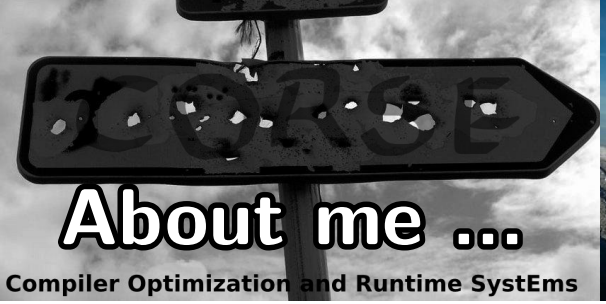



# Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget
Jean-François Méhaut, Miguel Santana

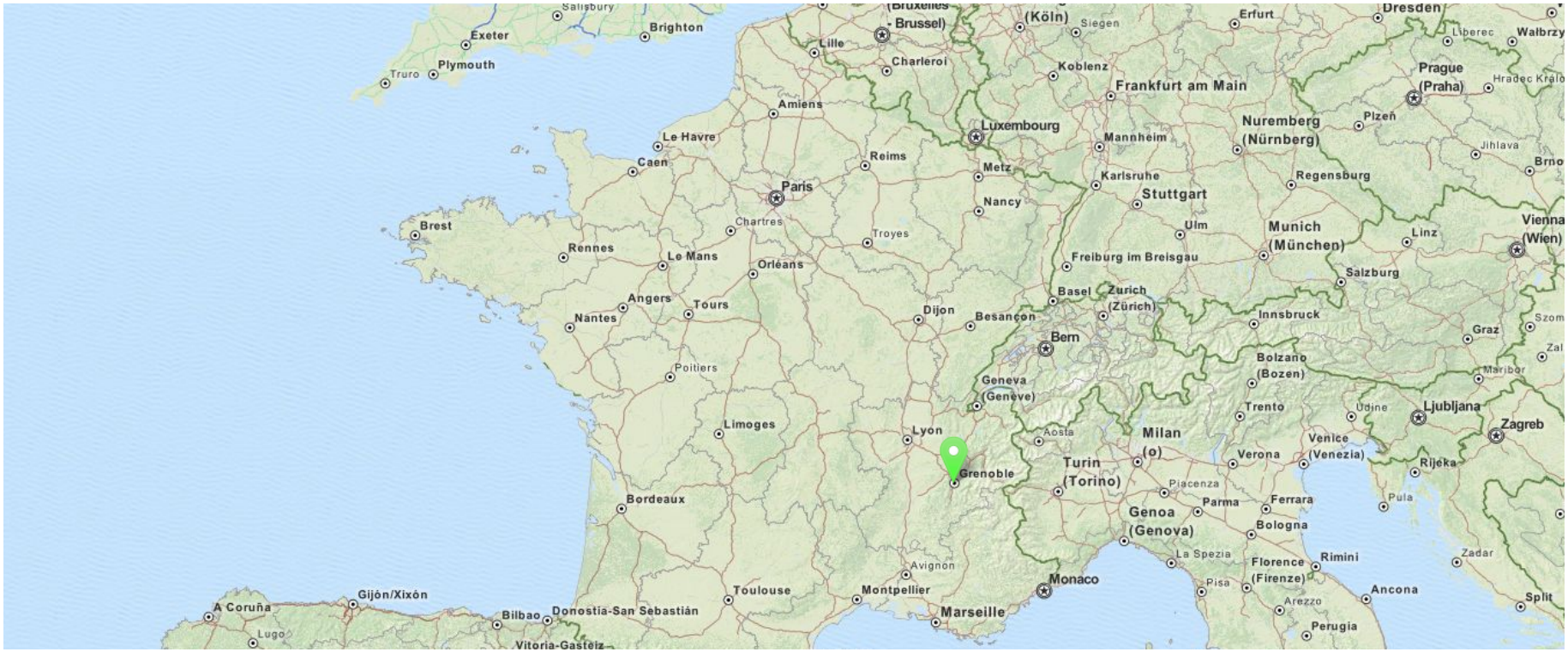
University Joseph Fourier / LIG, STMicroelectronics, Grenoble, France
Nano2017-DEMA project

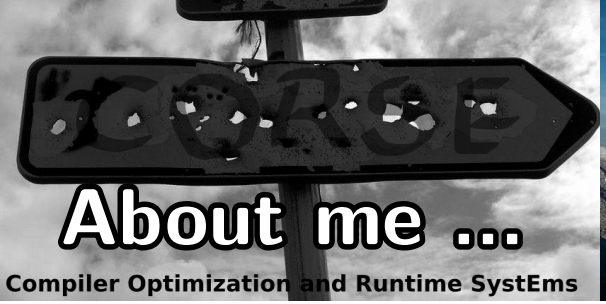
HLRS Institute, Stuttgart, Germany
August 31$^{st}$, 2015




# About me ...

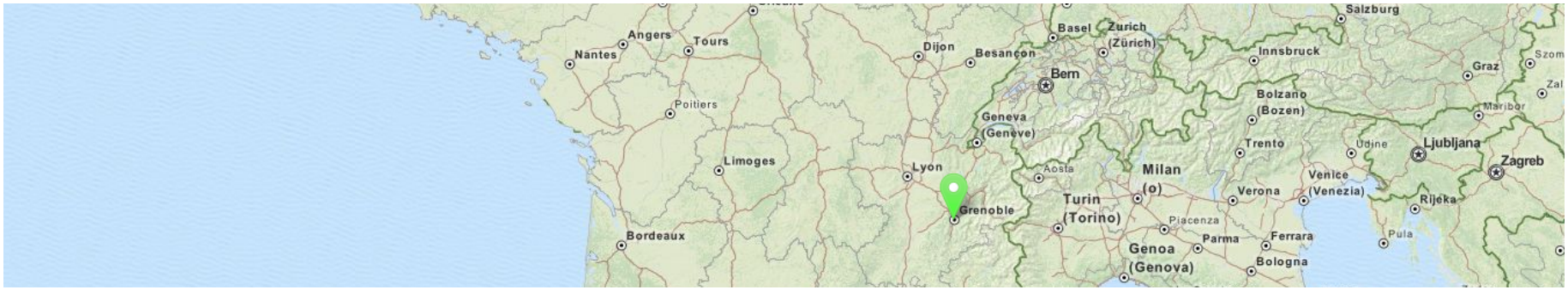
**Compiler Optimization and Runtime SystEms**

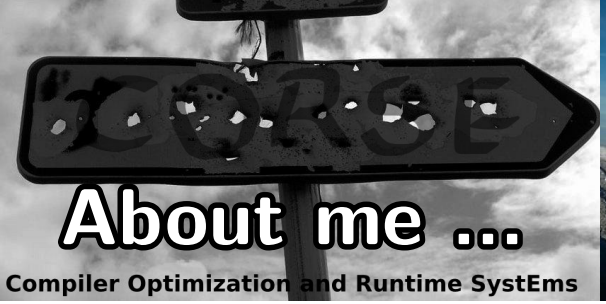




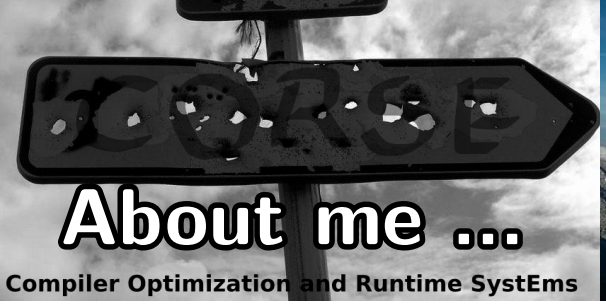

# About me ...

Compiler Optimization and Runtime SystEms



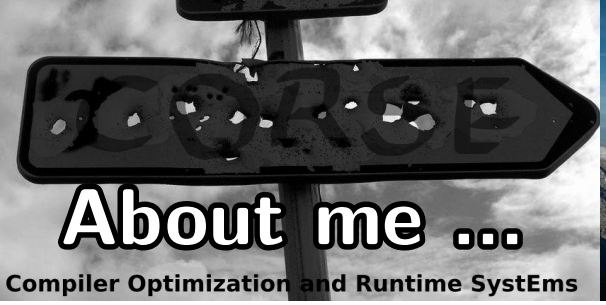

- PhD in 2014 from the University of Grenoble (UJF)
  - in partnership with STMicroelectronics




# About me ...

Compiler Optimization and Runtime SystEms

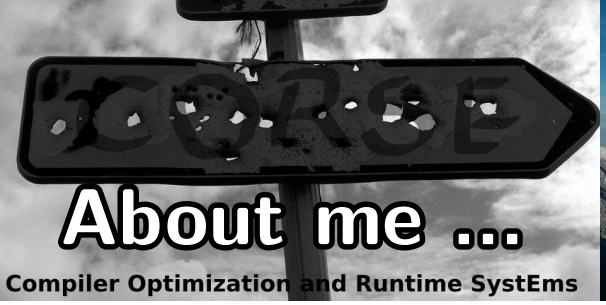
- PhD in 2014 from the University of Grenoble (UJF)
  - in partnership with STMicroelectronics
- Postdoc in same team to continue my PhD work
  - CORSE team
    - ★ Compiler Optimization and Runtime Systems for high-performance and low-energy consumption




# About me ...

Compiler Optimization and Runtime SystEms

- PhD in 2014 from the University of Grenoble (UJF)
  - in partnership with STMicroelectronics
- Postdoc in same team to continue my PhD work
  - CORSE team
    - ★ Compiler Optimization and Runtime Systems for high-performance and low-energy consumption
- Funds from NANO2017 DEMA project

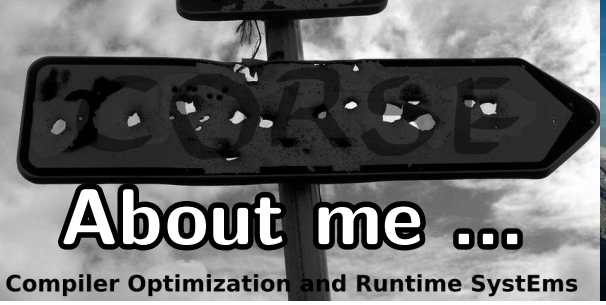



# About me ...

Compiler Optimization and Runtime SystEms

- PhD in 2014 from the University of Grenoble (UJF)
  - in partnership with STMicroelectronics
- Postdoc in same team to continue my PhD work
  - CORSE team
    - ★ Compiler Optimization and Runtime Systems for high-performance and low-energy consumption
- Funds from NANO2017 DEMA project
  - Debugging Embedded Multicore Application

    ⇒ Improve interactive and performance debugging for multimedia and concurrent applications running on multicore embedded systems.




# About me ...

Compiler Optimization and Runtime SystEms

- PhD in 2014 from the University of Grenoble (UJF)
  - in partnership with STMicroelectronics
- Postdoc in same team to continue my PhD work
  - CORSE team
    - ★ Compiler Optimization and Runtime Systems for high-performance and low-energy consumption
- Funds from NANO2017 DEMA project
  - Debugging Embedded Multicore Application

  ⇒ Improve interactive and performance debugging for multimedia and concurrent applications running on multicore embedded systems.

  - ★ Jean-François Méhaut (**UJF**), Albert Cohen (**INRIA Paris**), Karine Heydemann (**Uni. Paris VI**), Miguel Santana (**STMicroelectronics**)
  - ★ Co-funded by French Ministry of Industry and local authorities in the Grenoble area


## Consumer Electronics Devices

- 4K digital televisions
- Smartphones
- Hand-held music players

- High-resolution multimedia apps
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...


## Consumer Electronics Devices

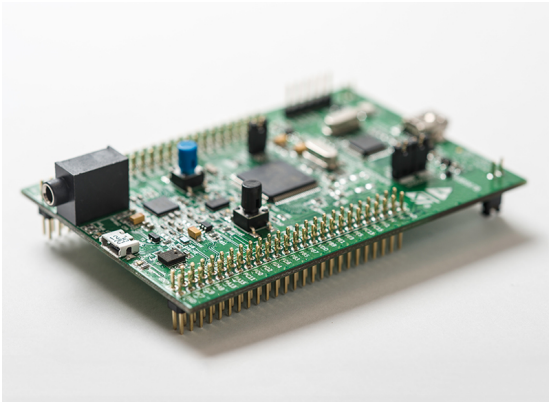
- 4K digital televisions
- Smartphones
- Hand-held music players

- High-resolution multimedia apps
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...

$\Rightarrow$ high performance expectations.


informatics mathematics

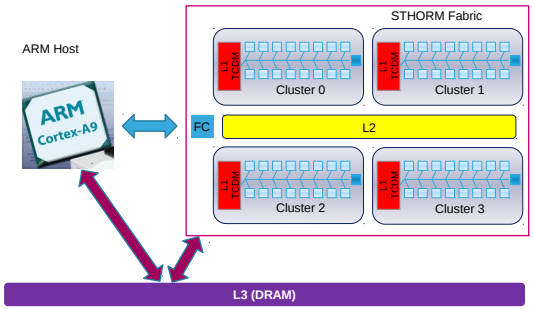

*Current applications have high performance expectations...*



⇒ important demand for:

- Powerful parallel architectures
- High-level development methodologies
- Efficient verification & validation tools

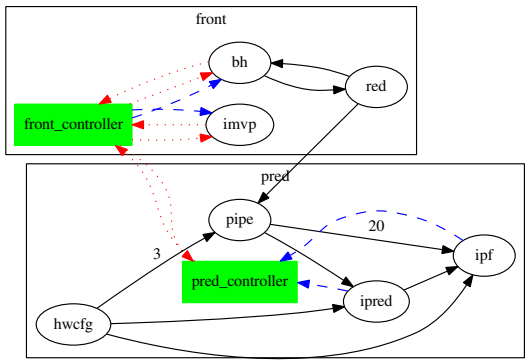

*Current applications have high performance expectations...*




## ⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies
- Efficient verification & validation tools

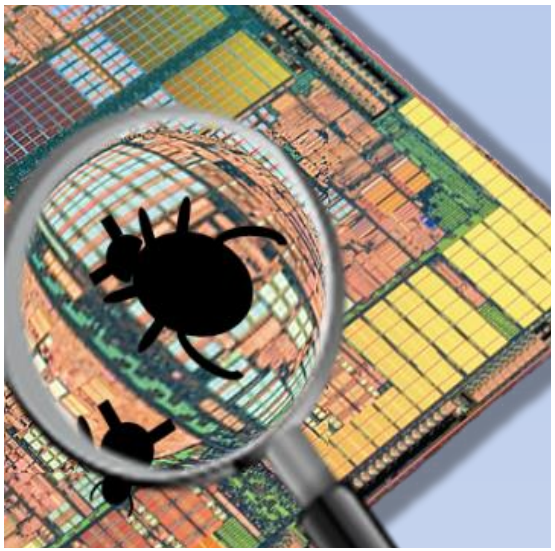

*Current applications have high performance expectations...*




⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies
  - Programming models & environments
- Efficient verification & validation tools


*Current applications have high performance expectations...*



⇒ important demand for:

- Powerful parallel architectures
  - MultiProcessor Systems-on-a-Chip (MPSoCs)
- High-level development methodologies
  - Programming models & environments
- Efficient verification & validation tools
  - Our research effort

