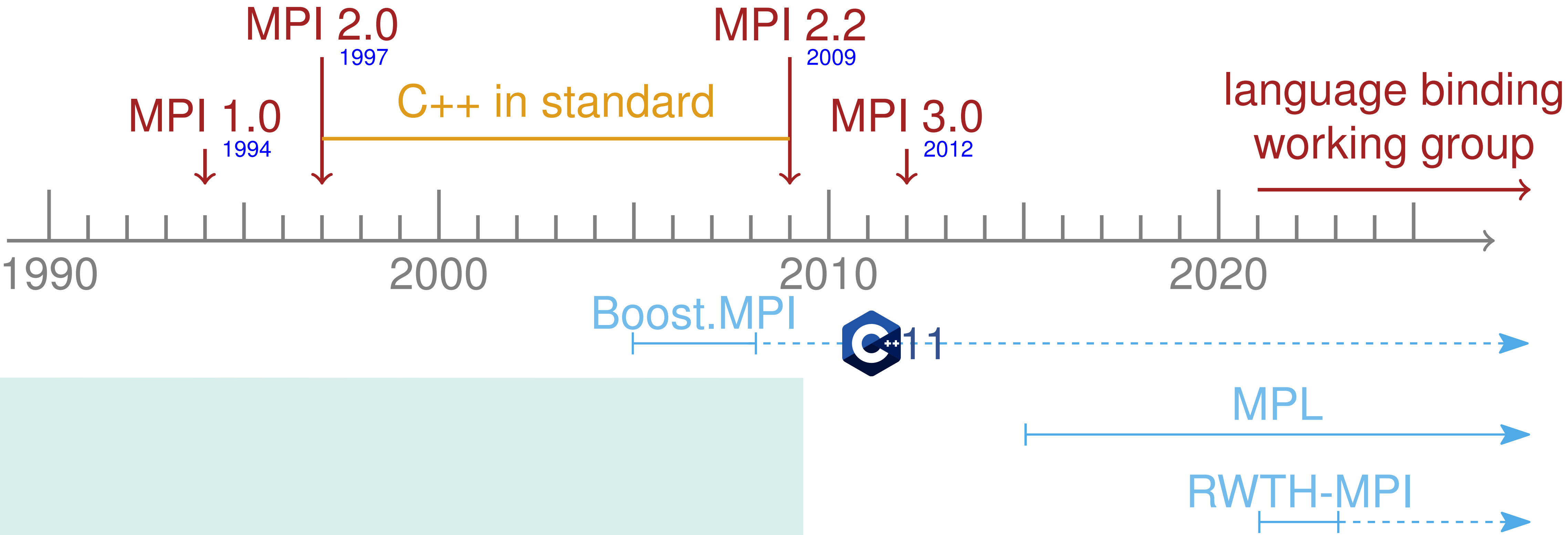
Boost.MPI

C++11

MPL

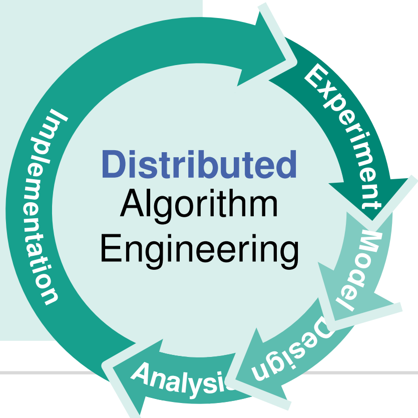
RWTH-MPI

A Walk Through the History of MPI and C++

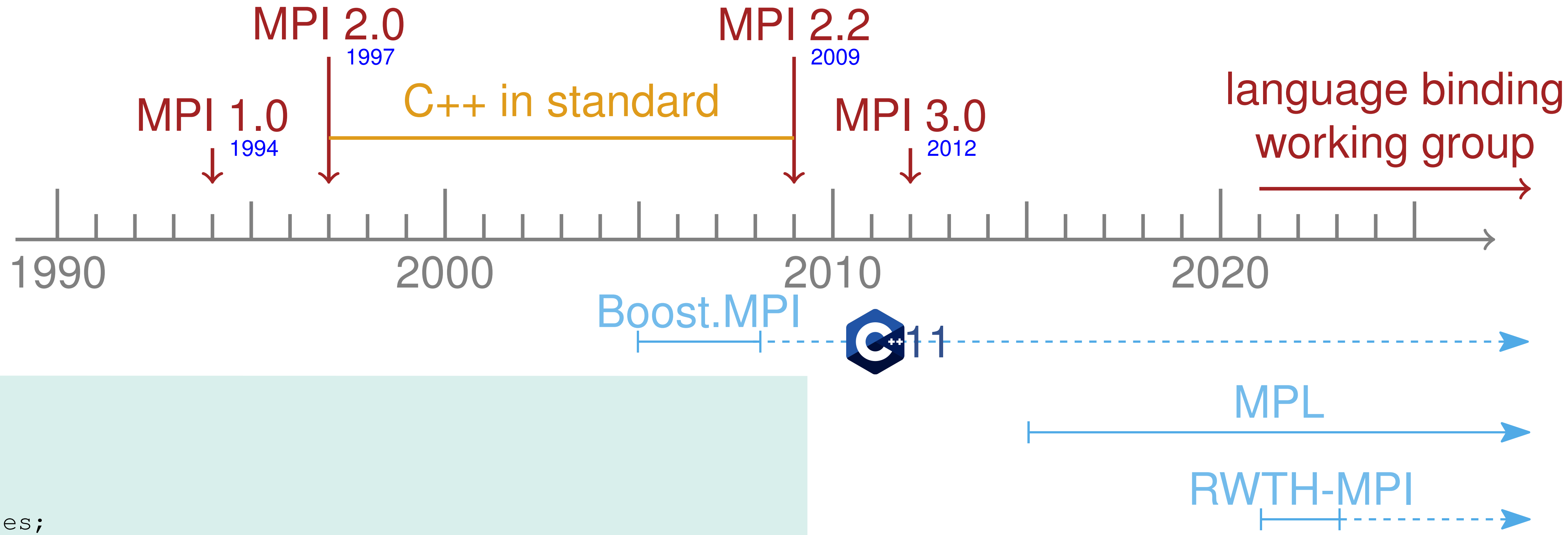


Example

```
// ...

boost::mpi::all_gatherv(comm, v_local, v_global);

// ...
for (auto& elem : v_global) {
  process(elem);
}
```

