inventors for the digital world

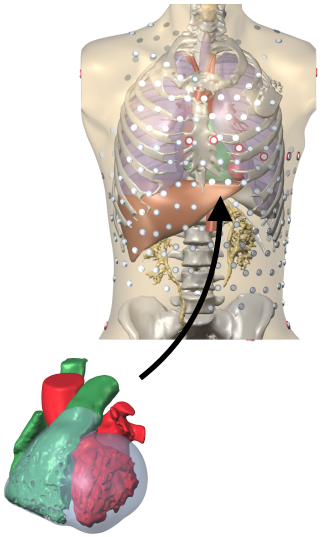
# Static/Dynamic Analysis for Parallel Applications Verification

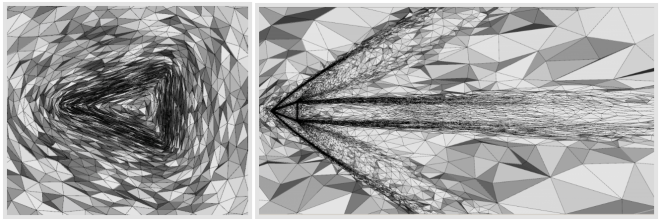
Emmanuelle Saillard, Inria STORM team

**Journée du groupe de travail LAMHA**

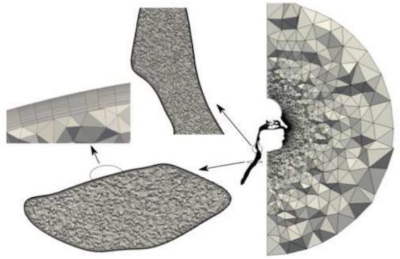
# High Performance Computing (HPC)

Propag5 (Inria)

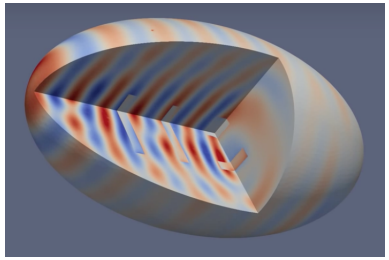




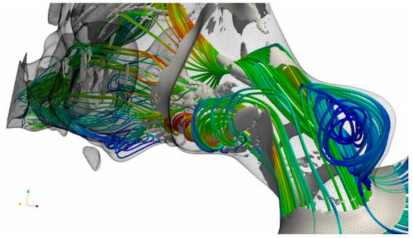
RealfluidsDS (Inria)



Meraculous (LBNL)

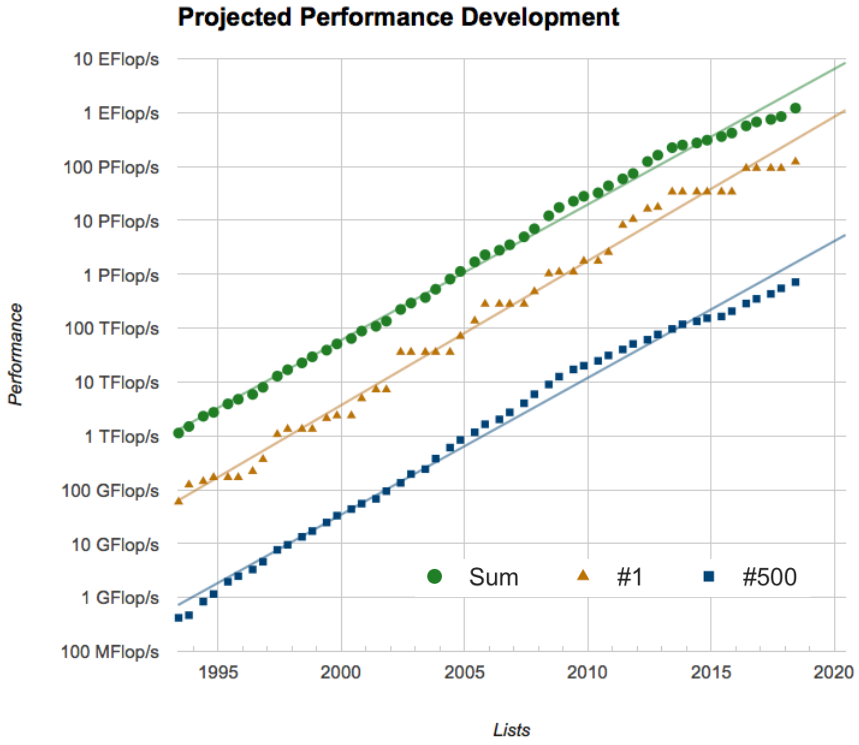


b



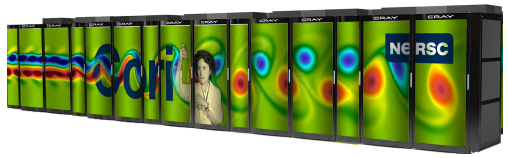
ALYA (BSC)

# High Performance Computing (HPC)




(http://www.top500.org)

- More **complex applications** (i.e., combination of parallel programming models)
- More **complex machines**, heterogeneous architectures
- **Exascale** systems targeted in 2020
- Cori supercomputer
  - #10
  - 622 336 cores
  - 14,014 PFlop/s