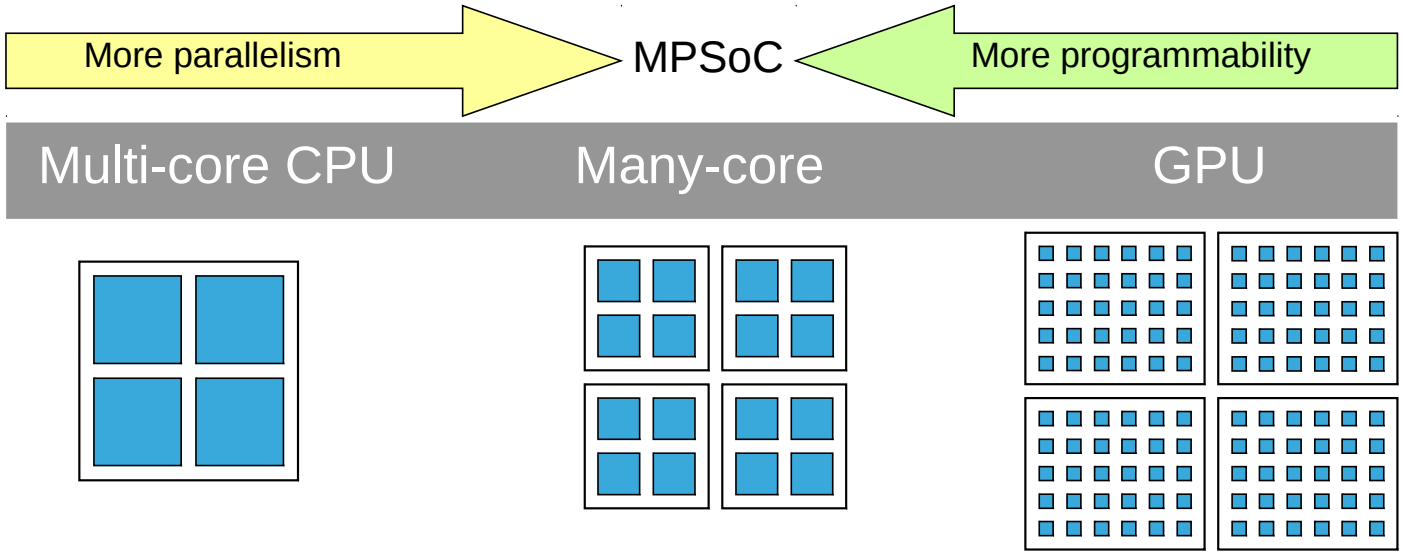

# Agenda

Compiler Optimization and Runtime SystEms

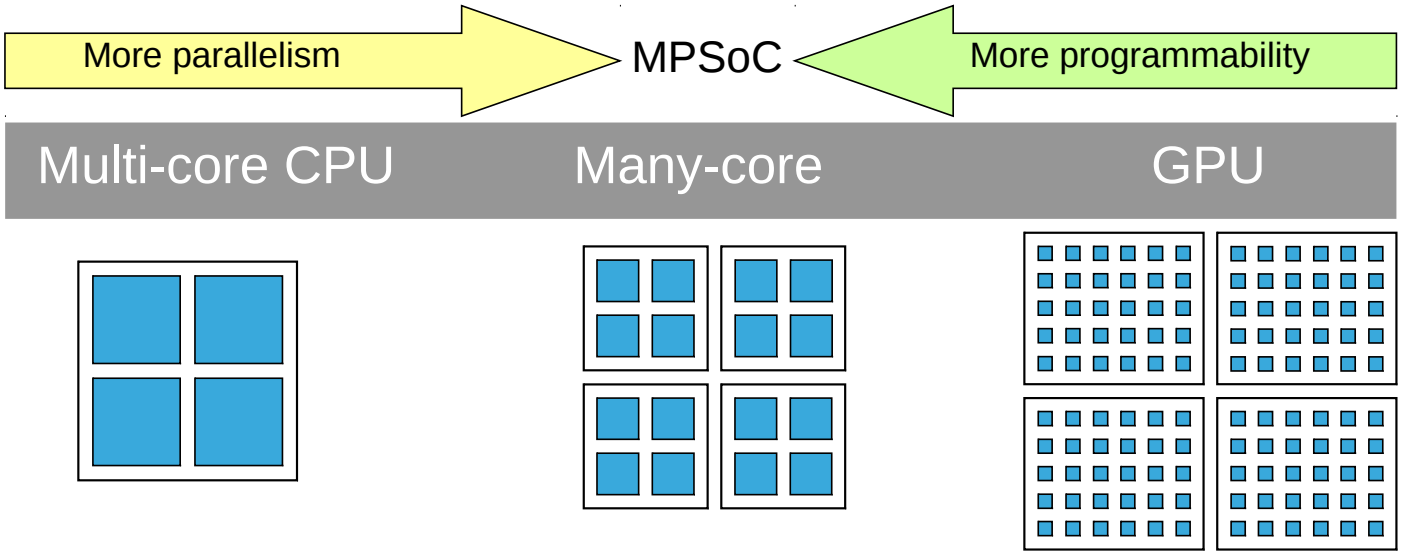

# Agenda

Compiler Optimization and Runtime SystEms

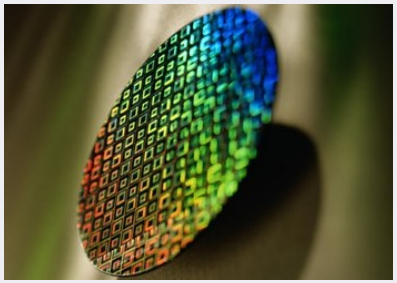
## MultiProcessor System on-a-Chip



## MultiProcessor System on-a-Chip




- Many-core processor for embedded systems
- Low energy-consumption
- Heterogeneous computing power




How to program such complex architectures?


How to program such complex architectures?

**Programming models and environments!**

*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

broad definition!

→ it's an abstract machine

- that separates application development / lower-level concerns

*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and ...
- requiring implementations for each of these operations on all of the architectures below.

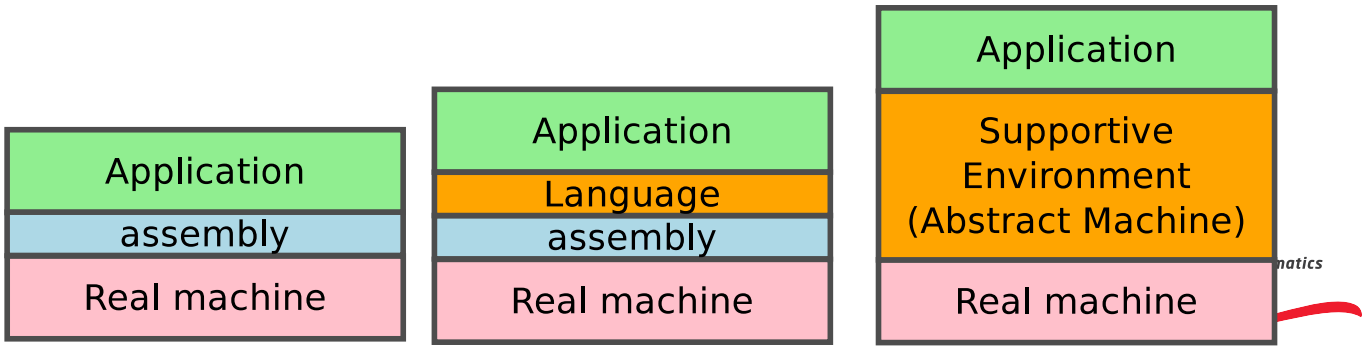
*... not so many clear definitions in the literature, so ...*

## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and

## Supportive Environment

- ... requiring implementations for each of these operations on all of the architectures below.



*... not so many clear definitions in the literature, so ...*

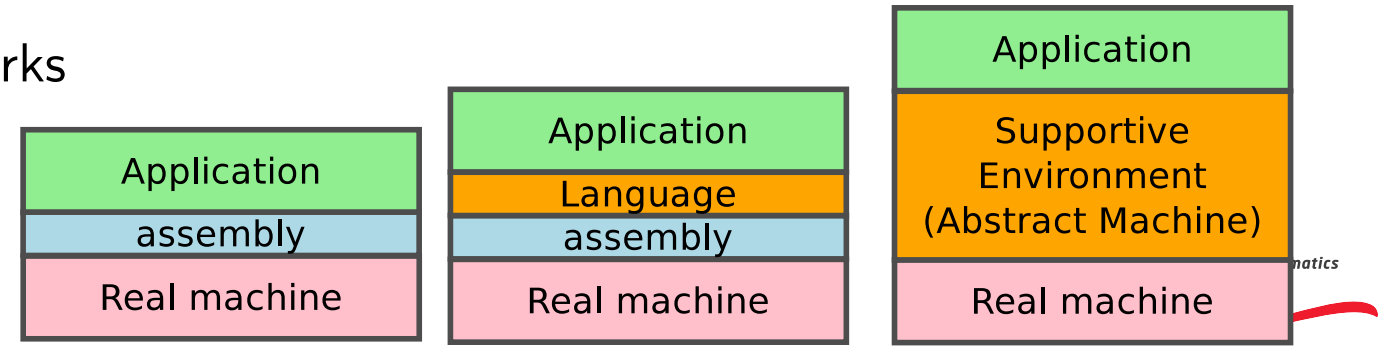
## Programming Model (Skillicorn and Talia '98)

- A model is an abstract machine...
- providing certain operations to the programming level above and

## Supportive Environment

- ... requiring implementations for each of these operations on all of the architectures below.

- development frameworks
- runtime libraries
- APIs





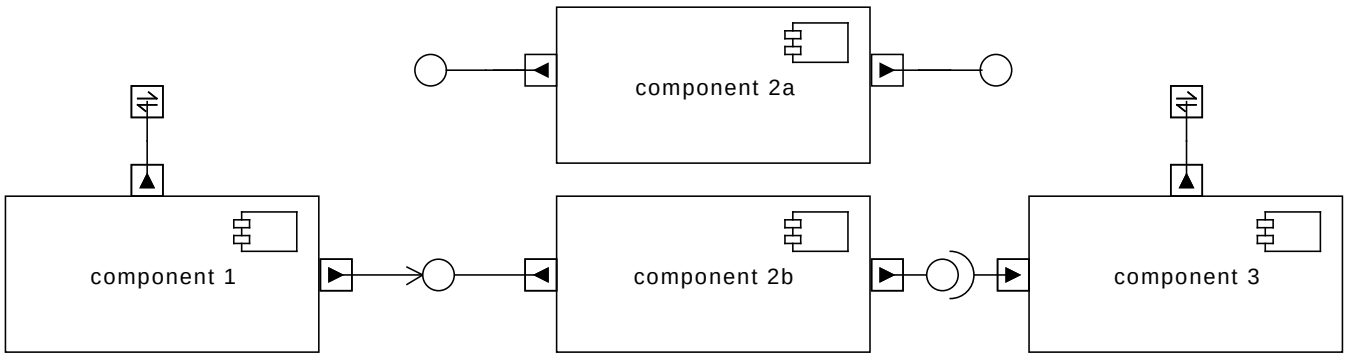

Components

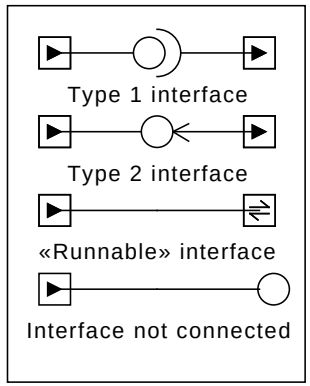
Dataflow

## Components

- Code/data encapsulation
- Software reuse
- Service contracts

## Dataflow

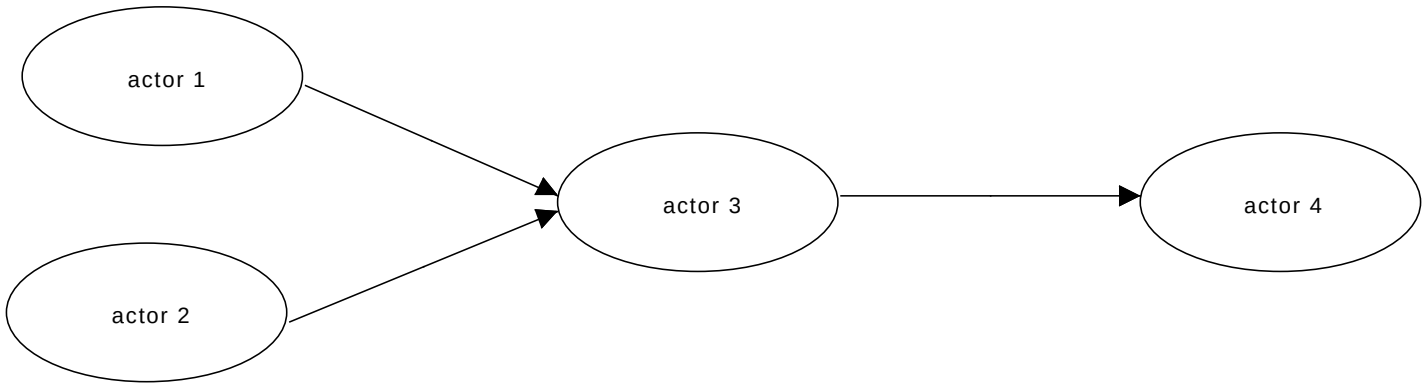


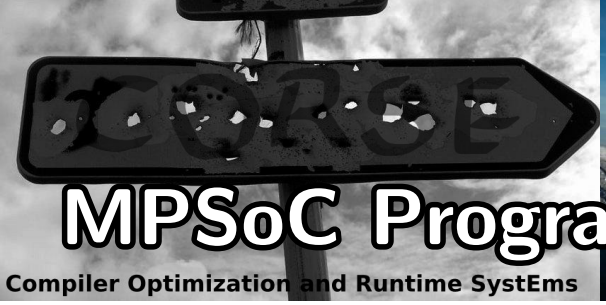



## Components

## Dataflow

- Emphasis on streams of data
- Implicit parallelism
- Roots in graph theory






# MPSoC Programming and Debugging

Compiler Optimization and Runtime SystEms

## Components

- Code/data encapsulation
- Software reuse
- Service contracts

## Dataflow

- Emphasis on streams of data
- Implicit parallelism
- Roots in graph theory

large range of computing problems...

informatics mathematics Inria


## Components

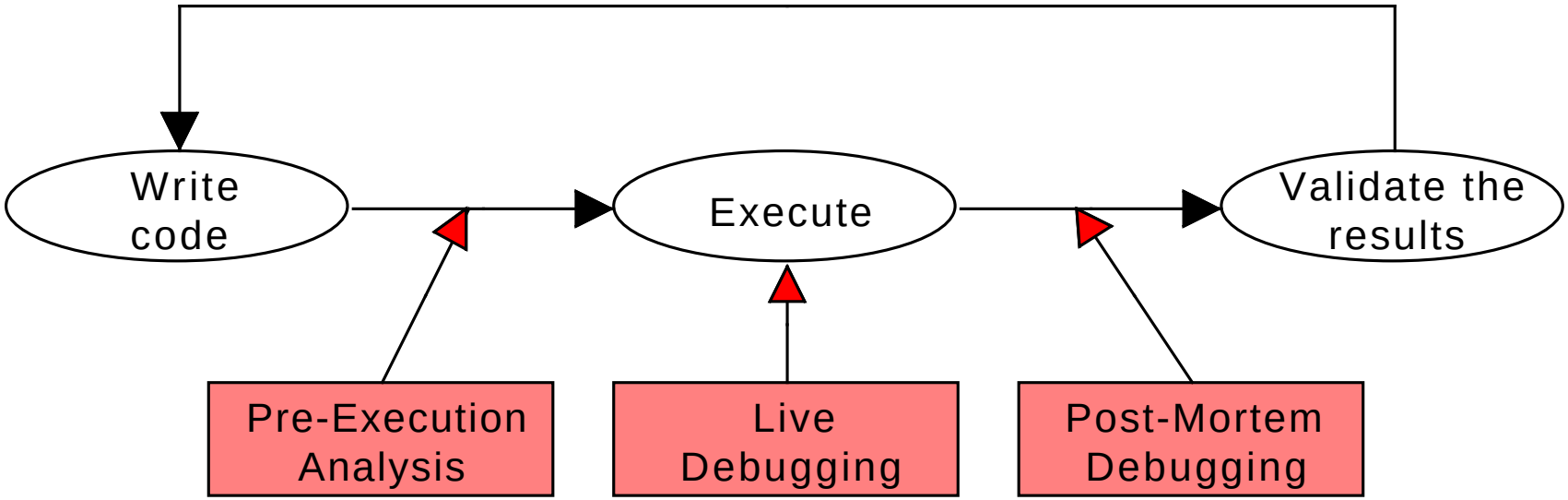
- Code/data encapsulation
- Software reuse
- Service contracts

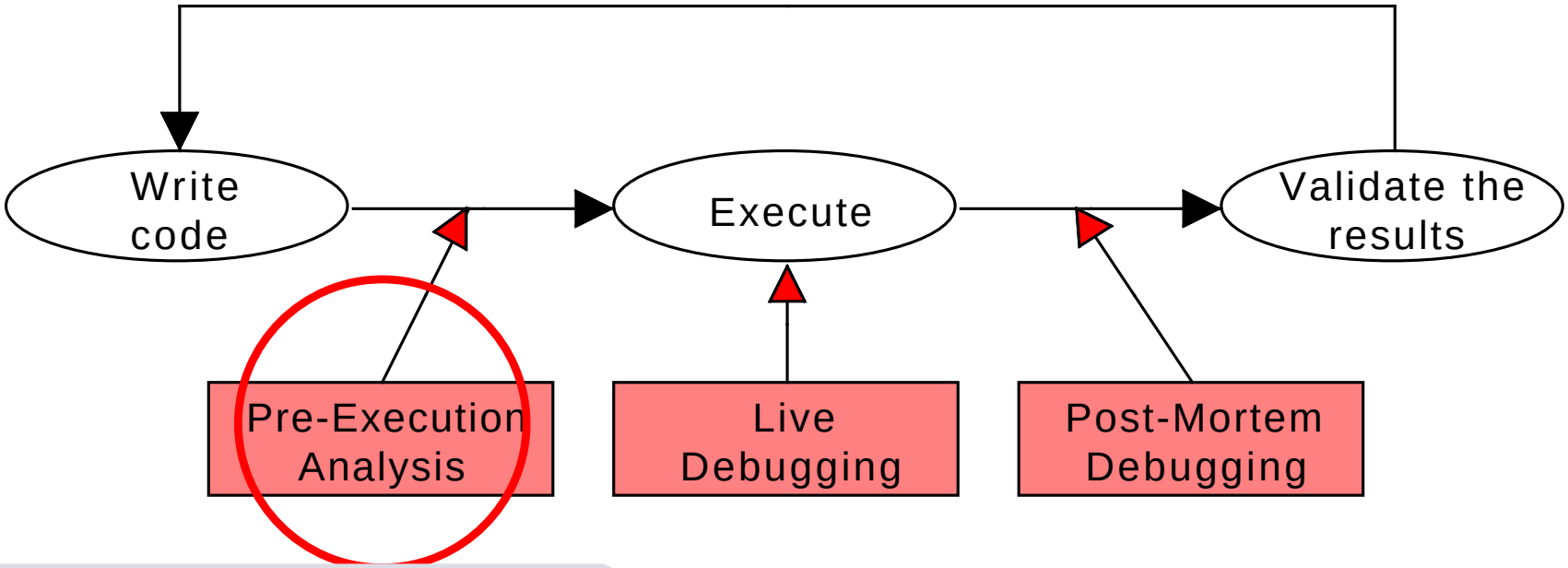
## Dataflow

- Emphasis on streams of data
- Implicit parallelism
- Roots in graph theory

large range of computing problems...