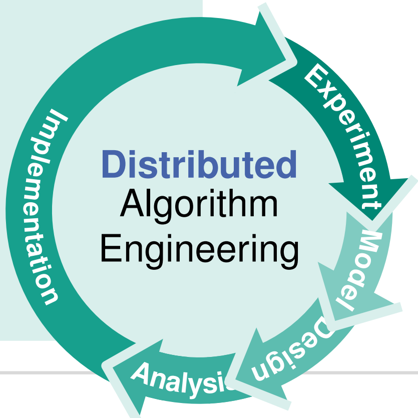


A Walk Through the History of MPI and C++

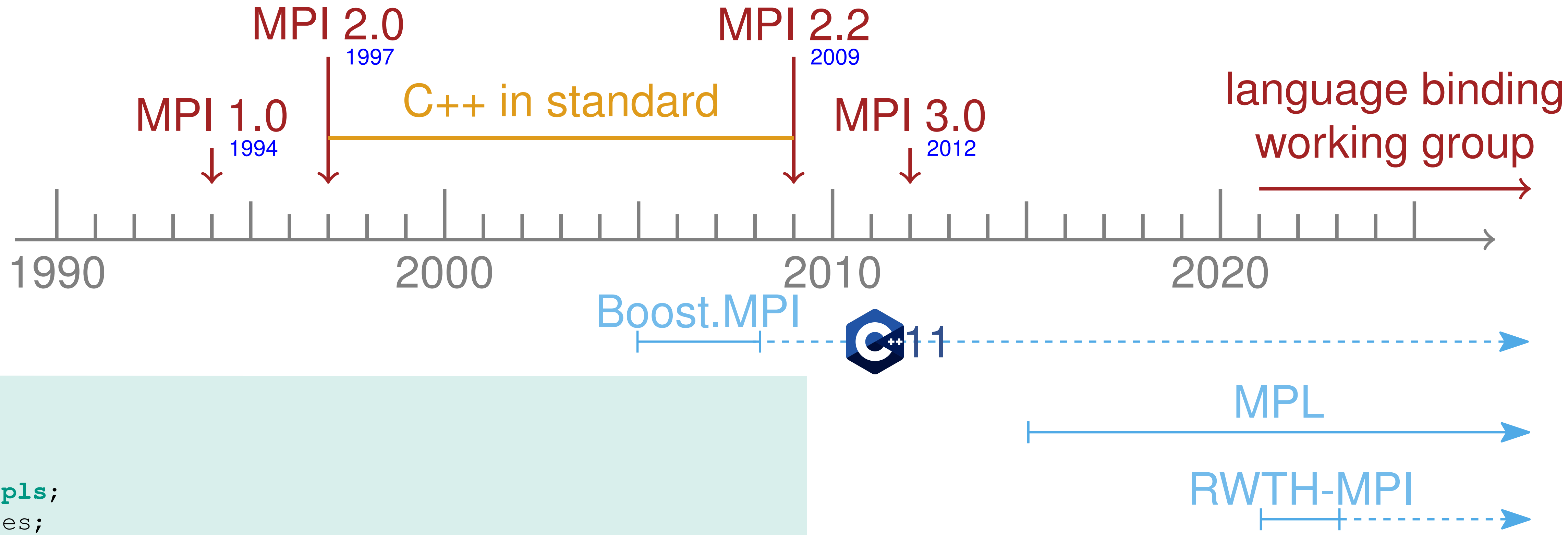


Example

```
// ...
std::vector<int> sizes;
boost::mpi::all_gather(comm, v_local.size(), sizes);
boost::mpi::all_gatherv(comm, v_local, v_global, sizes);
// ...
for (auto& elem : v_global) {
  process(elem);
}
```