# Problem Statement

**How can we help developers having correct HPC applications ?**

# Problem Statement

**How can we help developers having correct HPC applications ?**

**What is a correct application ?**

- ➡ Satisfy the specification
- ➡ Compute what it is supposed to compute

# Problem Statement

**How can we help developers having correct HPC applications ?**

**What is a correct application ?**

- ➡ Satisfy the specification
- ➡ Compute what it is supposed to compute

**Too many bug-types - What to focus on ?**

# Problem Statement

**How can we help developers having correct HPC applications ?**

**What is a correct application ?**

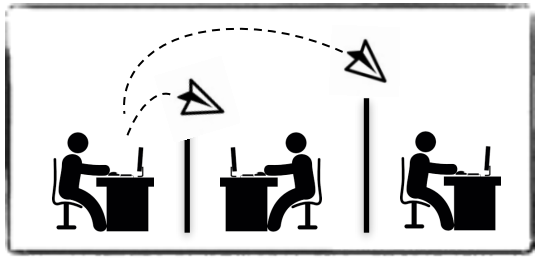
- Satisfy the specification
- Compute what it is supposed to compute

**Too many bug-types - What to focus on ?**

**Tools ?**

- Easy to use (« single pushed button »)
- Scalability
- Precision (locate source of errors)
- Usuability (what can be verified?)
- Detection of errors as soon as possible
- Heterogeneity (new class of errors)

# Parallel Programming Models



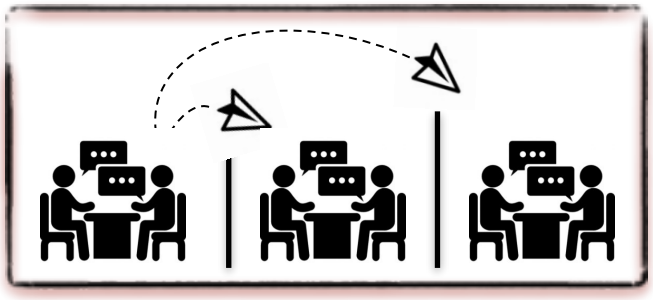
**Distributed Memory approach**

(e.g., MPI)

**Shared Memory approach**

(e.g., Pthread, OpenMP)

**PGAS** (e.g., UPC)

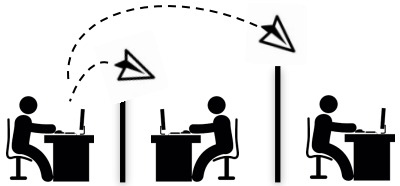


**Hybrid approach**

(e.g., MPI+X with , a shared memory model)

# Collective

## MPI



Blocking or non-blocking communication involving all MPI processes of the same communicator

`MPI_Barrier, MPI_Ibarrier, MPI_Bcast,…`

## OpenMP

Barrier and worksharing construct

`#pragma omp barrier/single/ for/sections/workshare`

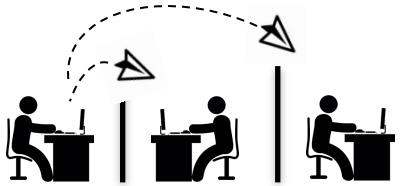
## UPC

Collective communication operations

`upc_barrier, upc_wait, upc_all_broadcast,…`

# Collective Errors

## MPI



Blocking or non-blocking communication involving all MPI processes of the same communicator