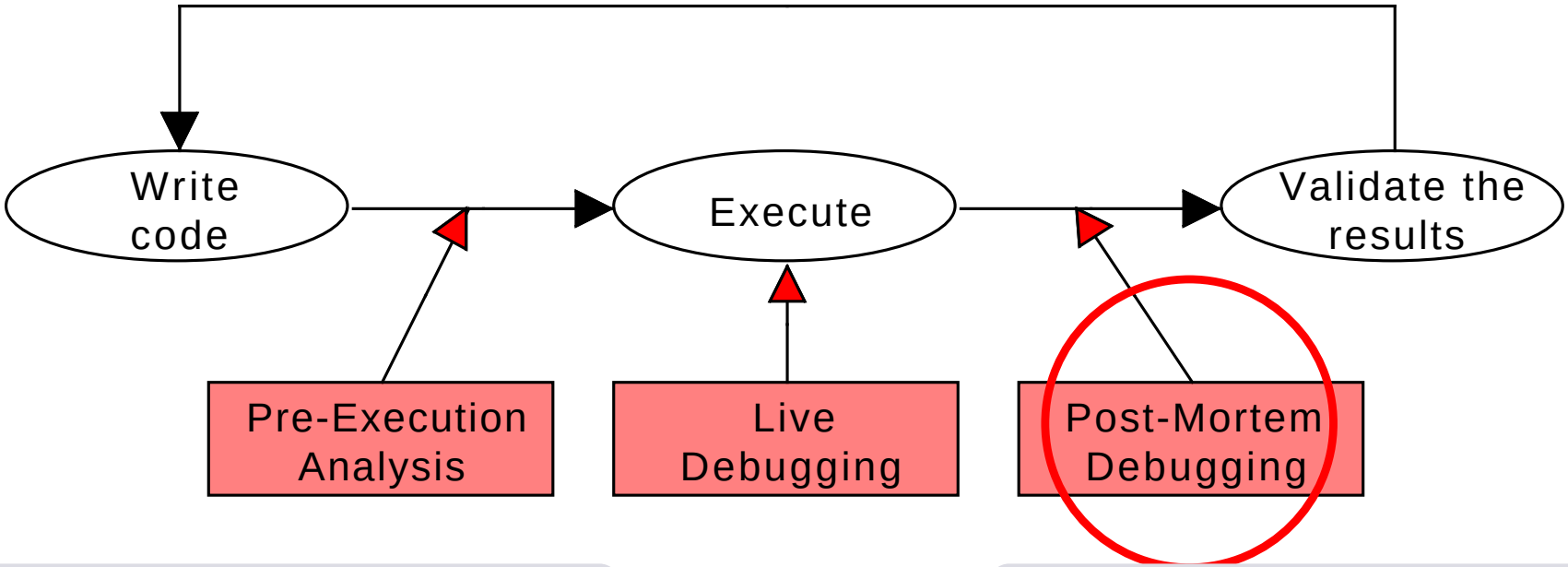
... but what about Verification & Validation of MPSoC applications?


Write code
Execute
Validate the results
Pre-Execution Analysis
Live Debugging
Post-Mortem Debugging




## Static/Formal Analysis

+ May be exhaustive
  - synchronous dataflow
− Not always feasible
  - dynamic behaviors

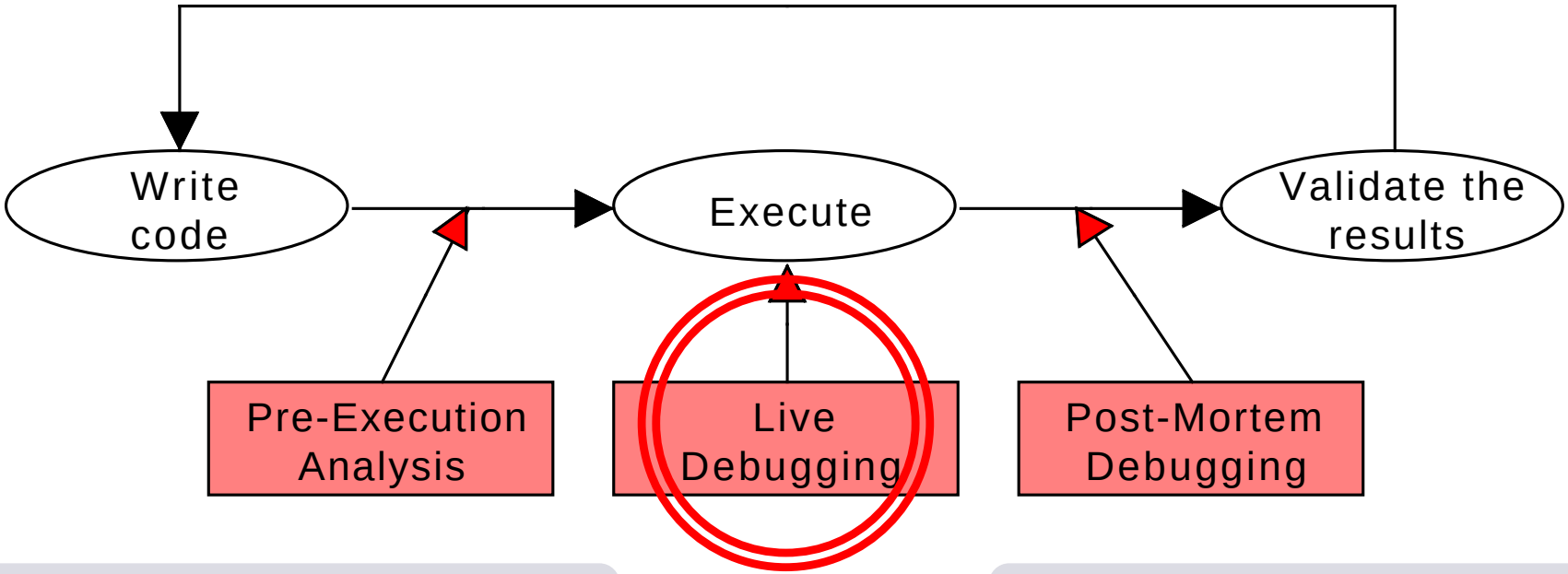



## Static/Formal Analysis

+ May be exhaustive
  - synchronous dataflow
- Not always feasible
  - dynamic behaviors

## Trace Analysis

- Manual/data-mining
+ long/time critical run
- What to trace?
  - fixed # of trace-points




## Static/Formal Analysis

+ May be exhaustive
  - synchronous dataflow
- Not always feasible
  - dynamic behaviors

## Trace Analysis

- Manual/data-mining
+ long/time critical run
- What to trace?
  - fixed # of trace-points

## Live and Interactive Debugging (i.e. a la GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

## Live and Interactive Debugging (i.e. a la GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

## Live and Interactive Debugging (i.e. a la GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

## Live and Interactive Debugging (i.e. a la GDB)

- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?


## Live and Interactive Debugging (i.e. a la GDB)

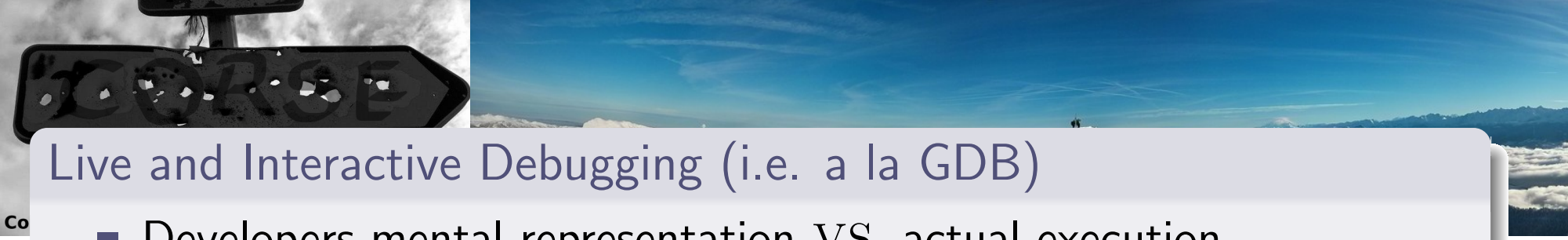
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!



## Live and Interactive Debugging (i.e. a la GDB)

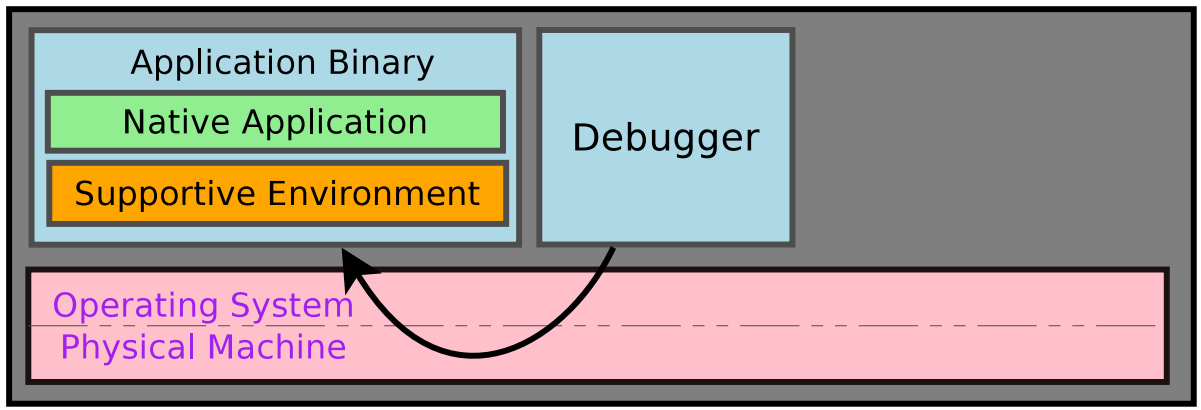
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

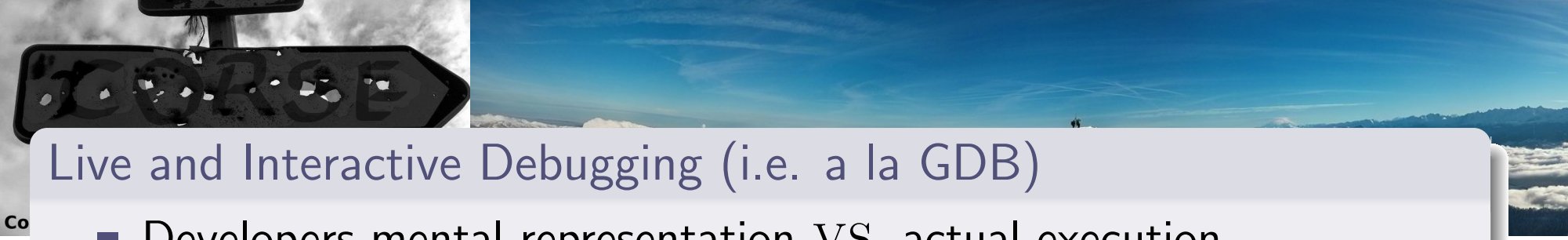
- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!





## Live and Interactive Debugging (i.e. a la GDB)

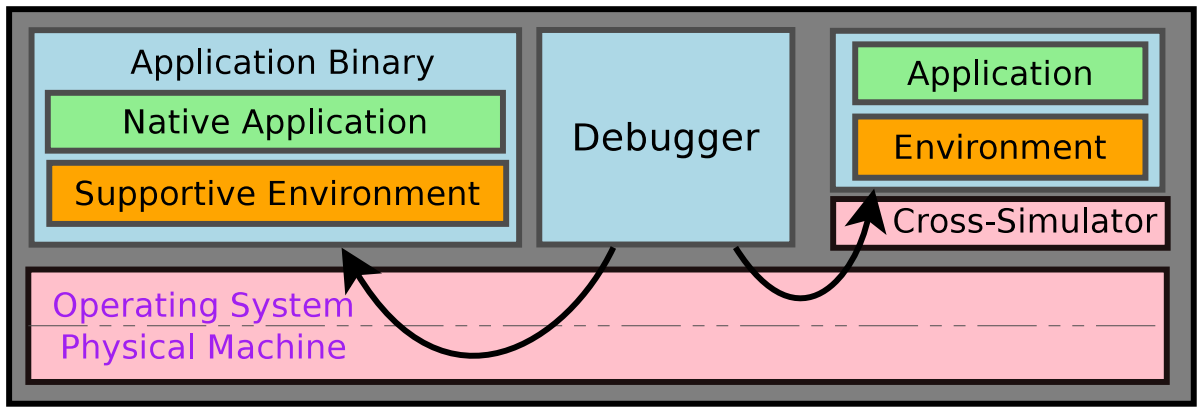
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

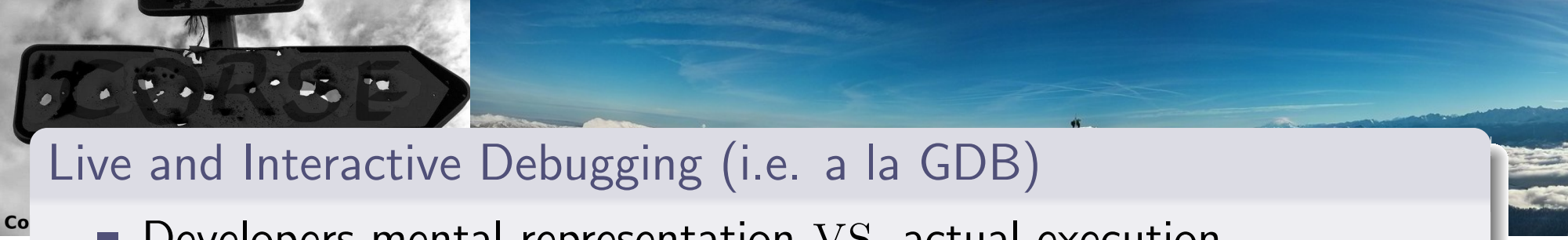
- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!





## Live and Interactive Debugging (i.e. a la GDB)

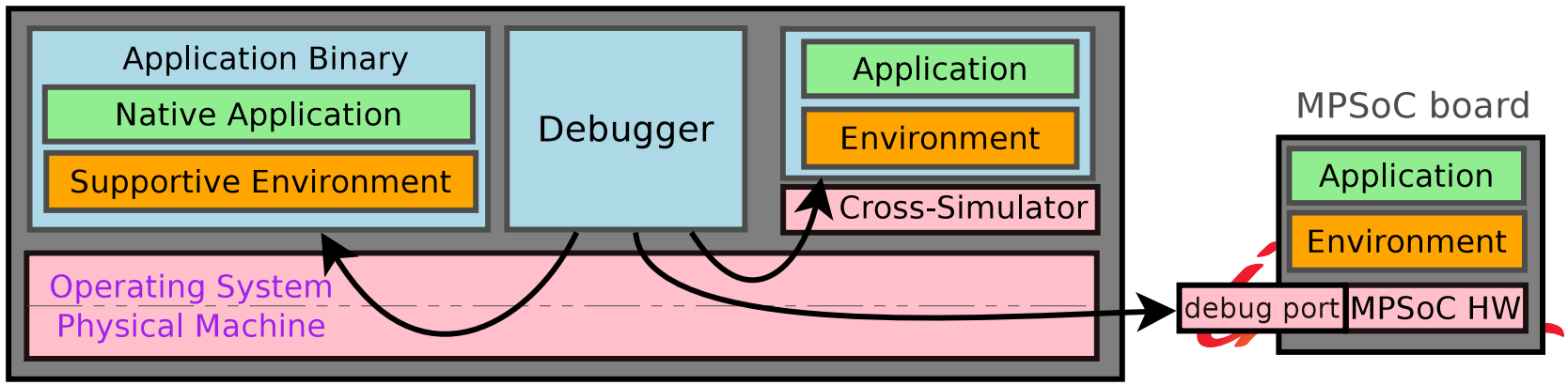
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

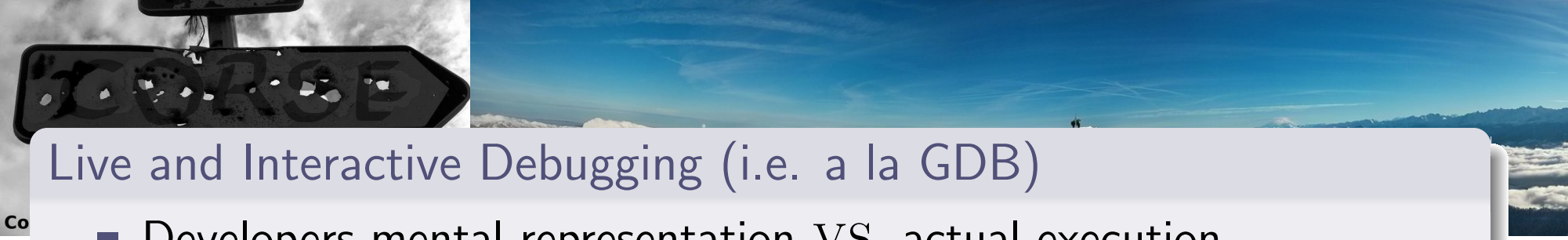
- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!





## Live and Interactive Debugging (i.e. a la GDB)

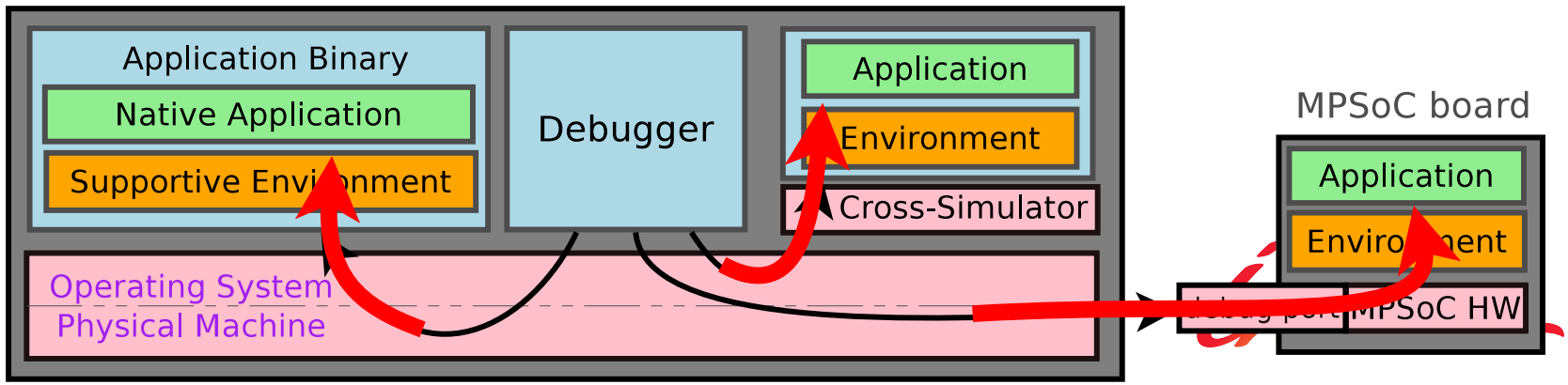
- Developers mental representation VS. actual execution
- Understand the different steps of the execution

- Instruction breakpoints
- Memory watchpoints
- Event catchpoints

- Step-by-step execution
  - Source code or assembly level
- Memory and processor inspection

What about the Supportive Environment?

Debuggers cannot access the *abstract* machine!