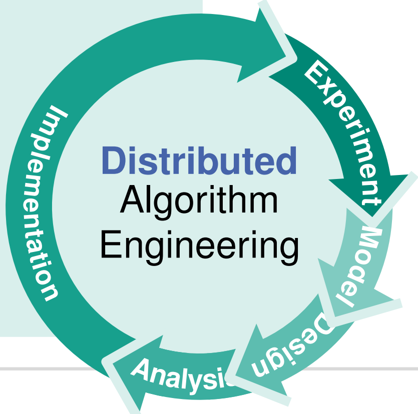


A Walk Through the History of MPI and C++

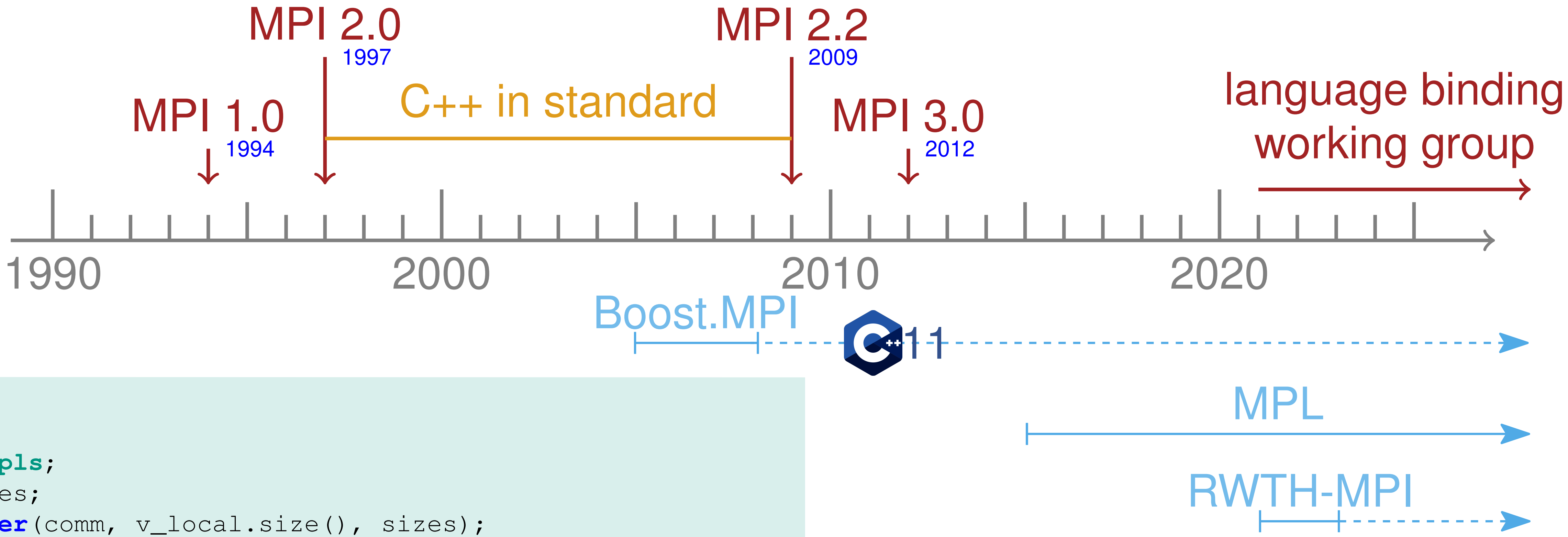


Example

```
std::vector<int> displs;
std::vector<int> sizes;
boost::mpi::all_gather(comm, v_local.size(), sizes);
boost::mpi::all_gatherv(comm, v_local, v_global, sizes);
// ...
for (auto& elem : v_global) {
  process(elem, heuristic(elem_rank));
}
```

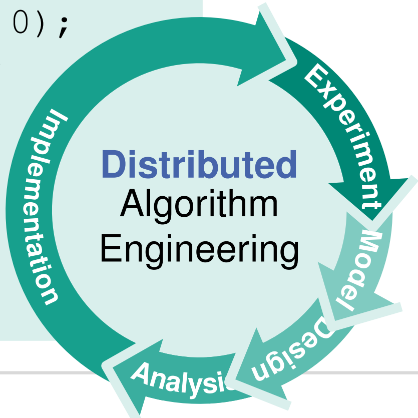


A Walk Through the History of MPI and C++

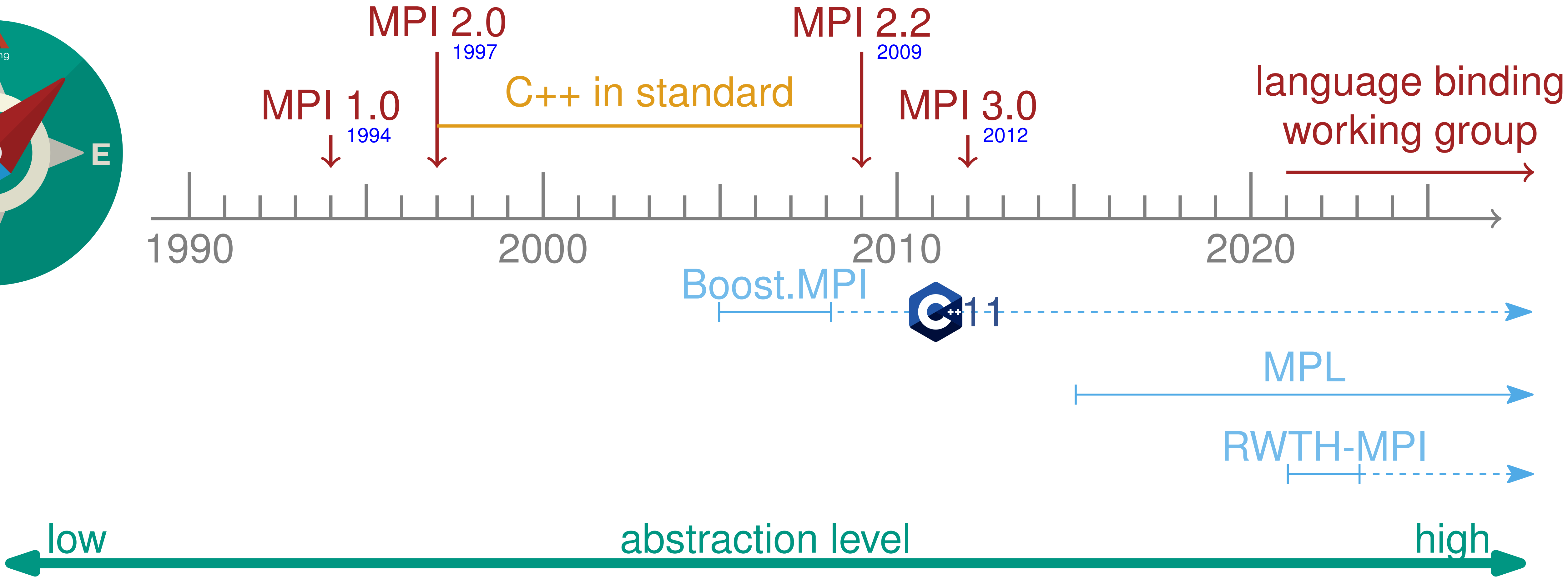


Example

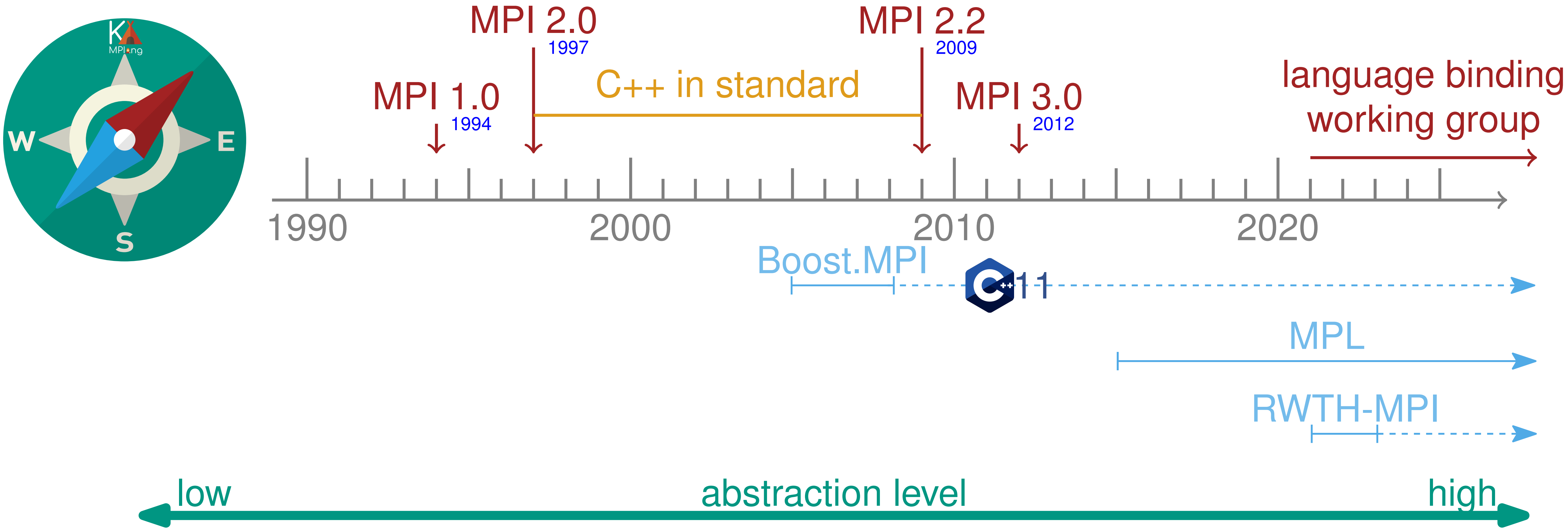
```
std::vector<int> displs;
std::vector<int> sizes;
boost::mpi::all_gather(comm, v_local.size(), sizes);
std::exclusive_scan(sizes.begin(), sizes.end(), displs.begin(), 0);
boost::mpi::all_gatherv(comm, v_local, v_global, sizes, displs);
// ...
for (auto& elem : v_global) {
  process(elem, heuristic(elem_rank));
}
```