`MPI_Barrier, MPI_Ibarrier, MPI_Bcast,…`

## OpenMP

Barrier and worksharing construct

`#pragma omp barrier/single/ for/sections/workshare`

## UPC

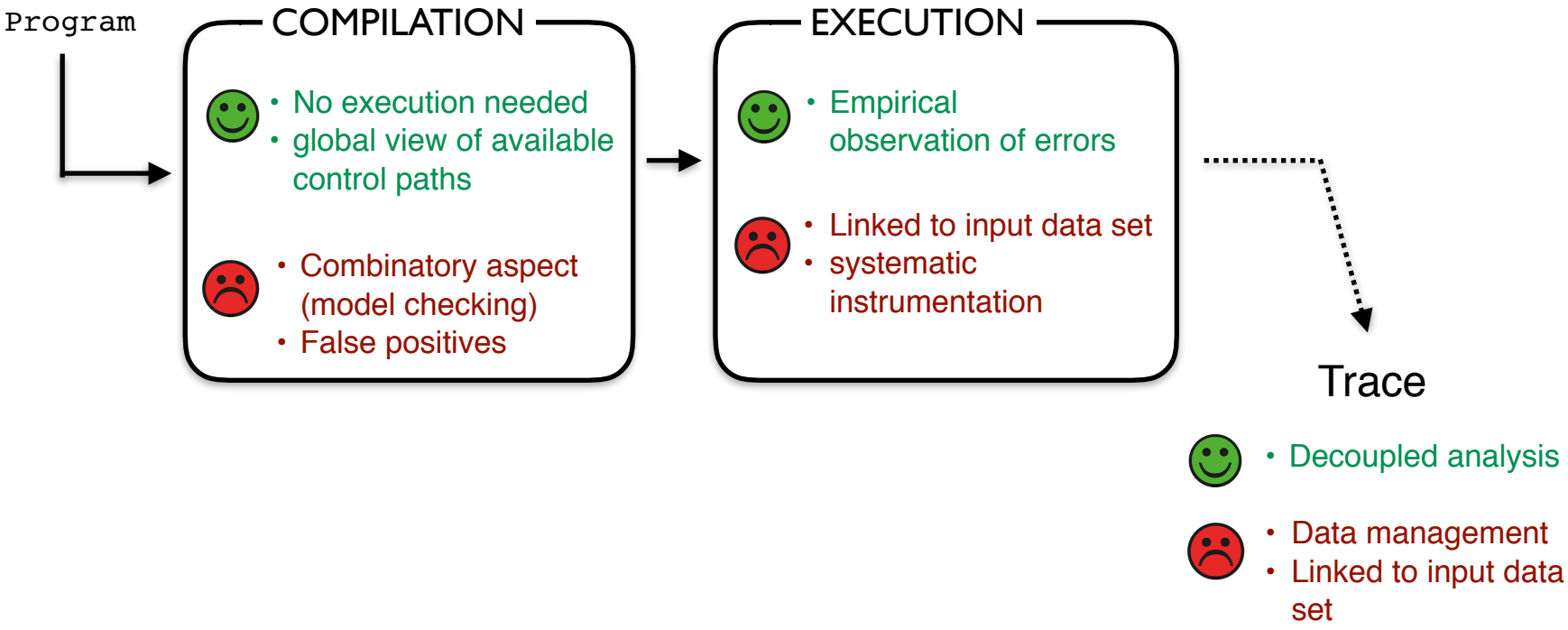
Collective communication operations

`upc_barrier, upc_wait, upc_all_broadcast,…`

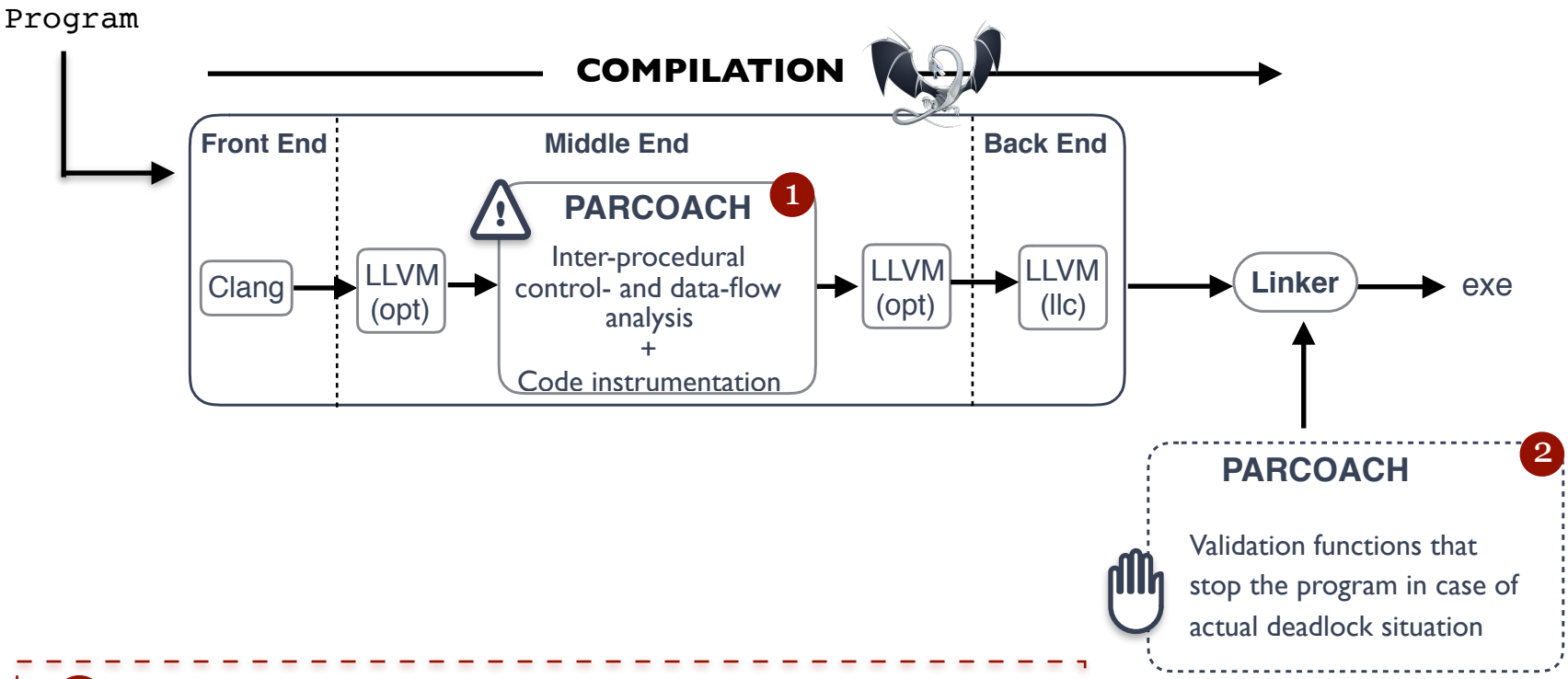
**Specification** : All processes / threads must have the same sequence of collectives