## Objective

Provide developers with means to
**better understand the state of the high-level applications**
**and control more easily their execution**,
suitable for various models and environments.

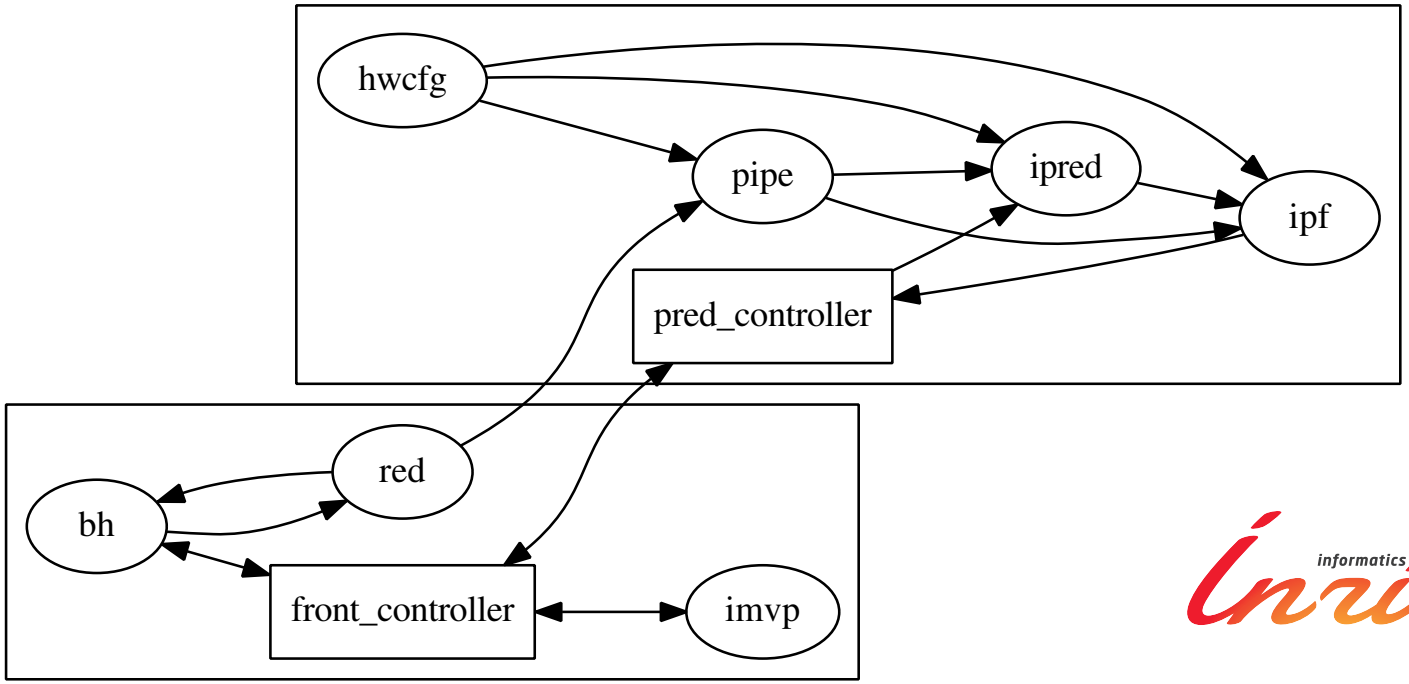

# Agenda

Compiler Optimization and Runtime SystEms


Idea: **Integrate programming model concepts in interactive debugging**

## 1 Provide a Structural Representation

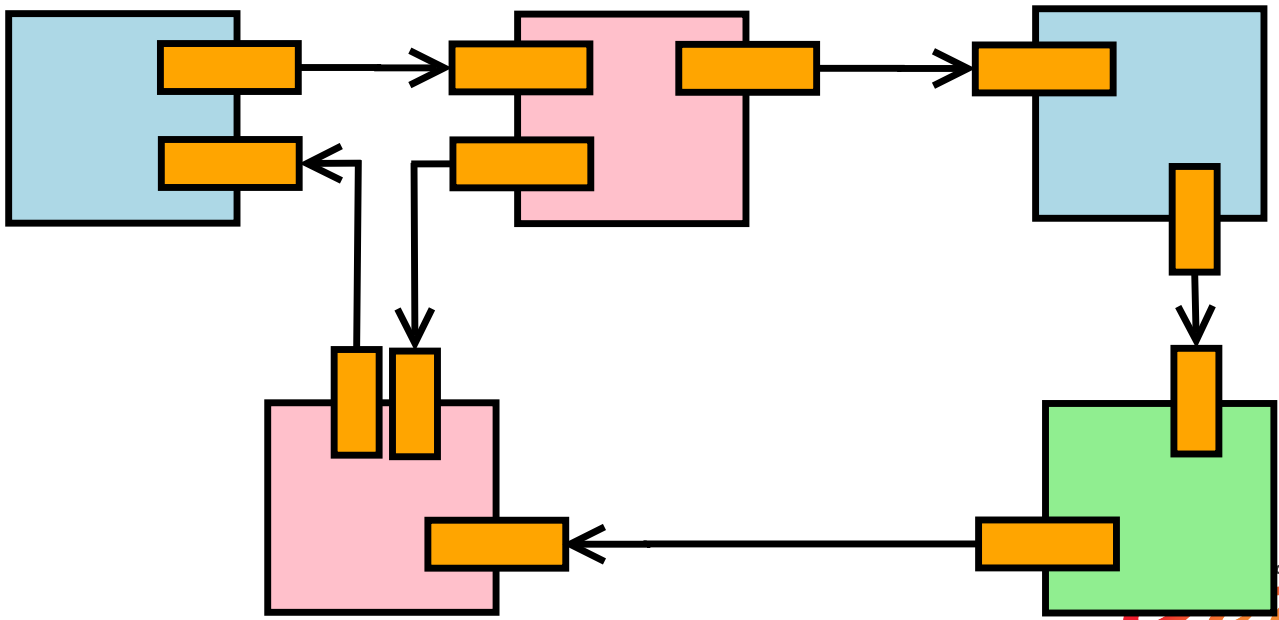
- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations




Dataflow graph
from the case-study

## 1 Provide a Structural Representation

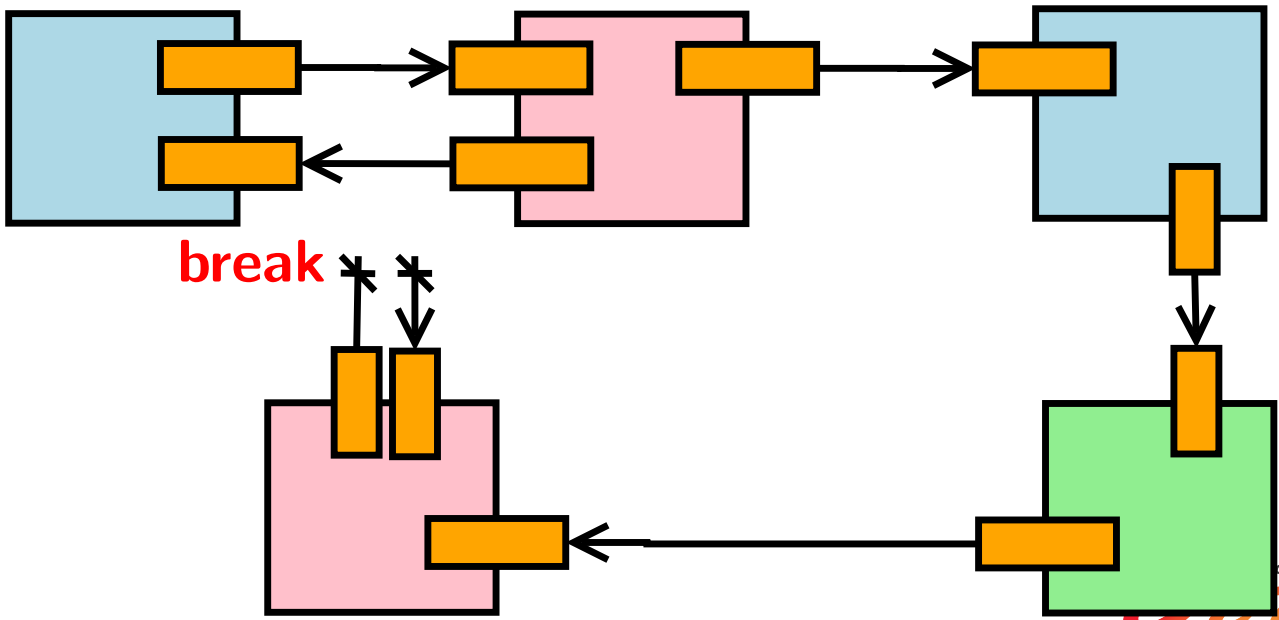
- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations



Reconfiguration of an application based on components

## 1 Provide a Structural Representation

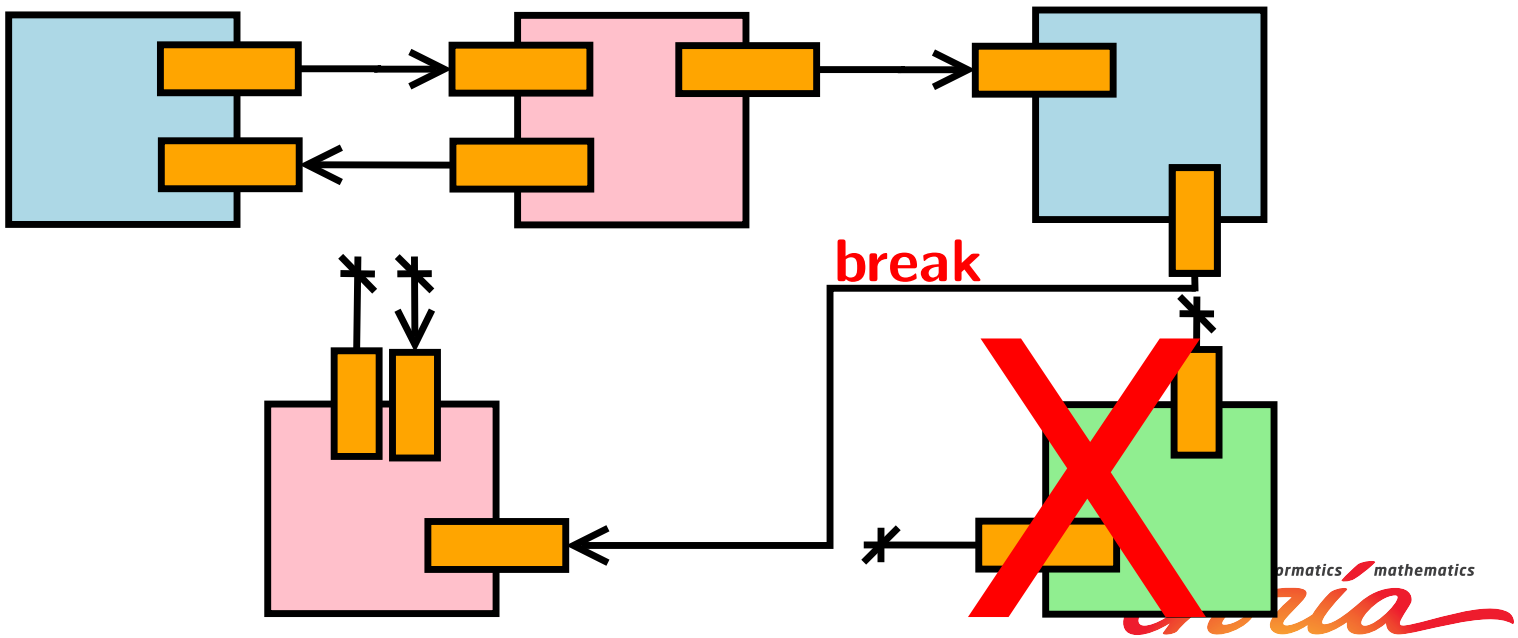
- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations




Reconfiguration of an application based on components

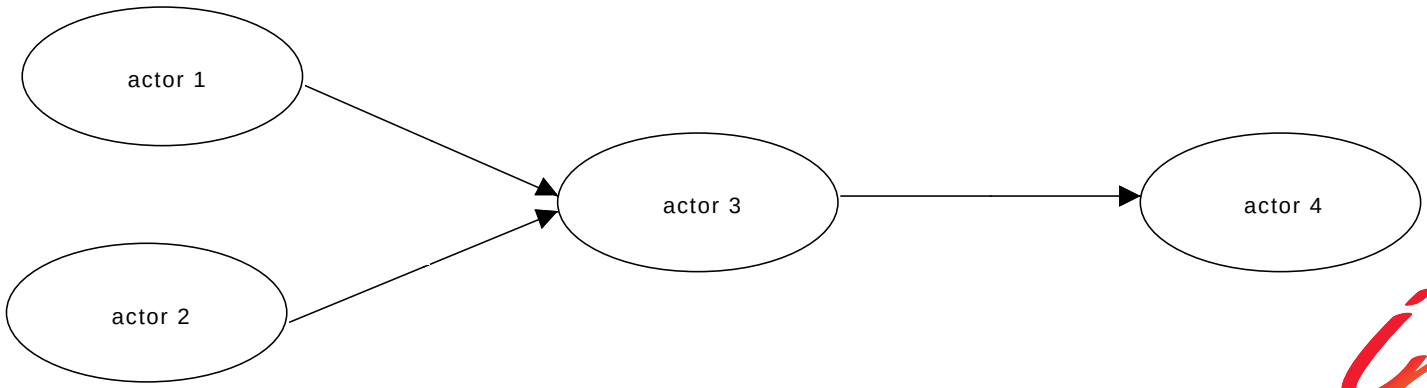
## 1 Provide a Structural Representation

- Draw application architecture diagrams
- Represent the relationship between the entities
- Offer catchpoints on architecture-related operations



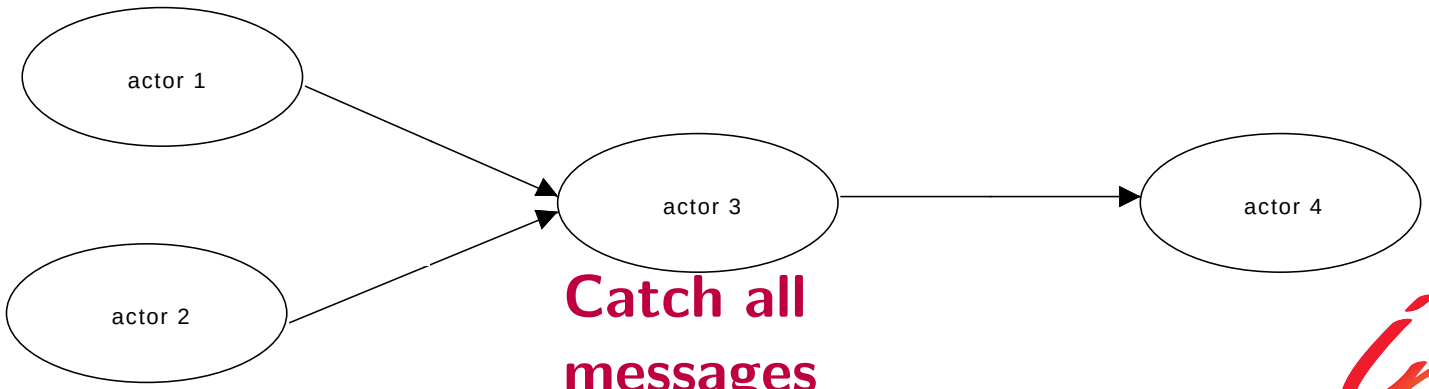
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



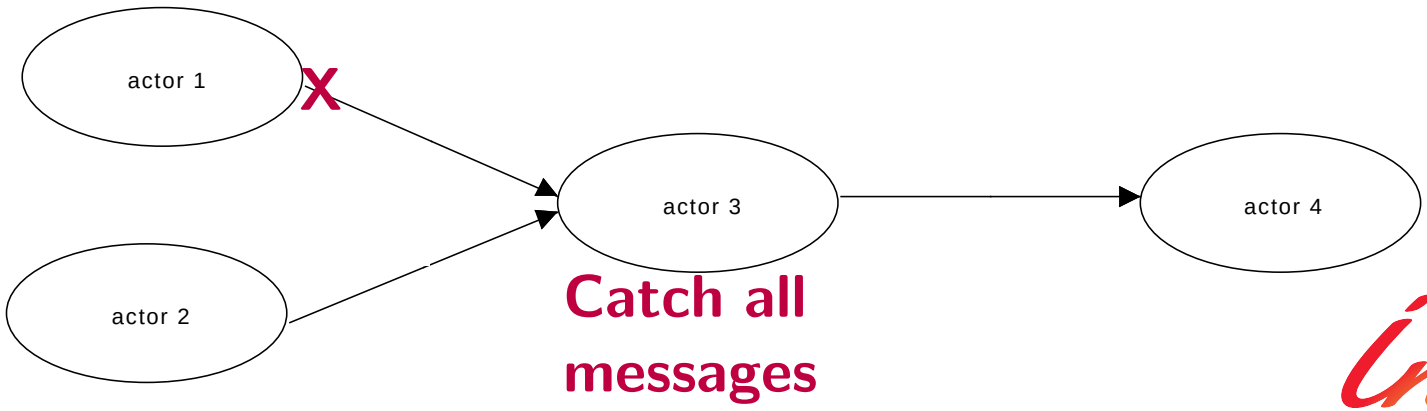
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