A Walk Through the History of MPI and C++



A Walk Through the History of MPI and C++

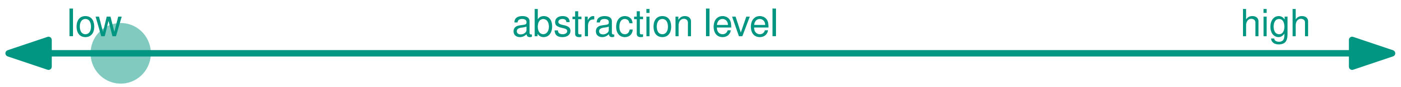


KaMPIng

Karlsruhe **MPI** **n**ext **g**eneration

A KaMPIng Trip

```
std::vector<double> get_whole_vector(std::vector<double> const& v_local, MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```



Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```cpp
std::vector<double> get_whole_vector(std::vector<double> const& v_local, MPI_Comm comm) {
  int size;
  int rank;
  MPI_Comm_size(comm, &size);
  MPI_Comm_rank(comm, &rank);
  std::vector<int> rc(size), rd(size);
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

C-ish API

all other parameters can be inferred

parameter order?

low
abstraction level
high

Goals:

□ zero-overhead **abstraction** over MPI

□ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms

□ flexible **parameter handling**, sensible defaults

□ configurable **memory management**

□ compatible with **move semantics**

A KaMPIng Trip

Karlsruhe Institute of Technology

```
std::vector<double> get_whole_vector(std::vector<double> const& v_local, MPI_Comm comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, rc.data(), 1, MPI_INT, comm);
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

all other parameters can be inferred

parameter order?

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

Karlsruhe Institute of Technology

```
std::vector<double> get_whole_vector(std::vector<double> const& v_local, MPI_Comm comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

all other parameters can be inferred

parameter order?

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
std::vector<double> get_whole_vector(std::vector<double> const& v_local, MPI_Comm comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<double> v_global(rd.back() + rc.back());
  MPI_Allgatherv(v_local.data(), v_local.size(), MPI_DOUBLE,
                 v_global.data(), rc.data(), rd.data(),
                 MPI_DOUBLE, comm);
  return v_global;
}
```

all other parameters can be inferred

parameter order?

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

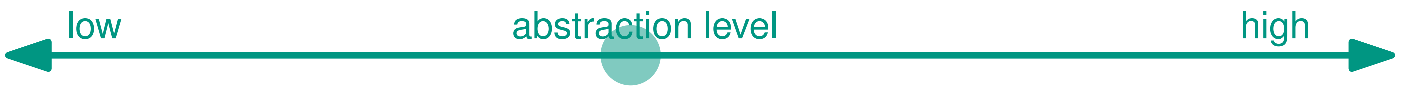
A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<T> v_global(rd.back() + rc.back());
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc), recv_displs(rd));

  return v_global;
}
```

all other parameters can be inferred

parameter order?

arbitrary parameter order!



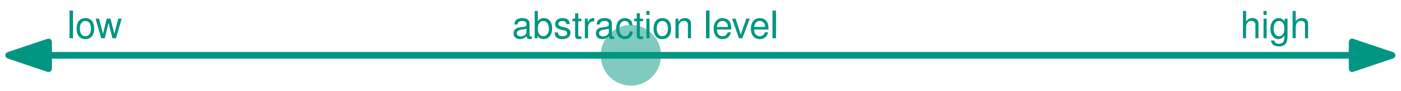
Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<T> v_global(rd.back() + rc.back());
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc), recv_displs(rd));

  return v_global;
}
```

manual allocation



Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<T> v_global;
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc), recv_displs(rd));

  return v_global;
}
```

automatic or manual allocation

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {


  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<T> v_global;
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc), recv_displs(rd));

  return v_global;
}
```

common idiom: boilerplate!

automatic or manual allocation

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  std::vector<int> rc(comm.size()), rd(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));
  std::exclusive_scan(rc.begin(), rc.end(), rd.begin(), 0);
  std::vector<T> v_global;
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc), recv_displs(rd));

  return v_global;
}
```

common idiom: boilerplate!

automatic or manual allocation

low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```cpp
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {

  std::vector<int> rc(comm.size());
  rc[rank] = v_local.size();
  comm.allgather(send_recv_buf(rc));

  std::vector<T> v_global;
  comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                  recv_counts(rc));

  return v_global;
}
```

common idiom: boilerplate!