**Goal** : Detect collective errors (i.e., collective mismatch)

# Existing Methods

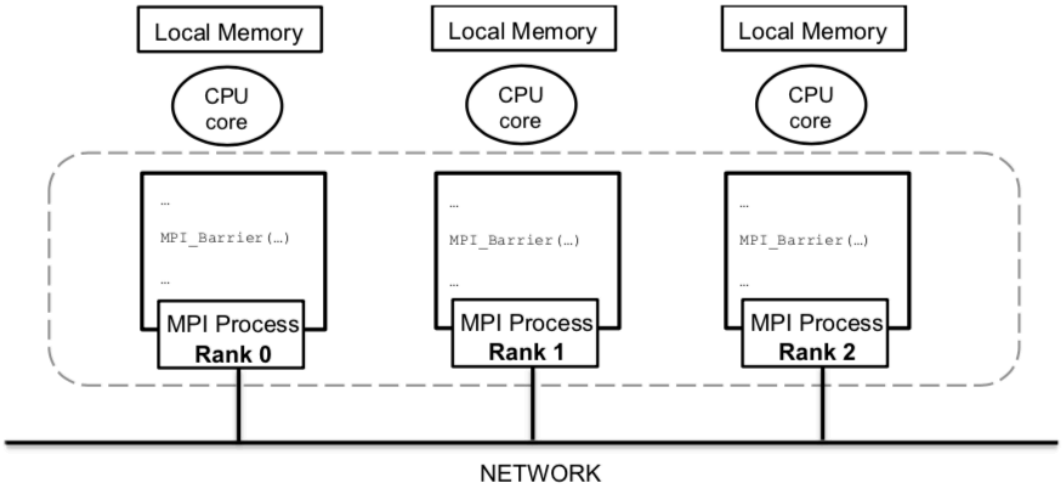


# PARallel COntrol flow Anomaly CHecker

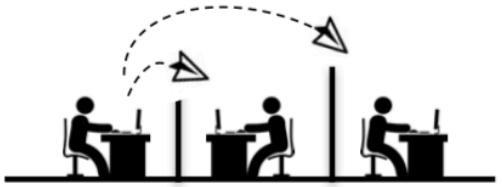



1. **Detection of potential deadlocks at compile-time**
2. **Verification of potential deadlocks at execution**

# Motivating Example




**Point-to-point communication**



**Collective communication**
**(MPI_Barrier, MPI_Reduce, …)**

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

**Deadlock**

**The machine state does not help to detect the source of the deadlock**

# Motivating Example

```
void c(int s) {

  if(s>256)
      MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

**Deadlock**

**The machine state does not help to detect the source of the deadlock**

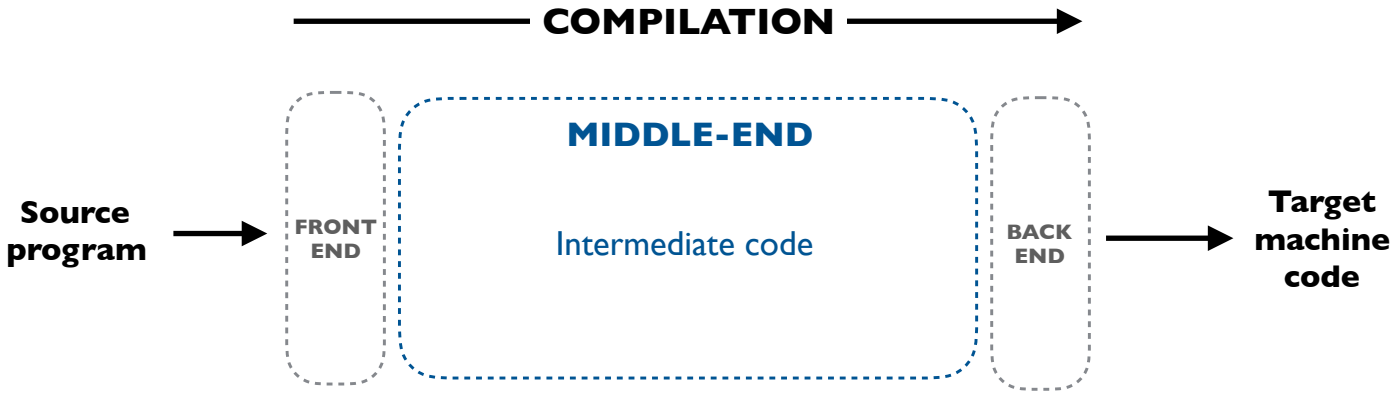
# Motivating Example