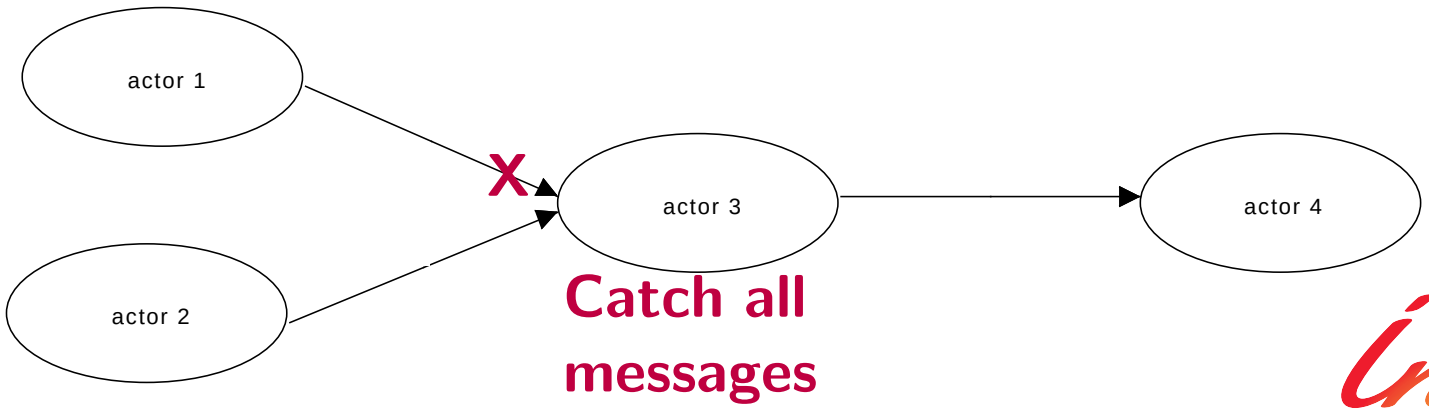
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



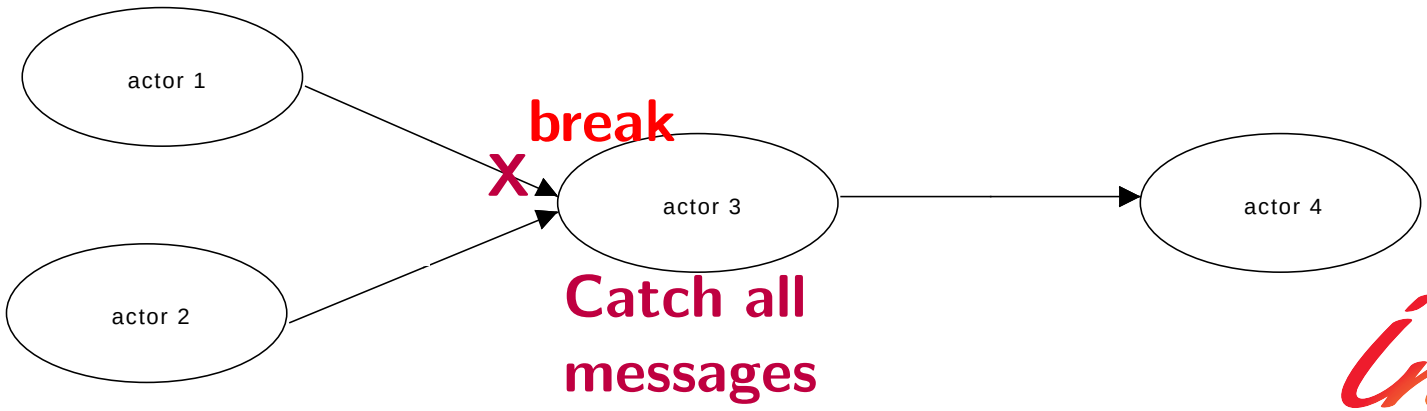
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



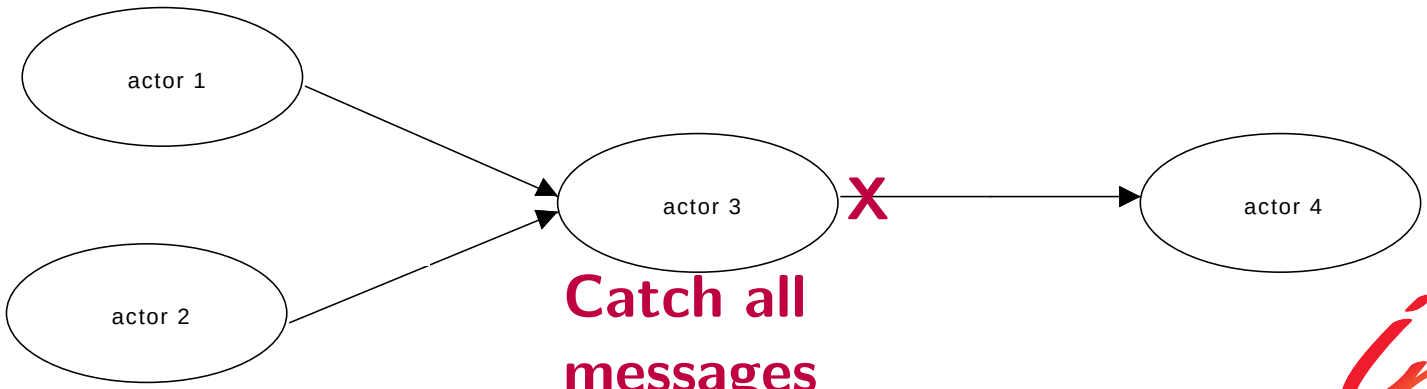
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



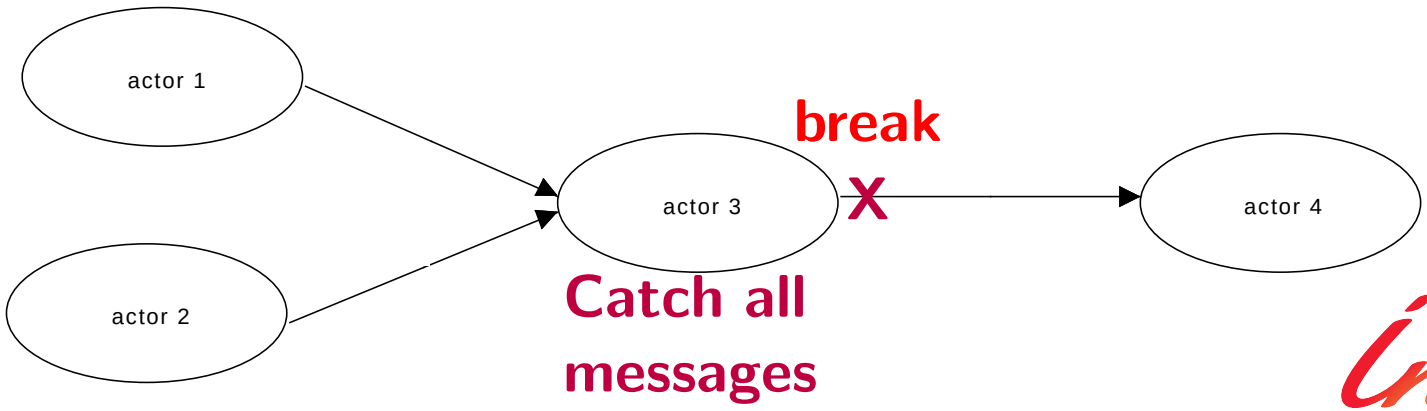
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



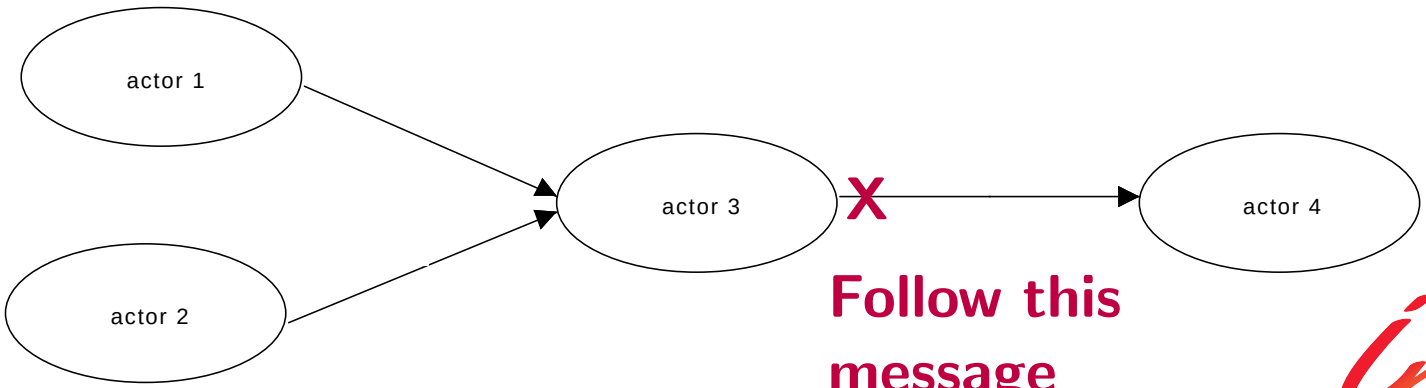
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



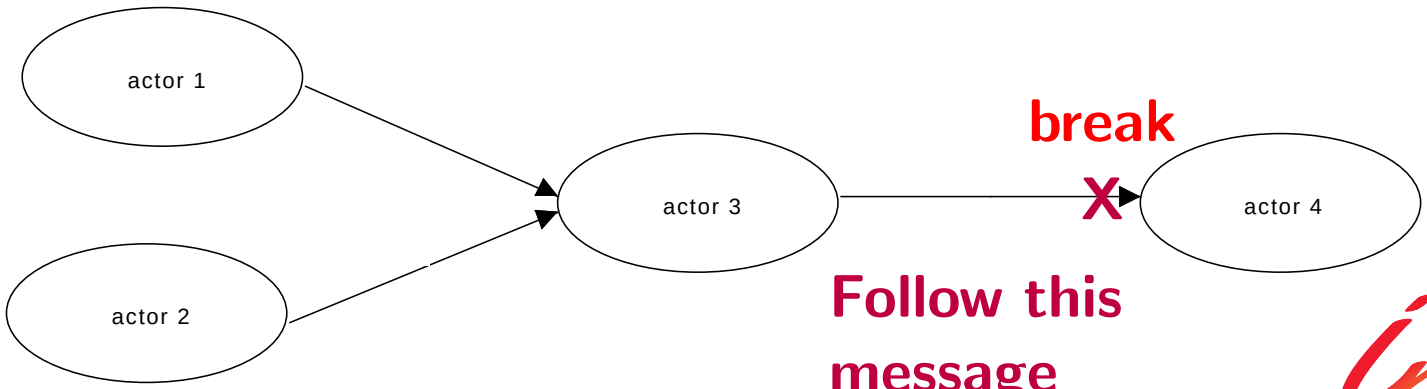
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



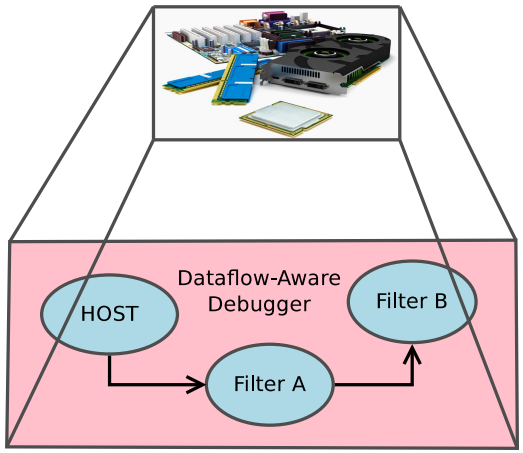
## 2 Monitor Dynamic Behaviors

- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



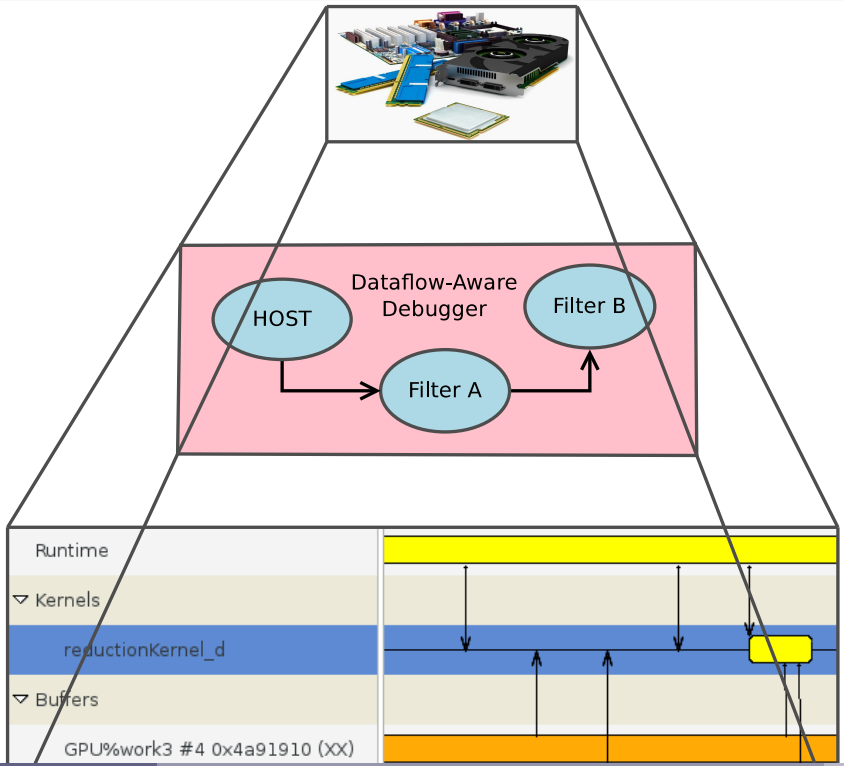
## 3 Interact with the Abstract Machine

- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



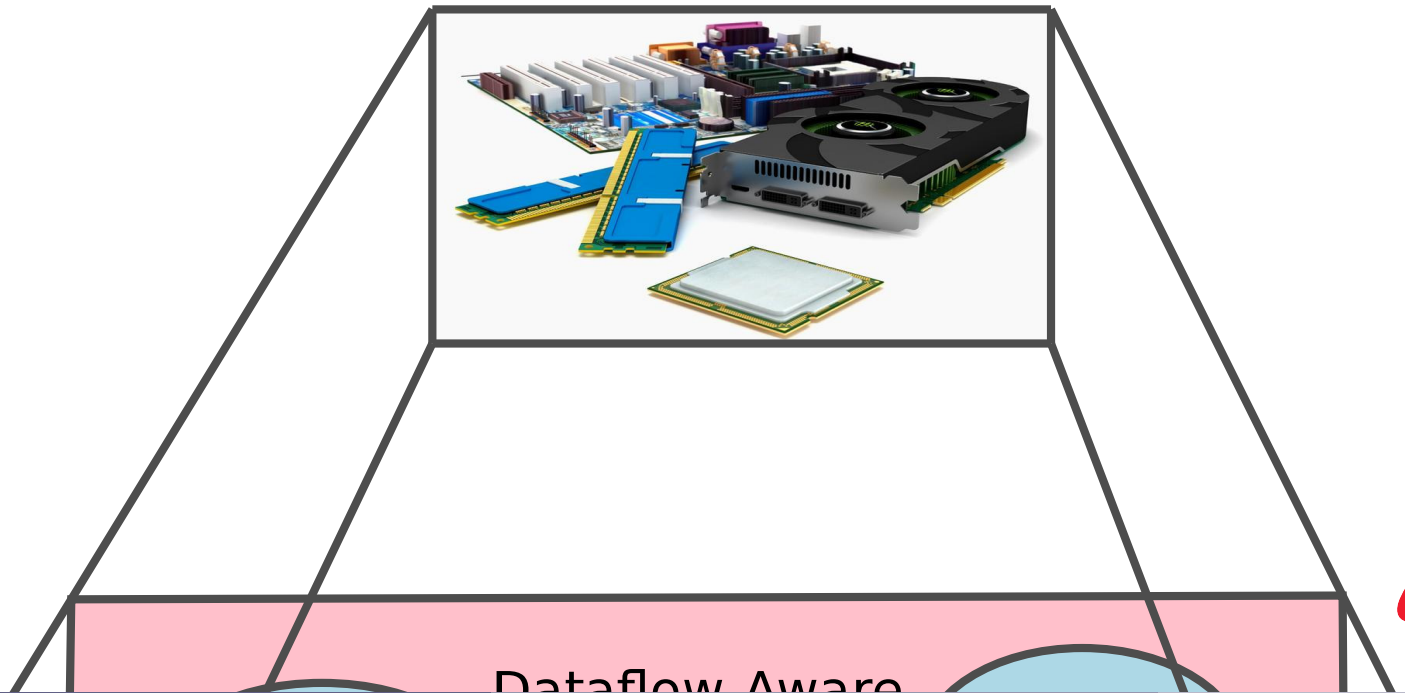
## 3 Interact with the Abstract Machine

- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



## 3 Interact with the ~~Abstract~~ Machine

- Support interactions with *real* machine
  - memory inspection
  - breakpoints
  - step-by-step

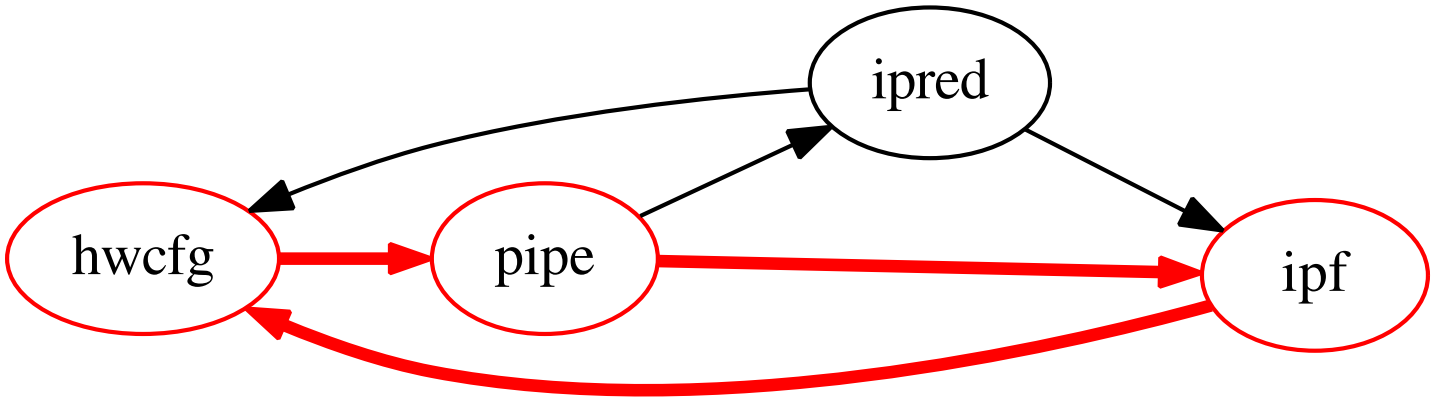



## 4 Open Up to Model and Environment Specific Features

- Follow messages over multiple entities
- User-defined constraints on the graph topology
- Deadlock detection in message-passing models