automatic or manual allocation

low
abstraction level
high

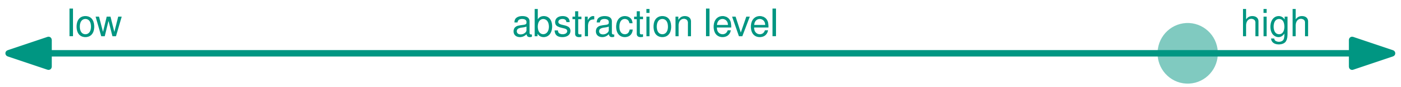
Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  std::vector<T> v_global;
  comm.allgatherv(send_buf(v_local), recv_buf(v_global));


  return v_global;
}
```

return by reference



Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {



  return comm.allgatherv(send_buf(v_local));


}
```

return by reference
or by value

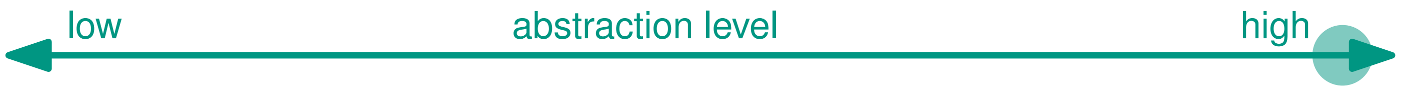
low
abstraction level
high

Goals:

- □ zero-overhead **abstraction** over MPI
- □ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {
  return comm.allgatherv(send_buf(v_local));
}
```



low

abstraction level

high

Goals:

- ☐ zero-overhead **abstraction** over MPI
- ☐ covering whole abstraction **range**: rapid prototyping ↔ highly engineered algorithms
- ☐ flexible **parameter handling**, sensible defaults
- ☐ configurable **memory management**
- ☐ compatible with **move semantics**

A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {
  return comm.allgatherv(send_buf(v_local));
}
```

```
// avoid implicit allocation
comm.allgatherv(send_buf(v_local),
                recv_counts_out<no_resize>(some_buf));

// pass buffer ownership to calls
rc = comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                     recv_counts_out<resize_to_fit>(std::move(rc)));

// retrieve auxiliary data
auto [recvbuf, displs] = comm.allgatherv(send_buf(v_local),
                                         recv_displs_out());
```

d **abstraction** over MPI

le abstraction **range**:
rapid prototyping ↔ highly engineered algorithms

☐ flexible **parameter handling**, sensible defaults

☐ configurable **memory management**

☐ compatible with **move semantics**

low
abstraction level
high

A KaMPIng Trip

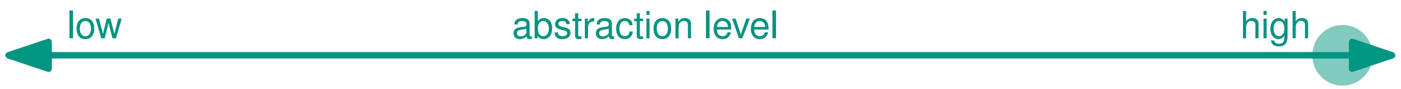
```cpp
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {
  return comm.allgatherv(send_buf(v_local));
}
```

```cpp
// avoid implicit allocation
comm.allgatherv(send_buf(v_local),
                recv_counts_out<no_resize>(some_buf));

// pass buffer ownership to calls
rc = comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                     recv_counts_out<resize_to_fit>(std::move(rc)));

// retrieve auxiliary data
auto [recvbuf, displs] = comm.allgatherv(send_buf(v_local),
                                         recv_displs_out());
```

d **abstraction** over MPI

le abstraction **range**:
rapid prototyping ↔ highly engineered algorithms

- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**



A KaMPIng Trip

```
template<typename T>
std::vector<T> get_whole_vector(std::vector<T> const& v_local, Communicator const& comm) {
  return comm.allgatherv(send_buf(v_local));
}
```

```
// avoid implicit allocation
comm.allgatherv(send_buf(v_local),
                recv_counts_out<no_resize>(some_buf));

// pass buffer ownership to calls
rc = comm.allgatherv(send_buf(v_local), recv_buf(v_global),
                     recv_counts_out<resize_to_fit>(std::move(rc)));

// retrieve auxiliary data
auto [recvbuf, displs] = comm.allgatherv(send_buf(v_local),
                                         recv_displs_out());
```

d **abstraction** over MPI

le abstraction **range**:
rapid prototyping ↔ highly engineered algorithms

- □ flexible **parameter handling**, sensible defaults
- □ configurable **memory management**
- □ compatible with **move semantics**

low
abstraction level
high

Equipped with More Features

Flexible Type System

- automatic type deduction
- type reflection
- opt-in serialization

```
using dict = std::unordered_map<std::string, std::string>;
dict data = ...;
comm.send(send_buf( kamping:: as_serialized(data)));

dict recv_dict = comm.recv(
  send_buf( kamping:: as_deserializable<dict>())
);
```

Equipped with More Features

Flexible Type System

- automatic type deduction
- type reflection
- opt-in serialization

```
using dict = std::unordered_map<std::string, std::string>;
dict data = ...;
comm.send(send_buf( kamping:: as_serialized(data)));

dict recv_dict = comm.recv(
  send_buf( kamping:: as_deserializable<dict>())
);
```

Safety Features

preventing programming errors for

- non-blocking communication
- inplace operations
- invalid arguments

```
std::vector<int> v = ...;
auto r1 = comm.isend(
    send_buf_out(std::move(v)), destination(1)
);

v = r1.wait(); // v  moved back after completion

auto r2 = comm.irecv<int>(recv_count(42));
// data returned after completion
std::optional<std::vector<int>> data = r2.test();
```

Equipped with More Features

Flexible Type System

- automatic type deduction
- type reflection
- opt-in serialization

```
using dict = std::unordered_map<std::string, std::string>;
dict data = ...;
comm.send(send_buf( kamping:: as_serialized(data)));

dict recv_dict = comm.recv(
  send_buf( kamping:: as_deserializable<dict>())
);
```

Safety Features

preventing programming errors for

- non-blocking communication
- inplace operations
- invalid arguments

```
std::vector<int> v = ...;
auto r1 = comm.isend(
    send_buf_out(std::move(v)), destination(1)
);

v = r1.wait(); // v  moved back after completion

auto r2 = comm.irecv<int>(recv_count(42));
// data returned after completion
std::optional<std::vector<int>> data = r2.test();
```