```
void c(int s) {

  if(s>256)
     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/
  MPI_Finalize();
}
```

**Deadlock**

**The machine state does not help to detect the source of the deadlock**

## Structure of a compiler




## Intermediate representation

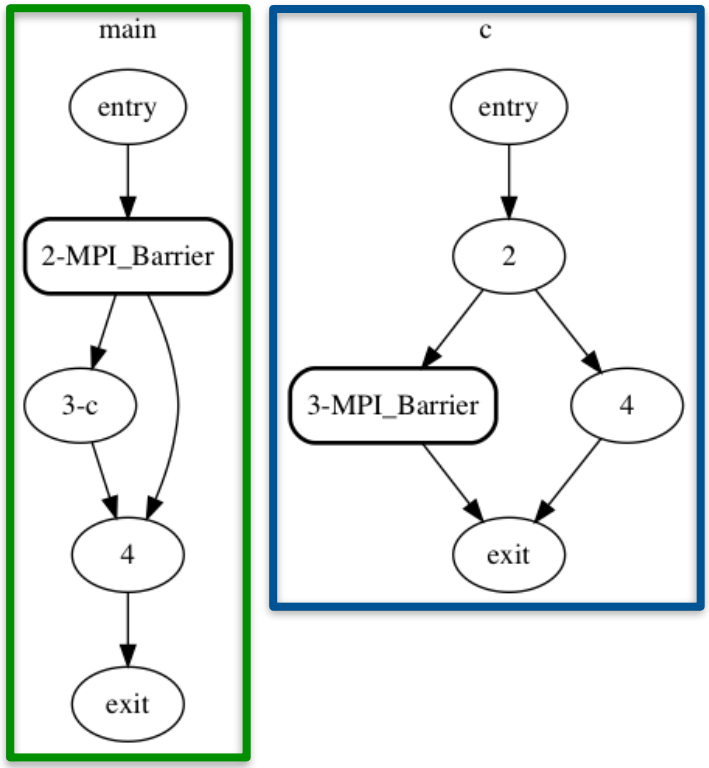
- Programming language independent
- Target machine independent

## Control Flow Graph (CFG)

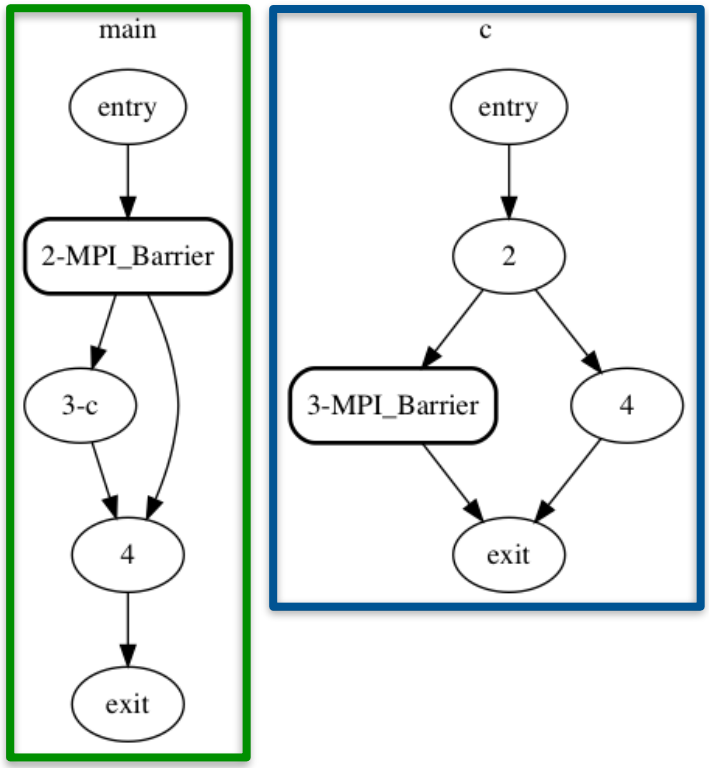
- Models all program executions
- Right representation to study instruction flow

# Control Flow Graph

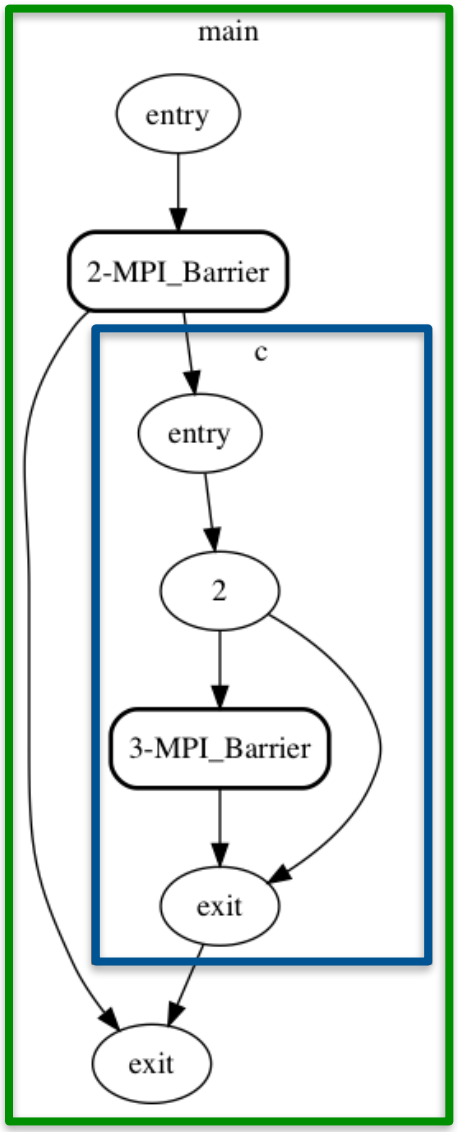
```
void c( ) {

  if( )                          2
     MPI_Barrier(com2);          3
  else
    /*…*/                        4
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);             2