## 4 Open Up to Model and Environment Specific Features

- Follow messages over multiple entities
- User-defined constraints on the graph topology
- Deadlock detection in message-passing models




cycles in the graph of blocking communications $\Longrightarrow$ deadlock


# Programming Model Centric Debugging

Compiler Optimization and Runtime SystEms

1. Provide a Structural Representation
2. Monitor Dynamic Behaviors
3. Interact with the Abstract Machine
4. Open Up to Model and Environment Specific Features


# Agenda

Compiler Optimization and Runtime SystEms

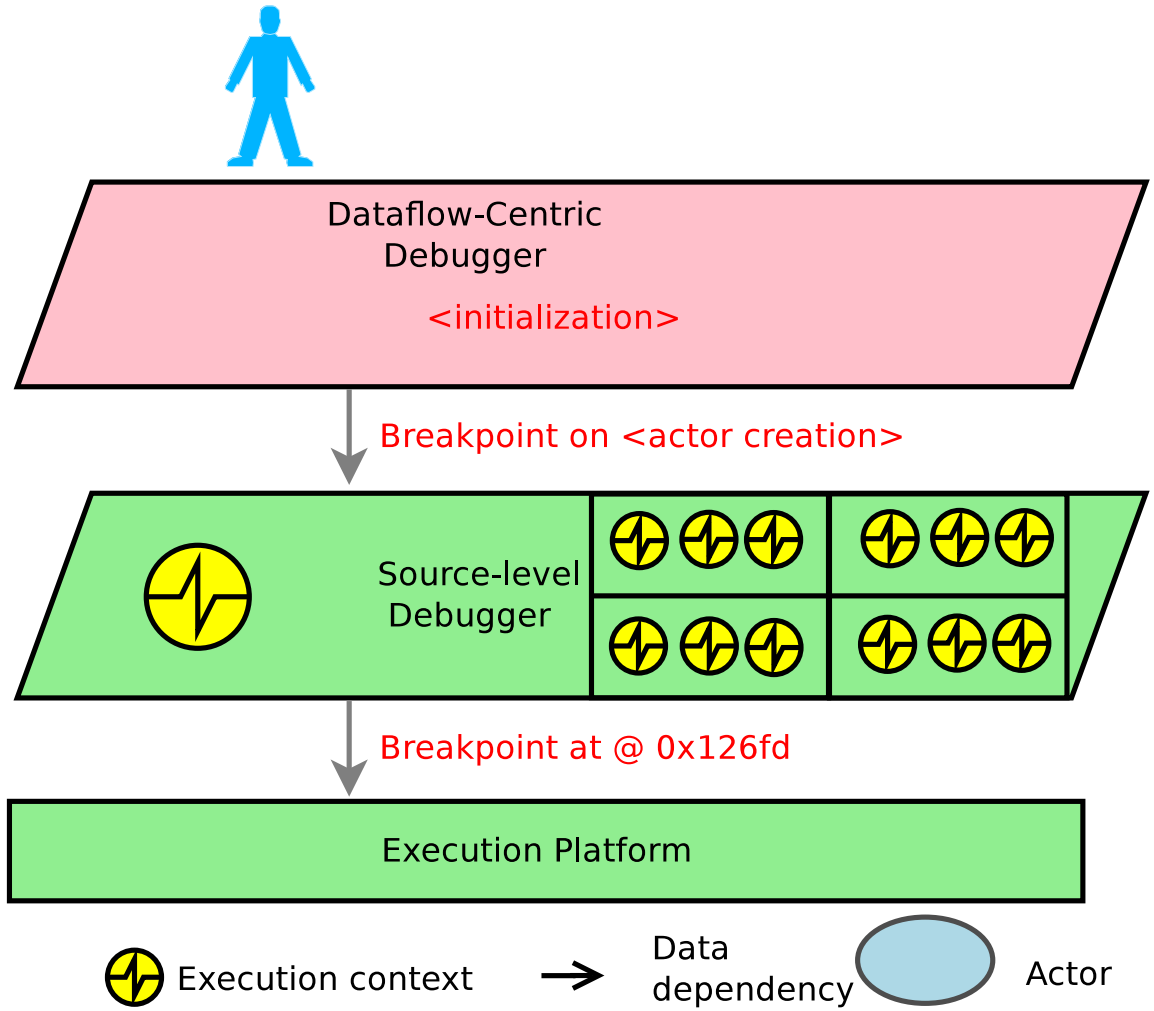

# Building Blocks of a Model-Centric Debugger

Compiler Optimization and Runtime SystEms

⇒ Detect and interpret the execution events of the runtime framework

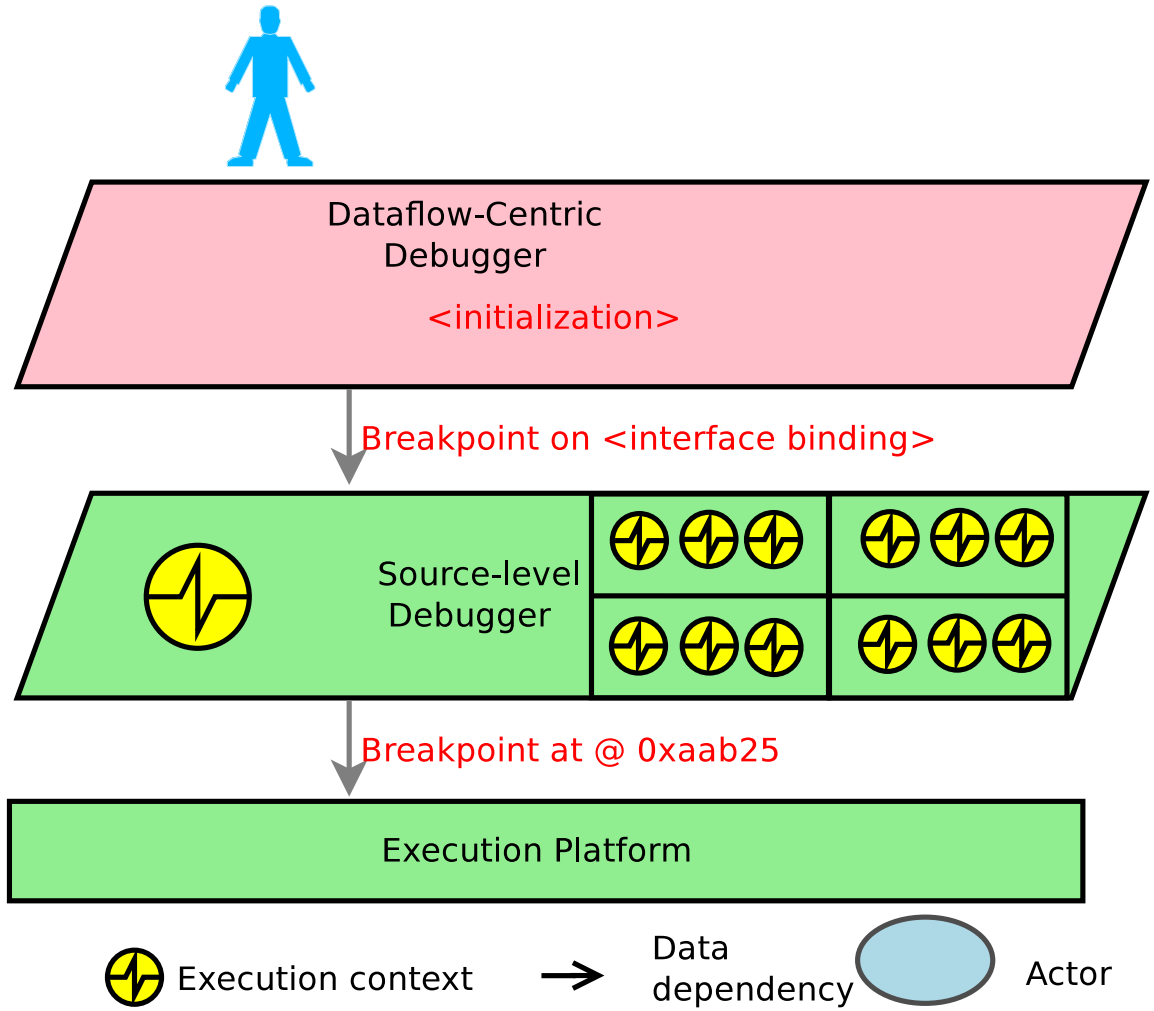
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



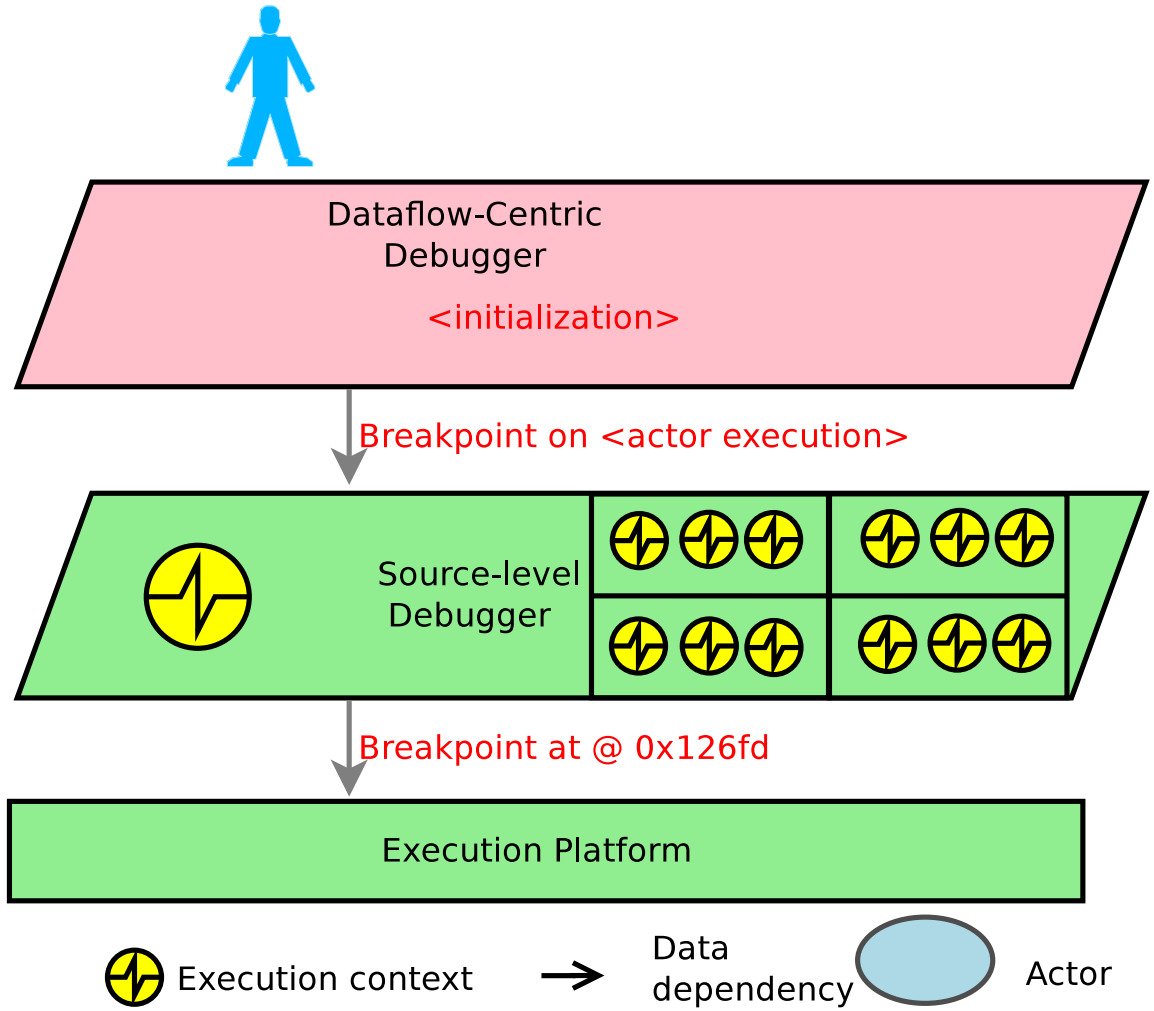
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



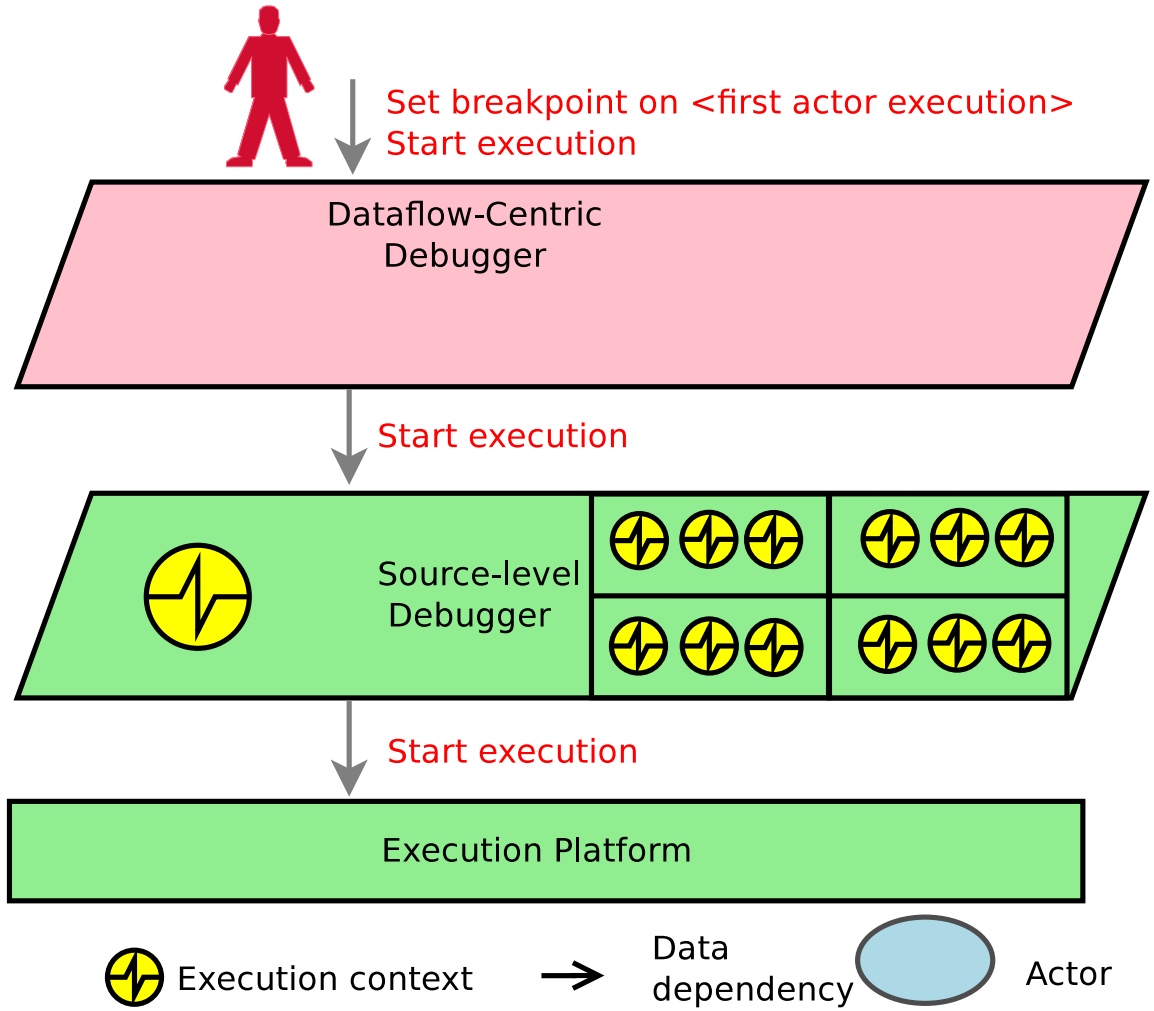
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