Extensibility

Plugins for

- specialized collectives
- fault tolerance
- STL-style algorithms

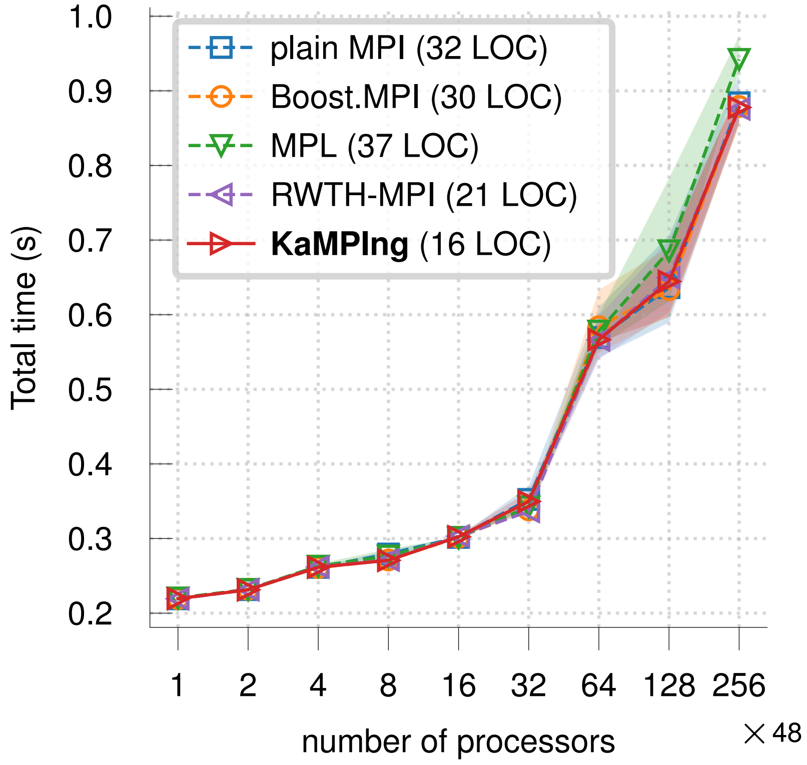
KaMPIng Out in the Wild

Artifacts Available V1.1 acm · Artifacts Evaluated Functional V1.1 acm · Results Replicated V1.1 acm

Karlsruhe Institute of Technology

Sorting

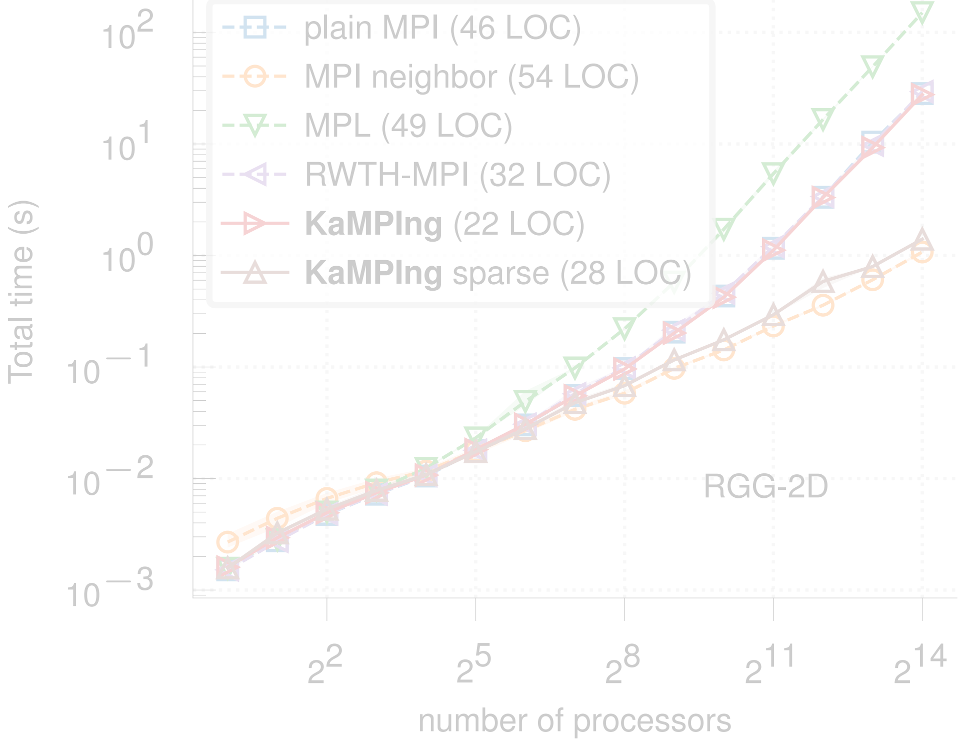
Sample Sort:



Suffix Sorting: < 200 LOC

Graph Algorithms

BFS:



Graph Partitioning: 15% less code

Bioinformatics

Phylogenetic Inference:

```
   .--.
  / \  \
 /\  \  \
/ /\ /\ |\
R Ax ML ng
```

- over 50 000 citations
- over 700 lines of custom MPI wrapper

compile time 1:15min + 0:15min

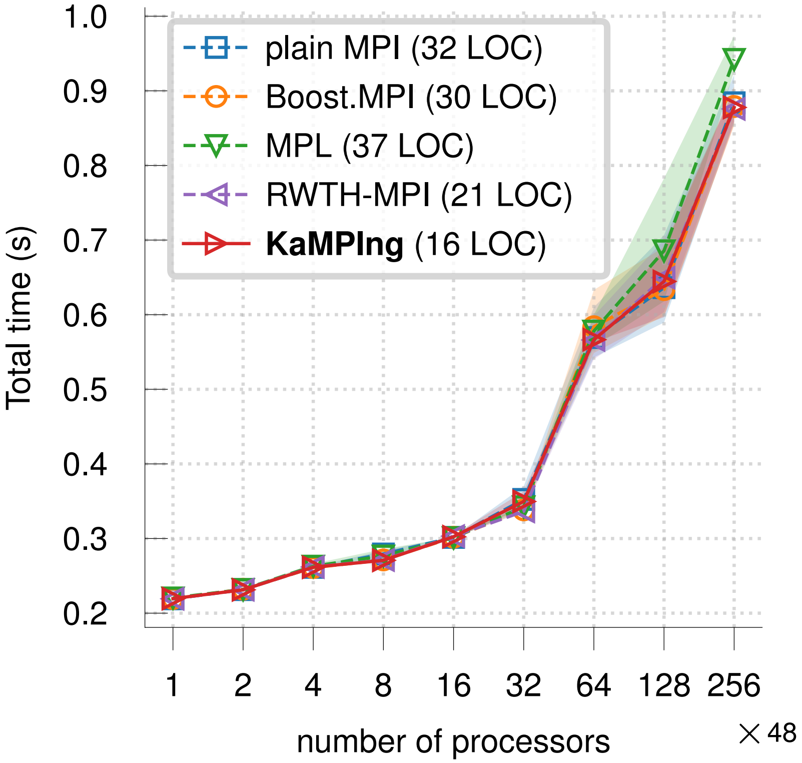
binary size +2.5%

KaMPIng Out in the Wild

Artifacts Available V1.1 acm
Artifacts Evaluated Functional V1.1 acm
Results Replicated V1.1 acm