  if( )
    c( );                        3
  /*…*/
  MPI_Finalize();                4
}
```

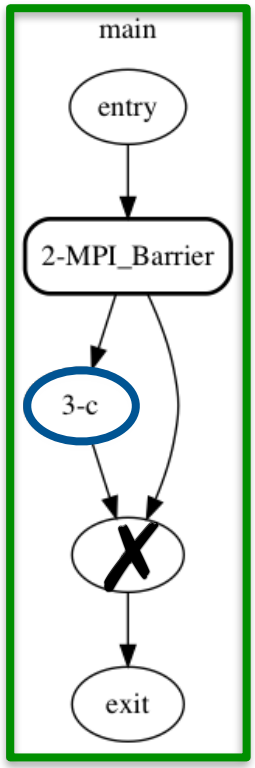


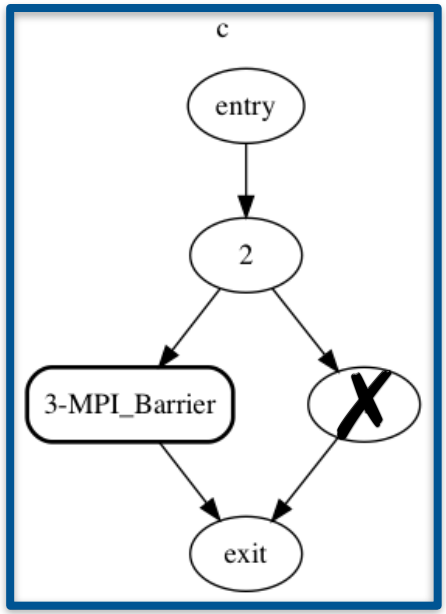
# Parallel Program Control Flow Graph



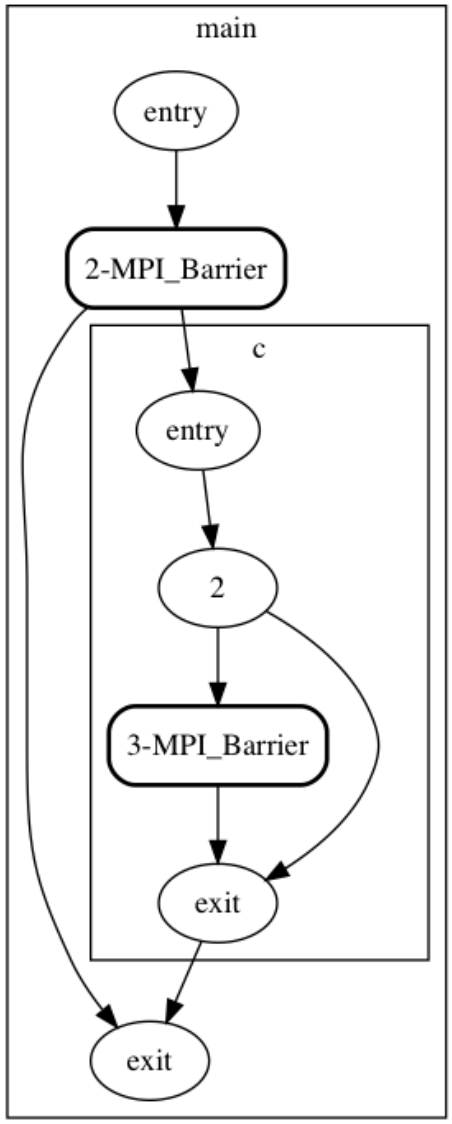
# Parallel Program Control Flow Graph



# Collective Error Detection



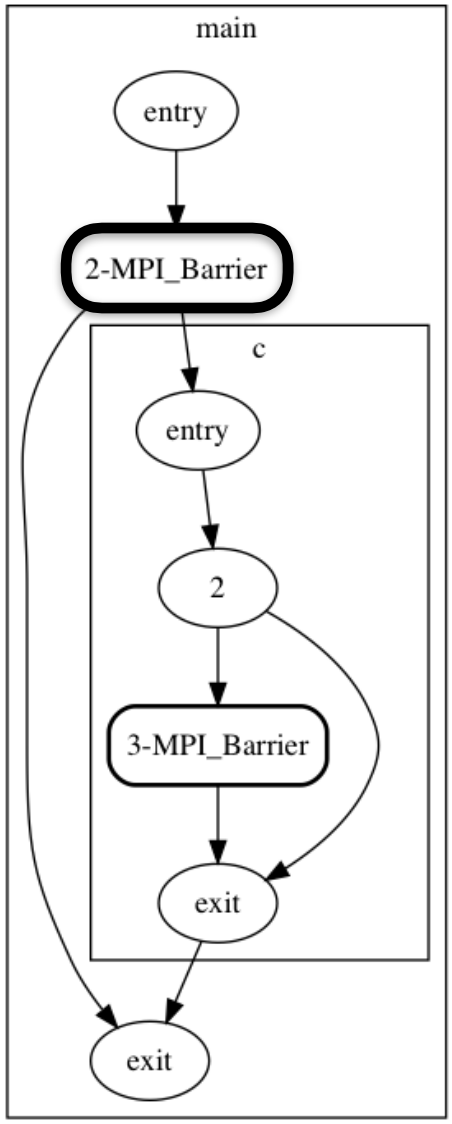

## Iterated post dominance frontier (PDF+)

**U** : Set of nodes in PPCFG

**PDF+(U)** : Set of nodes that can lead both to a node in U or not

**Find control-flow divergence that may lead to different sequences of collectives**

# Collective Error Detection



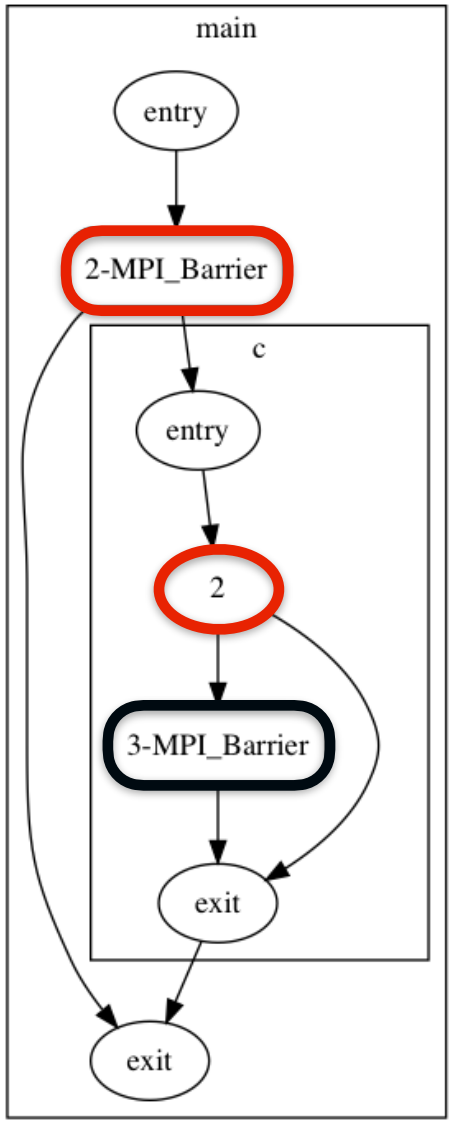