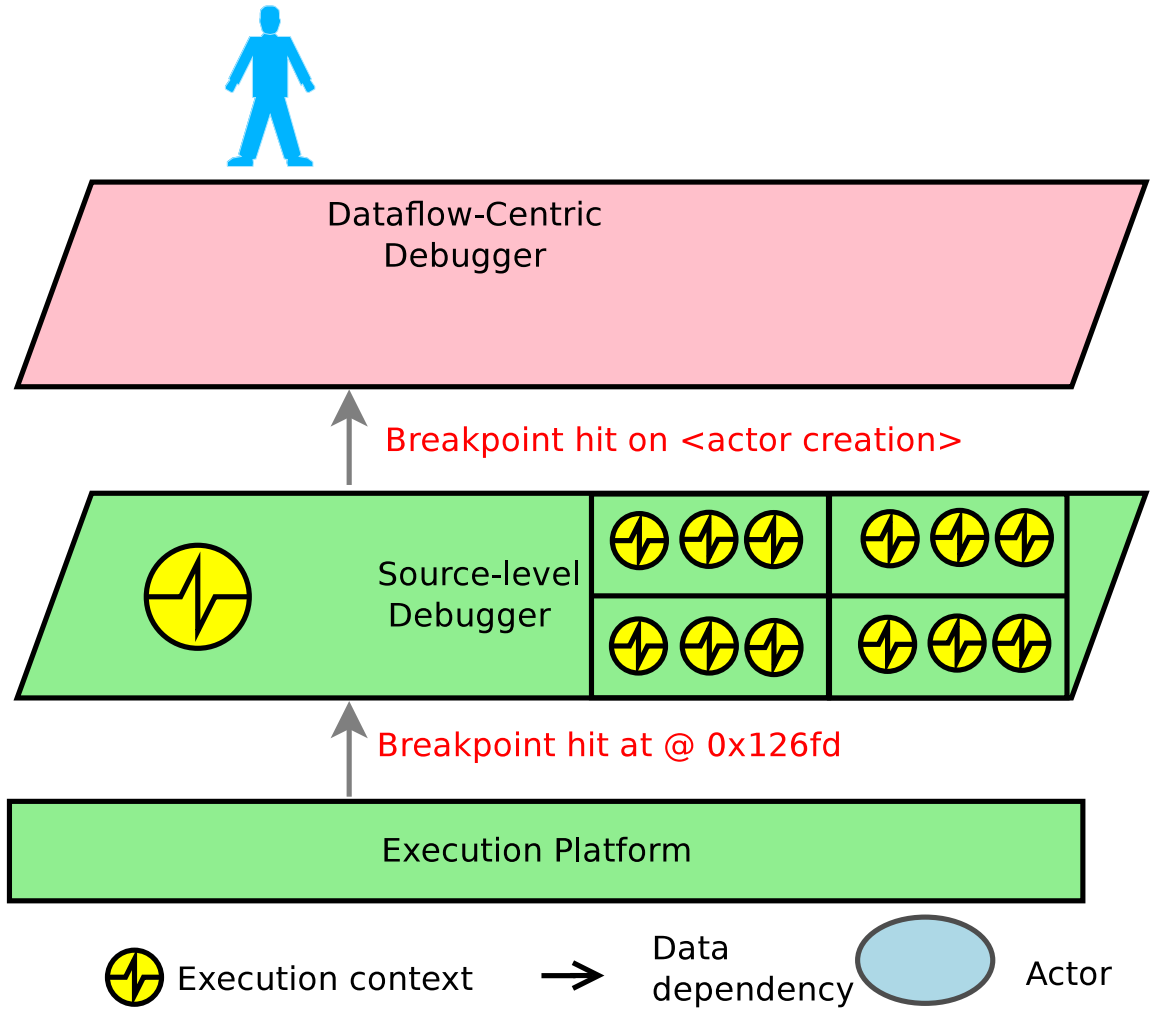
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



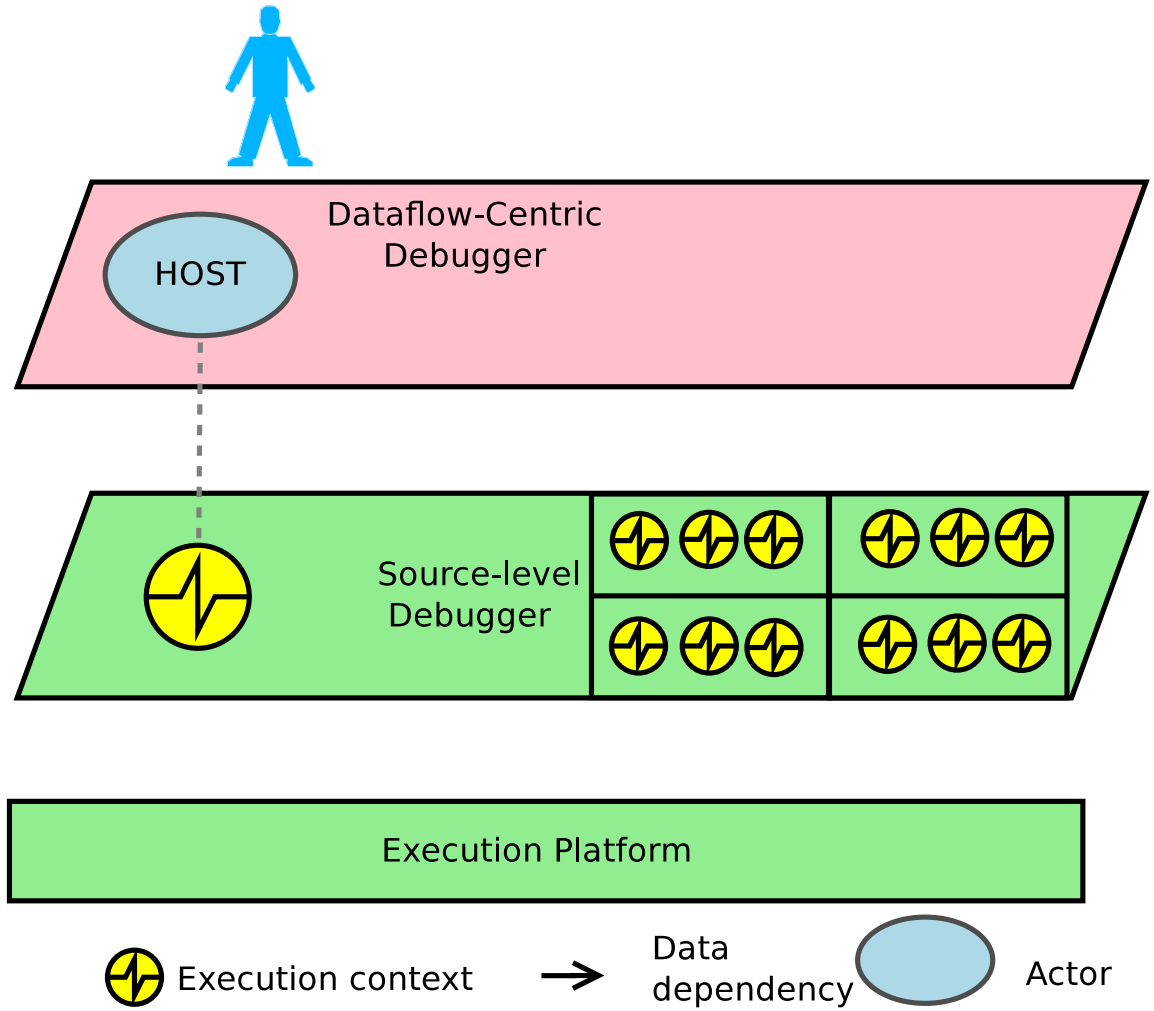
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



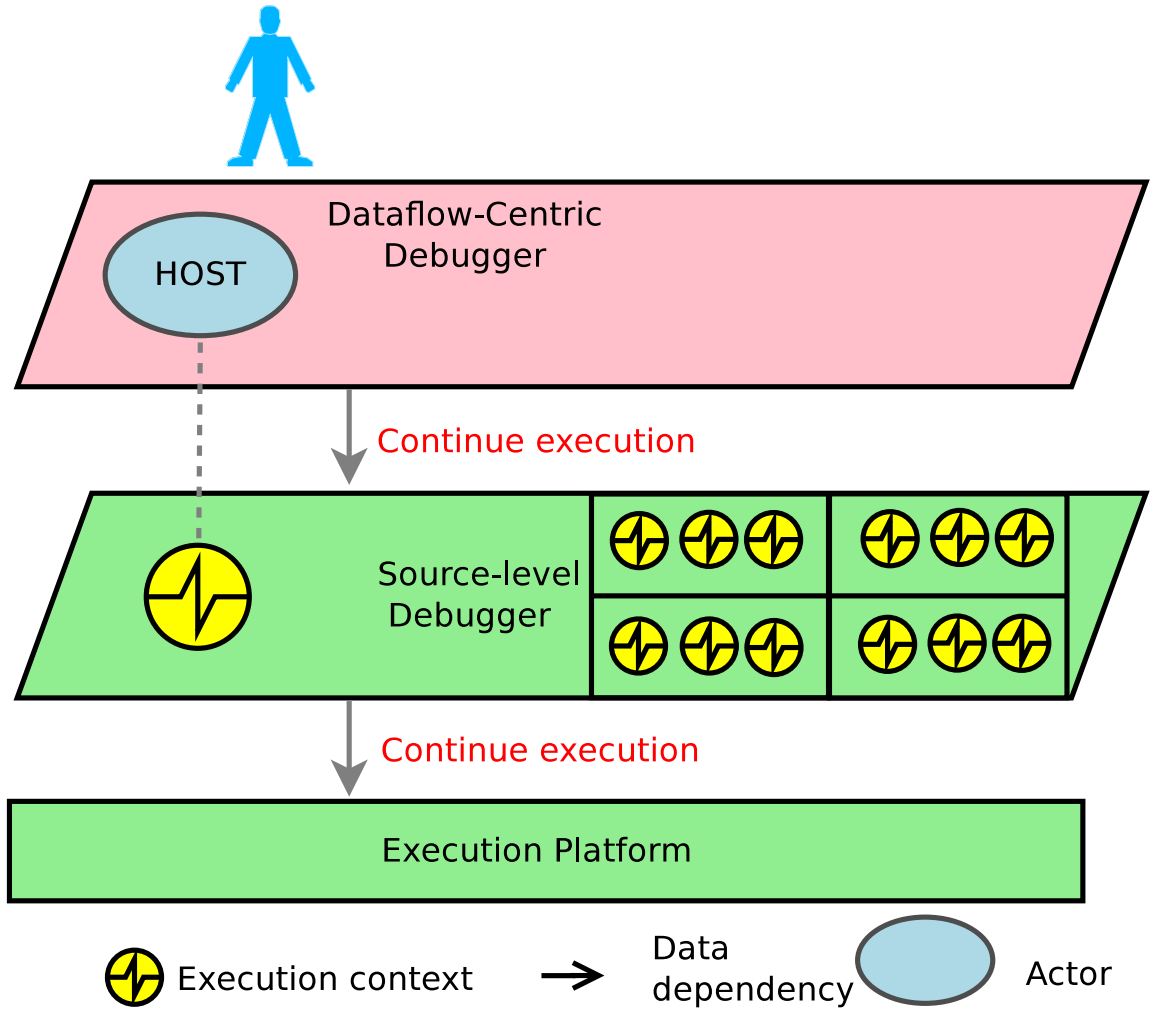
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



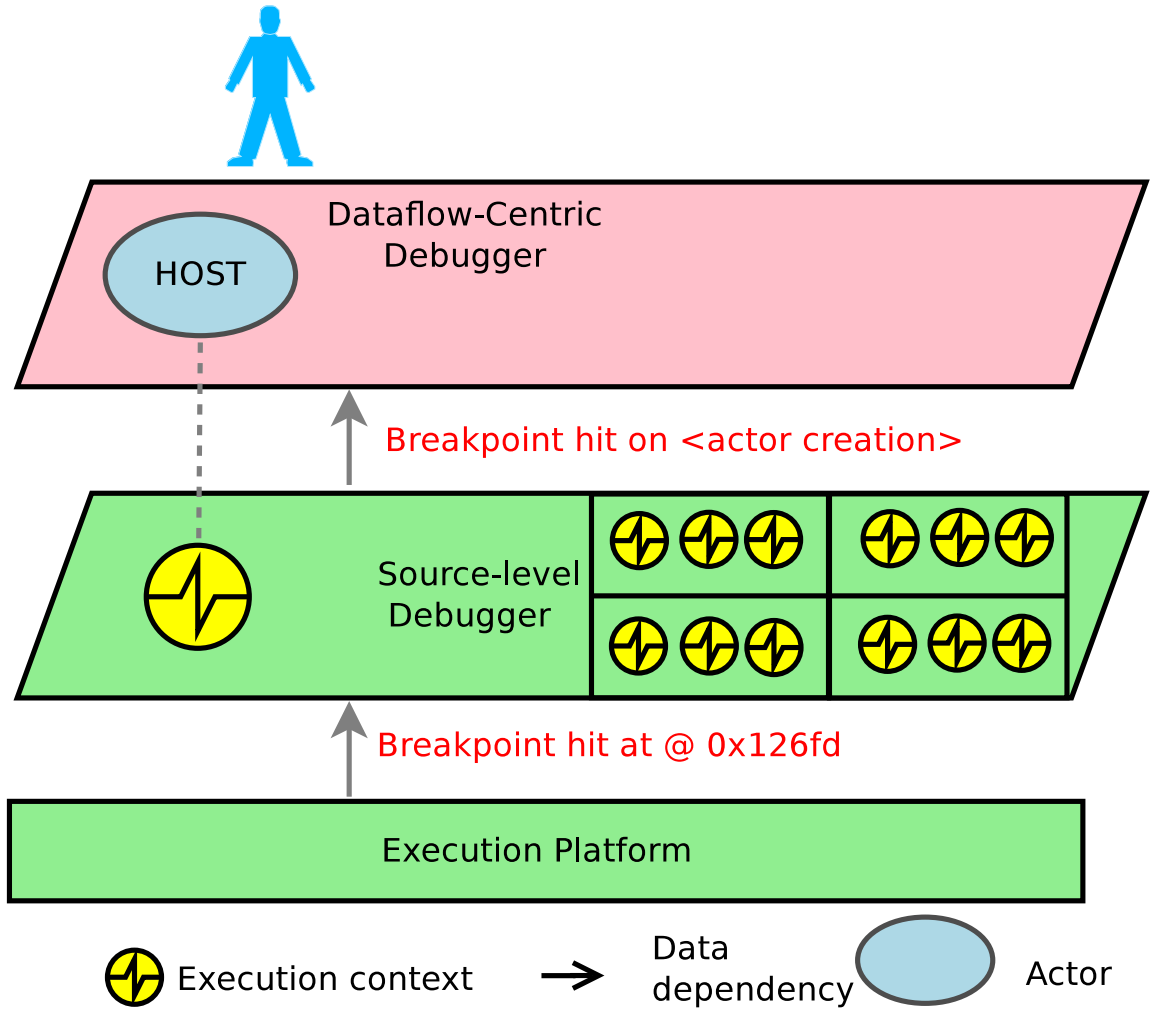
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



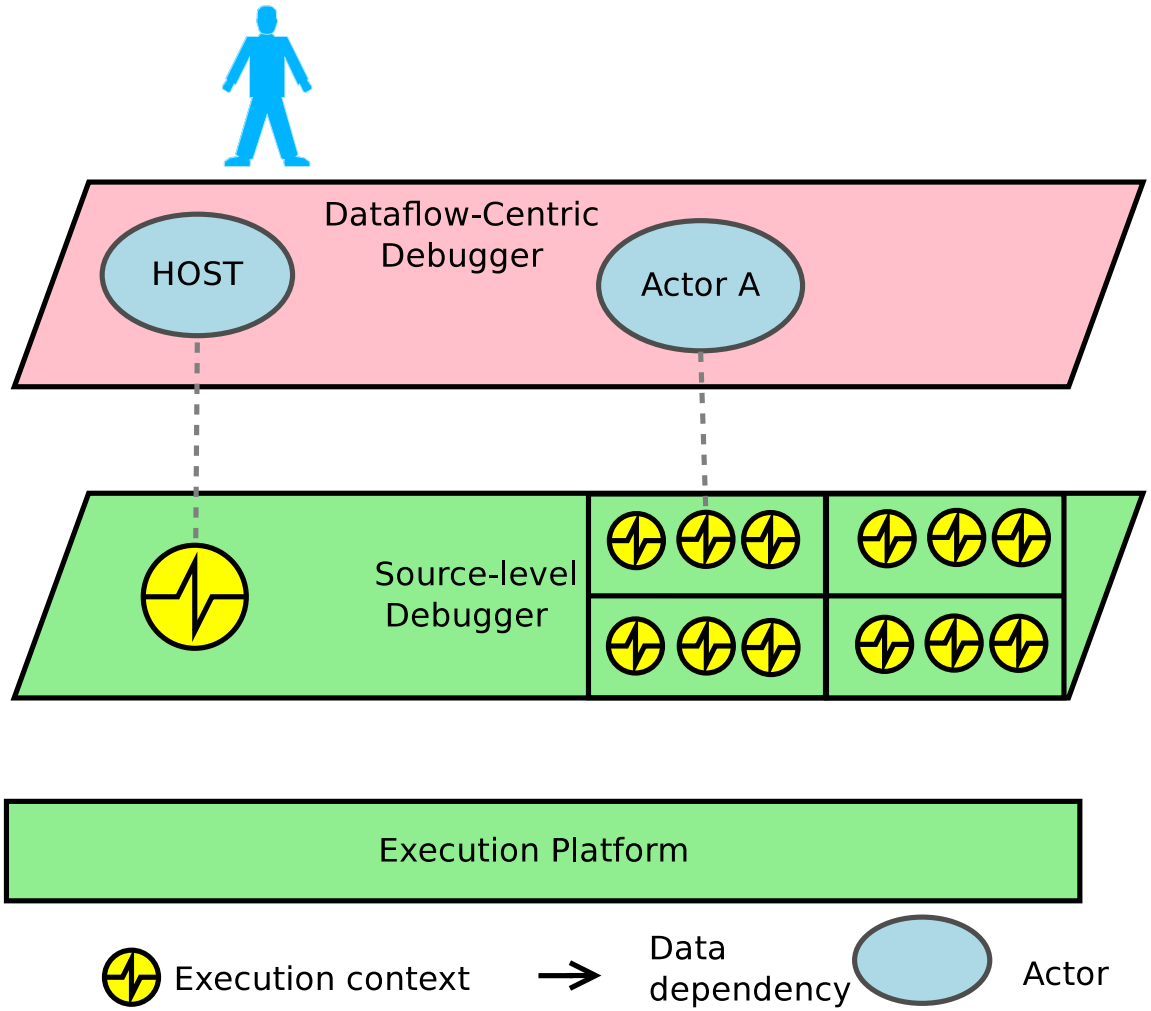
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