Sorting

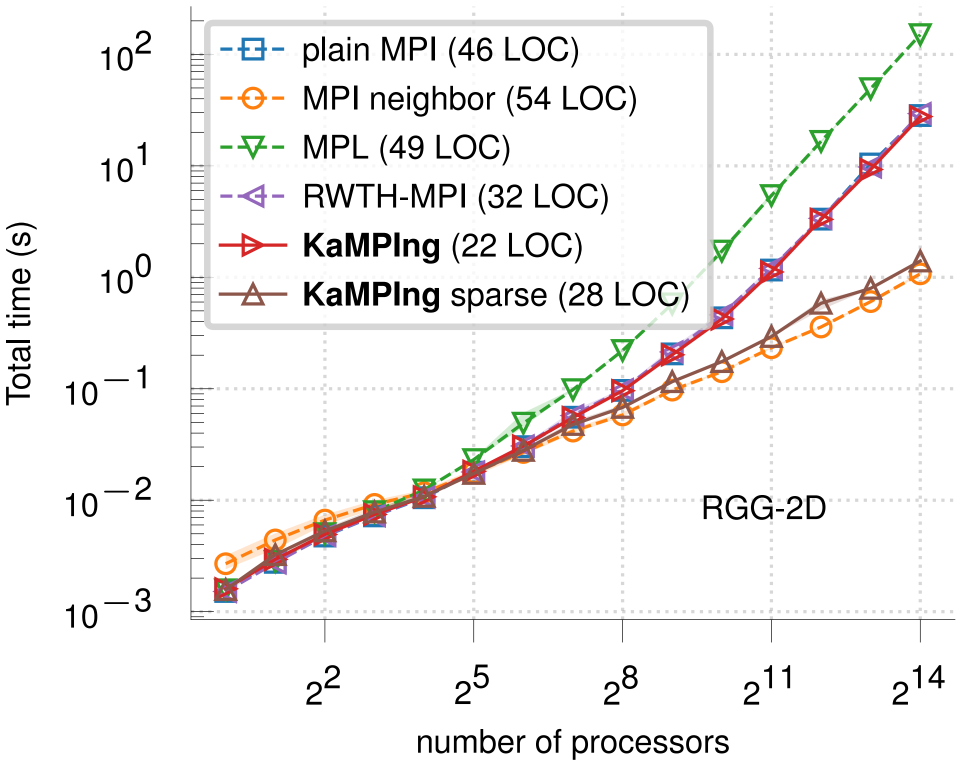
Sample Sort:



Suffix Sorting: < 200 LOC

Graph Algorithms

BFS:



Graph Partitioning: 15% less code

Bioinformatics

Phylogenetic Inference:

```
   .--.
  / \  \
 /\  \  \
/ /\ /\ |\
R Ax ML ng
```

- over 50 000 citations
- over 700 lines of custom MPI wrapper

compile time 1:15min + 0:15min

binary size +2.5%

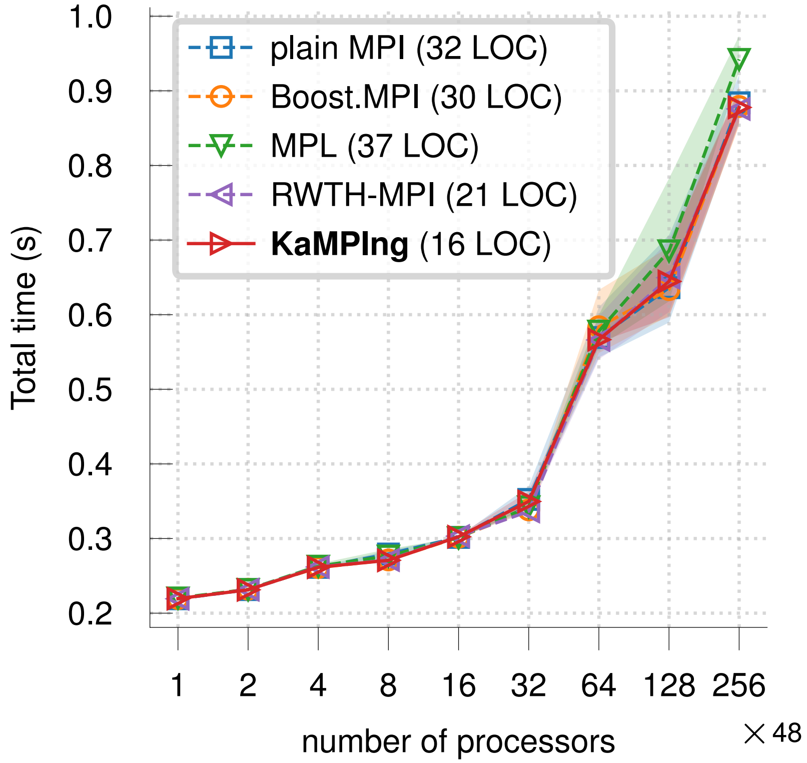
KaMPIng Out in the Wild

Artifacts Available acm V1.1 · Artifacts Evaluated Functional V1.1 acm · Results Replicated acm V1.1