## Iterated post dominance frontier ($PDF^+$)

**U** : Set of nodes in PPCFG

**$PDF^+(U)$** : Set of nodes that can lead both to a node in U or not

$$PDF^+(\{2_{main}\}) = \varnothing$$

# Collective Error Detection




## Iterated post dominance frontier (PDF+)

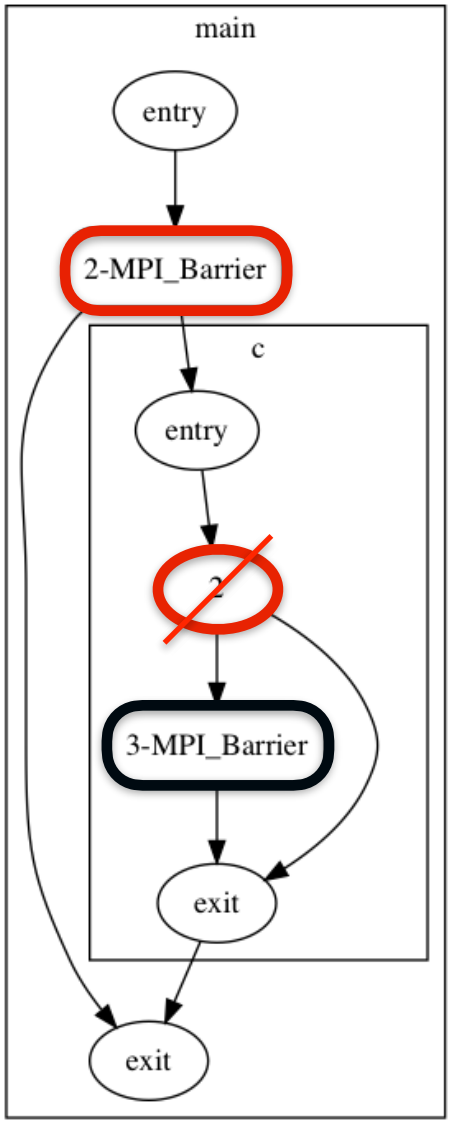
**U** : Set of nodes in PPCFG

**PDF+(U)** : Set of nodes that can lead both to a node in U or not

$PDF^+(\{2_{main}\}) = \varnothing$

$PDF^+(\{3_c\}) = \{2_c, 2_{main}\}$

# Collective Error Detection




## Iterated post dominance frontier (PDF+)

**U** : Set of nodes in PPCFG

**PDF+(U)** : Set of nodes that can lead both to a node in U or not

$PDF^+(\{2_{main}\}) = \varnothing$

$PDF^+(\{3_c\}) = \{2_c, 2_{main}\}$

**Filter out conditionals on rank-independent expression**

# Feedback to users