Karlsruhe Institute of Technology

Sorting

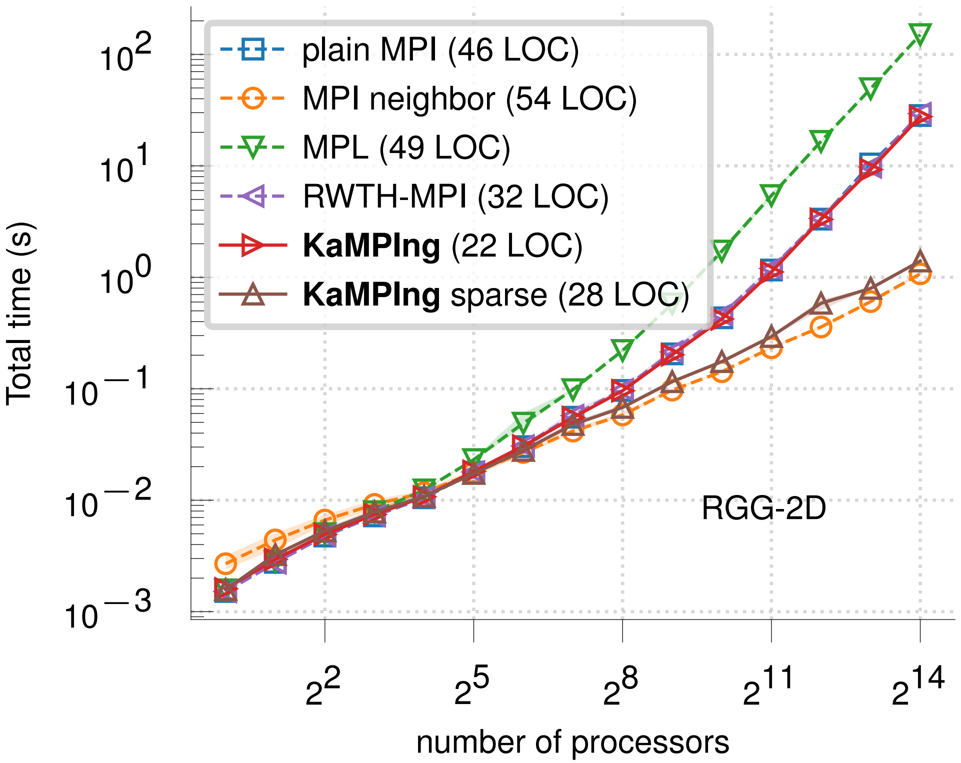
Sample Sort:



Suffix Sorting: < 200 LOC

Graph Algorithms

BFS:



Graph Partitioning: 15% less code

Bioinformatics

Phylogenetic Inference:

```
   .--.
  / \  \
 /\  \  \
/ /\ /\ |\
R Ax ML ng
```

- over 50 000 citations
- over 700 lines of custom MPI wrapper

compile time 1:15min + 0:15min

binary size +2.5%

Join the KAmp Today!

```
template<typename T>
static void mpi_broadcast(T& obj) {
  if (_num_ranks > 1) {
    size_t size = master() ?
      BinaryStream::serialize(
        _parallel_buf.data(),
        _parallel_buf.capacity(),
        obj)
      : 0;
    mpi_broadcast((void *) &size, sizeof(size_t));
    mpi_broadcast((void *) _parallel_buf.data(), size);
    if (!master()) {
      BinaryStream bs(_parallel_buf.data(), size);
      bs >> obj;
    }
  }
}
```

original RAxML-NG code

```
template <typename T>
static void mpi_broadcast(T &obj) {
  if (_num_ranks > 1) {
    _comm->bcast(send_recv_buf( as_serialized(obj)));
  }
}
```

Get started!

CMake

```
FetchContent_Declare(
  kamping
  GIT_REPOSITORY https://github.com/kamping-site/kamping.g
  GIT_TAG v0.1.1
)

FetchContent_MakeAvailable(kamping)

target_link_libraries(myapp PRIVATE kamping::kamping)
```

C++

```
#include <kamping/communicator.hpp>
#include <kamping/collectives/bcast.hpp>

kamping::Communicator comm(my_comm);
comm.bcast(...);
```

github.com/kamping-site/kamping

Join the KAmp Today!

```
template<typename T>
static void mpi_broadcast(T& obj) {
  if (_num_ranks > 1) {
    size_t size = master() ?
      BinaryStream::serialize(
        _parallel_buf.data(),
        _parallel_buf.capacity(),
        obj)
      : 0;
    mpi_broadcast((void *) &size, sizeof(size_t));
    mpi_broadcast((void *) _parallel_buf.data(), size);
    if (!master()) {
      BinaryStream bs(_parallel_buf.data(), size);
      bs >> obj;
    }
  }
}
```

original RAxML-NG code

```
template <typename T>
static void mpi_broadcast(T &obj) {
  if (_num_ranks > 1) {
    _comm->bcast(send_recv_buf( as_serialized(obj)));
  }
}
```

Get started!

CMake

```
FetchContent_Declare(
  kamping
  GIT_REPOSITORY https://github.com/kamping-site/kamping.g
  GIT_TAG v0.1.1
)

FetchContent_MakeAvailable(kamping)

target_link_libraries(myapp PRIVATE kamping::kamping)
```

C++

```
#include <kamping/communicator.hpp>
#include <kamping/collectives/bcast.hpp>

kamping::Communicator comm(my_comm);
comm.bcast(...);
```

github.com/kamping-site/kamping

Join the KAmp Today!

Karlsruhe Institute of Technology

```
template<typename T>
static void mpi_broadcast(T& obj) {
  if (_num_ranks > 1) {
    size_t size = master() ?
      BinaryStream::serialize(
        _parallel_buf.data(),
        _parallel_buf.capacity(),
        obj)
      : 0;
    mpi_broadcast((void *) &size, sizeof(size_t));
    mpi_broadcast((void *) _parallel_buf.data(), size);
    if (!master()) {
      BinaryStream bs(_parallel_buf.data(), size);
      bs >> obj;
    }
  }
}
```

original RAxML-NG code

```
template <typename T>
static void mpi_broadcast(T &obj) {
  if (_num_ranks > 1) {
    _comm->bcast(send_recv_buf( as_serialized(obj)));
  }
}
```

MPIng

Get started!

CMake

```
FetchContent_Declare(
  kamping
  GIT_REPOSITORY https://github.com/kamping-site/kamping.g
  GIT_TAG v0.1.1
)

FetchContent_MakeAvailable(kamping)

target_link_libraries(myapp PRIVATE kamping::kamping)
```

C++

```
#include <kamping/communicator.hpp>
#include <kamping/collectives/bcast.hpp>

kamping::Communicator comm(my_comm);
comm.bcast(...);
```

github.com/kamping-site/kamping

Packing Up: The Journey Ahead

- **low-to-high-level** C++ bindings for MPI
- no **runtime-overhead**
- reduce boilerplate and error-proneness in MPI applications
  - default parameters
  - safety guarantess
  - fine-grained memory management
- base for a future **standard library** of distributed algorithms and data structures

github.com/kamping-site/kamping

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500).