```
1  void c(int s) {
2
3    if(s>256)
4      MPI_Barrier(com2);
5    else
6      /*…*/
7  }
8
9  int main( ) {
10
11   /*…*/
12   MPI_Barrier(com1);
13
14   if(r%2)
15     c(s);
16   /*…*/
17   MPI_Finalize();
18 }
```

**What a user can read on stderr**

```
PARCOACH: warning: MPI_Barrier line 4
possibly not called by all processes
because of conditional(s) line(s) 14
```

# Code Instrumentation

```
void c(int s) {

  if(s>256)

     MPI_Barrier(com2);
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);

  if(r%2)
    c(s);
  /*…*/


  MPI_Finalize();
}
```

- **No warning = no instrumentation**

# Code Instrumentation

```
void c(int s) {

  if(s>256)
     CC(MPI,com2,i_barrier,O)
     MPI_Barrier(com2)✗
  else
    /*…*/
}

int main( ) {

  /*…*/
  MPI_Barrier(com1);✓

  if(r%2)
    c(s);
  /*…*/
  CC(MPI,com2,0,∅)
  MPI_Finalize();
}
```

- **No warning = no instrumentation**

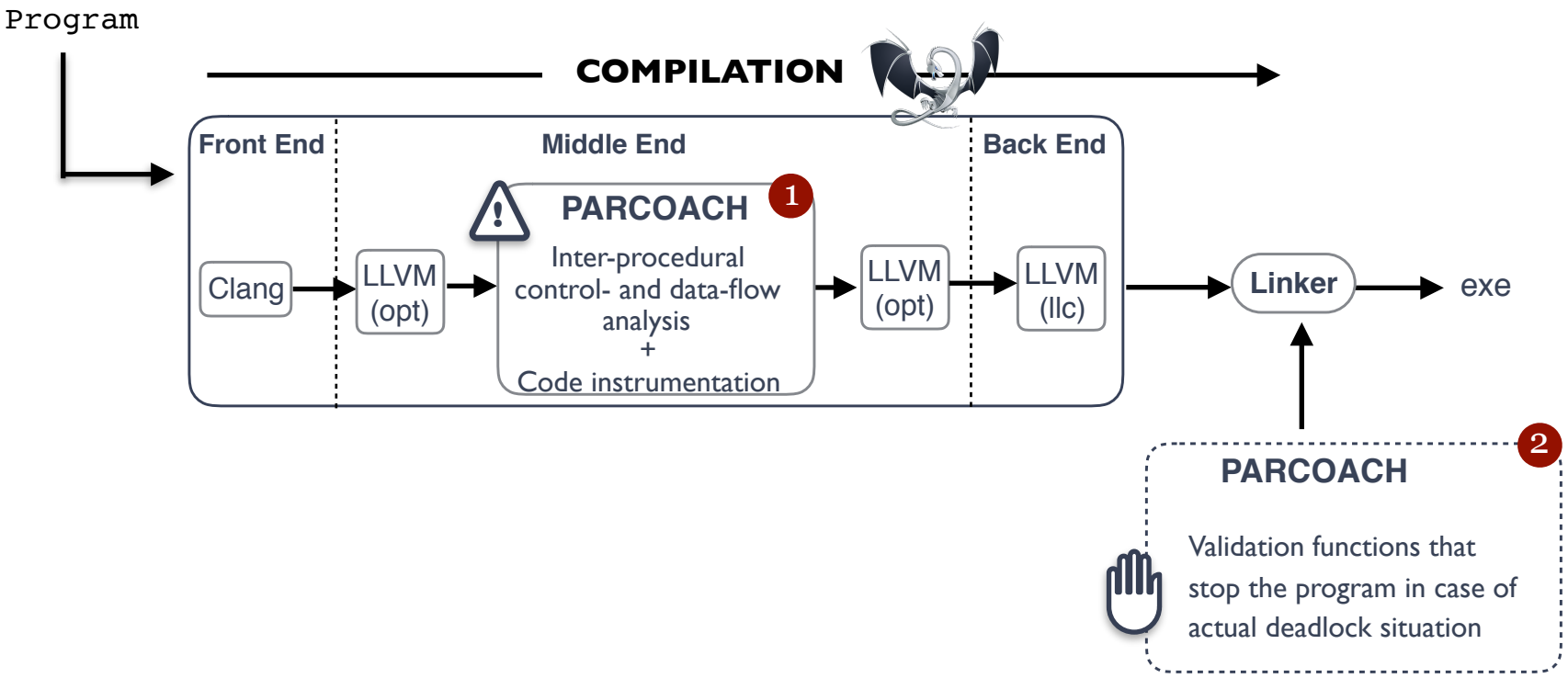
- If at least one warning:
  - > Insert a Check Collective (CC) function **CC($i_{model}$,$com_c$,$i_c$,$O$)** before each collective c, **starting with the 1st collective that may deadlock**
  - > Insert **CC($i_{model}$,com,0,∅)** before exit statements and some MPI functions (`MPI_Abort, MPI_Finalize`)

  - ✳ $i_{model}$ = Parallel programming model used
  - ✳ $com_c$ = communicator related to c (0 for OpenMP)
  - ✳ $i_c$ = collective ID
  - ✳ $O$ = collectives that may deadlock (set generated at compile-time)

# Conclusion

**How can we help developers having correct HPC applications ?**

## PARallel COntrol flow Anomaly CHecker (PARCOACH)



# Conclusion

**How can we help developers having correct HPC applications ?**

**PARallel COntrol flow Anomaly CHecker (PARCOACH)**

**Tools ?**

- Easy to use (« single pushed button »)
- Scalability
- Precision (locate source of errors)
- Usuability (what can be verified?)
- Detection of errors as soon as possible
- Heterogeneity (new class of errors)

# Thank you!

Follow us on www.inria.fr

inventors for the digital world