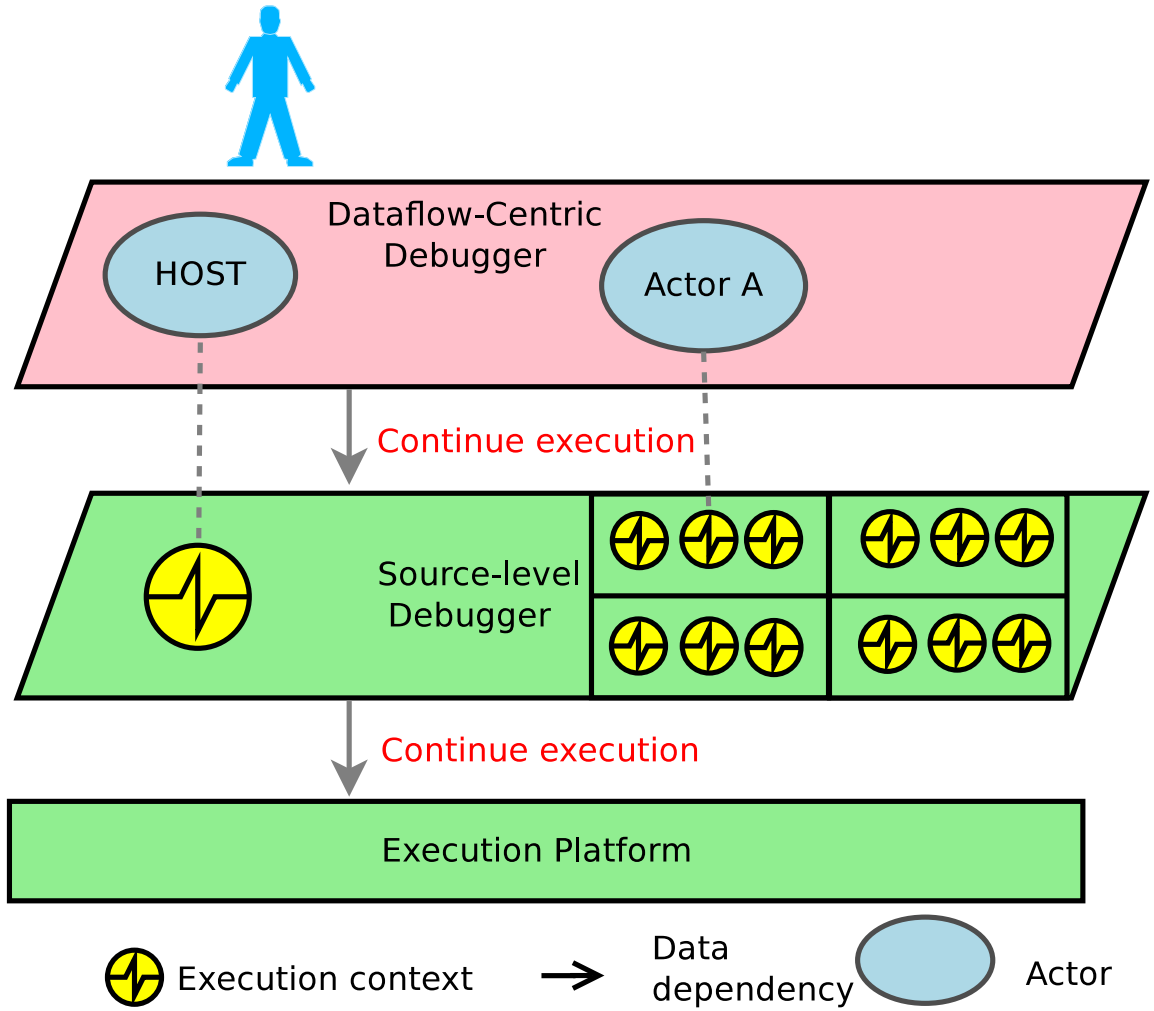
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



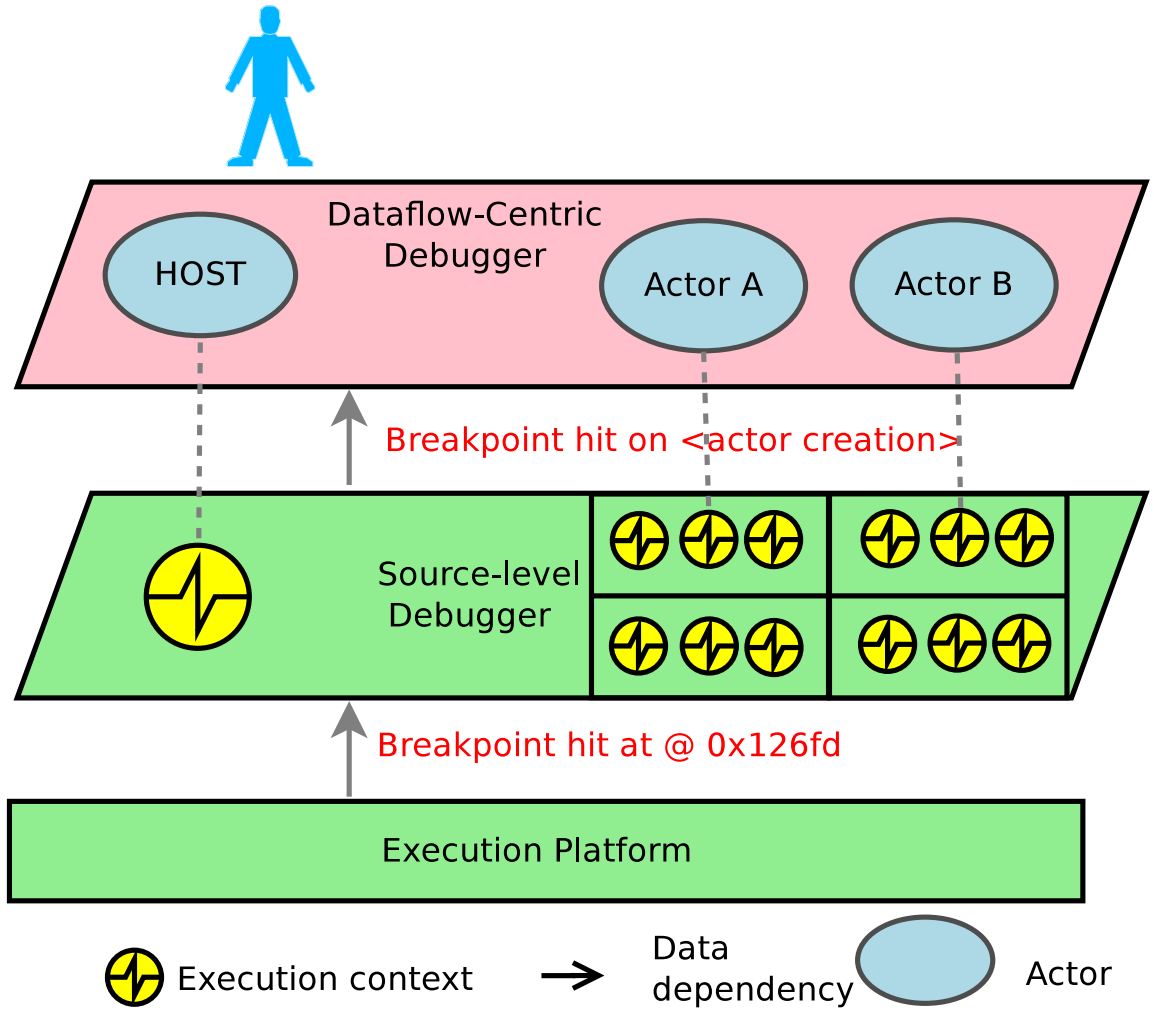
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



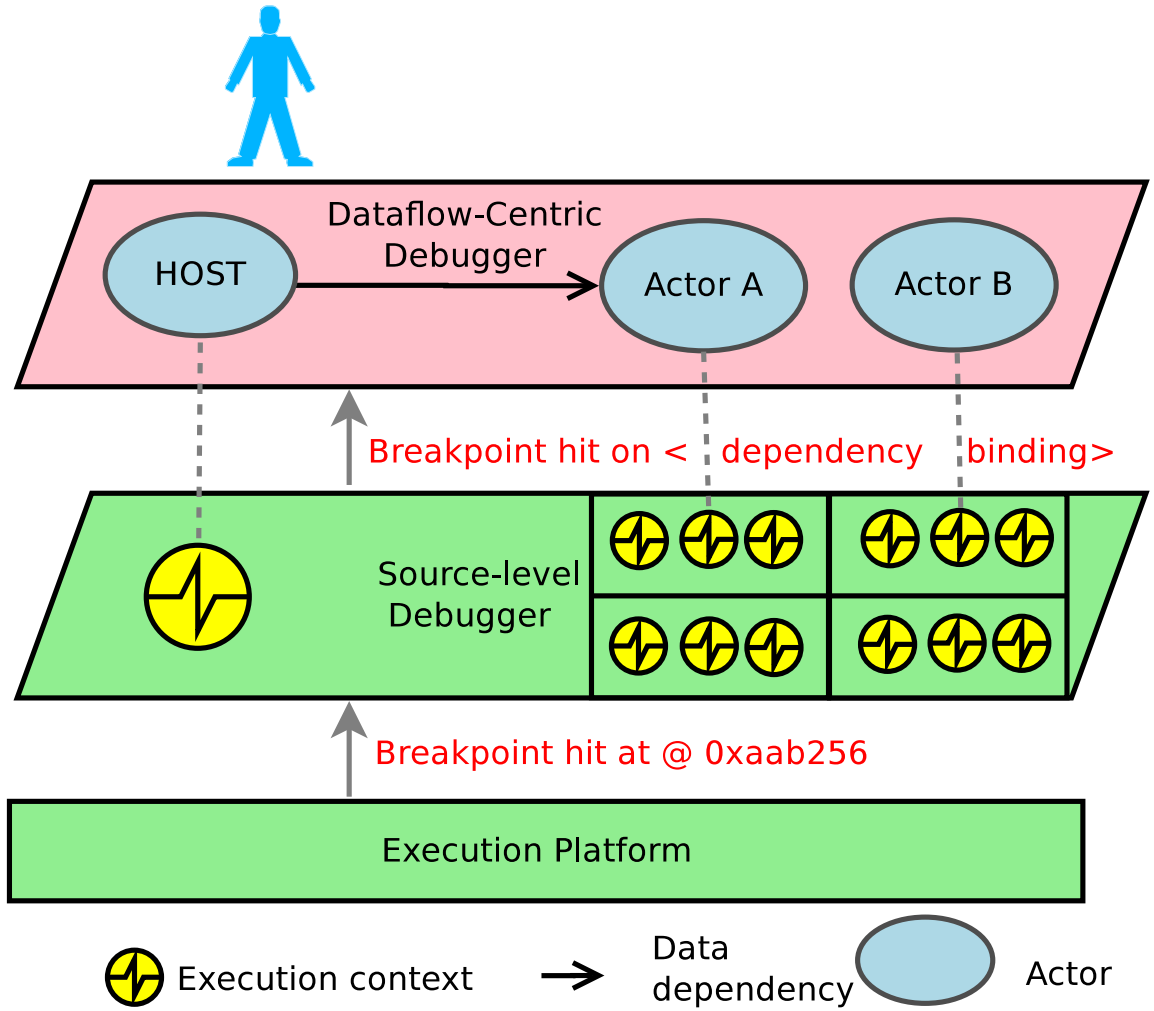
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



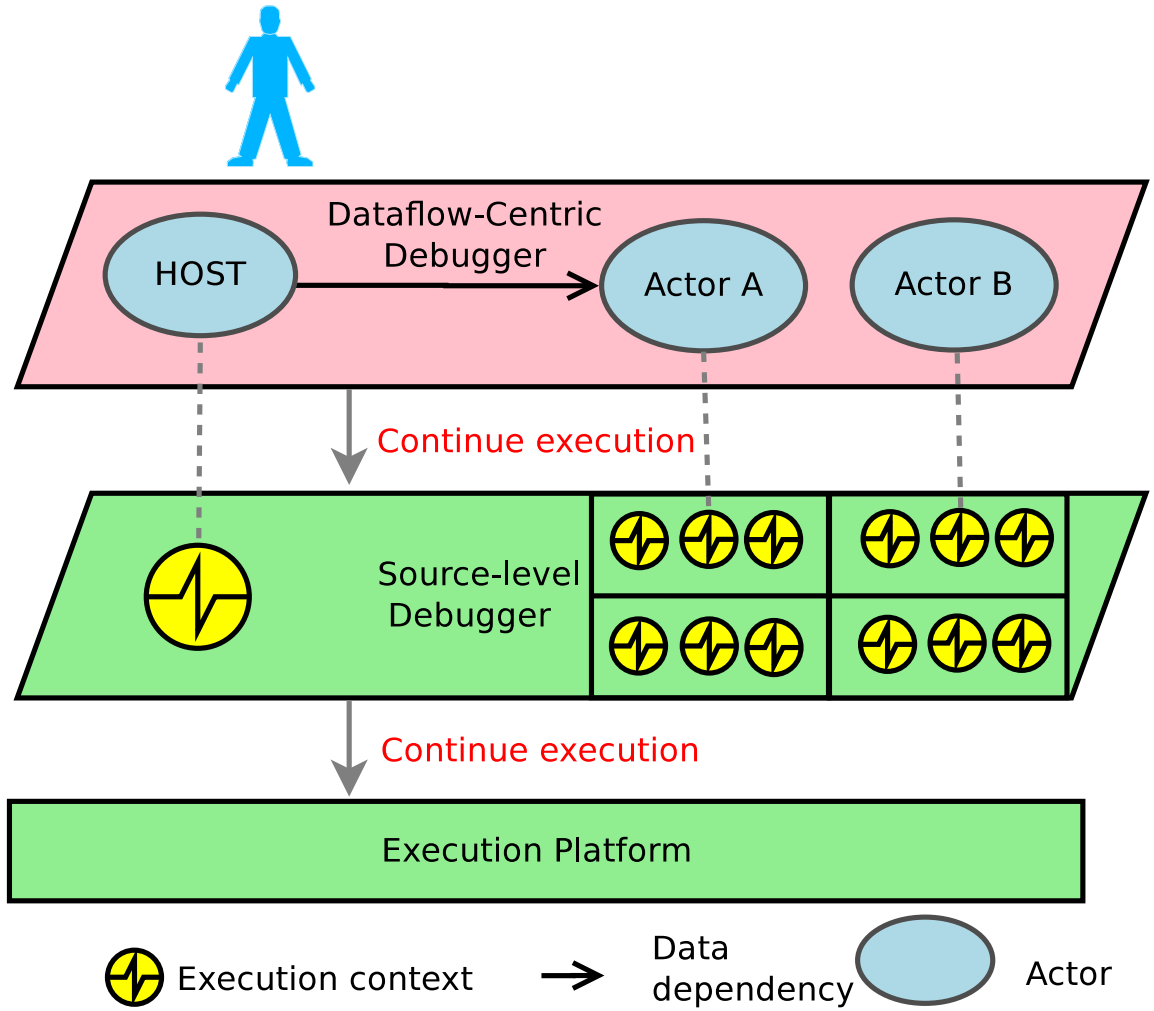
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



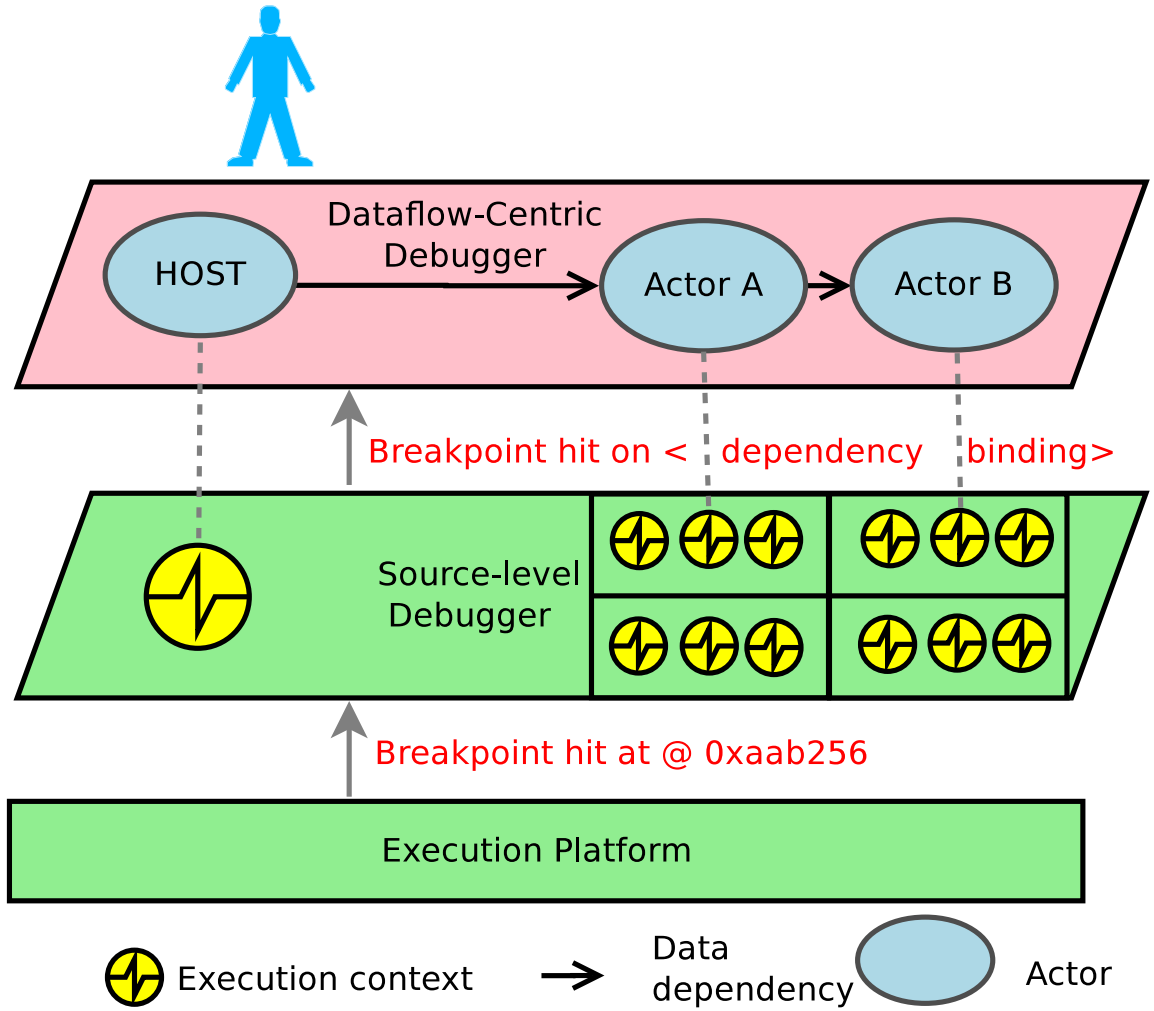
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



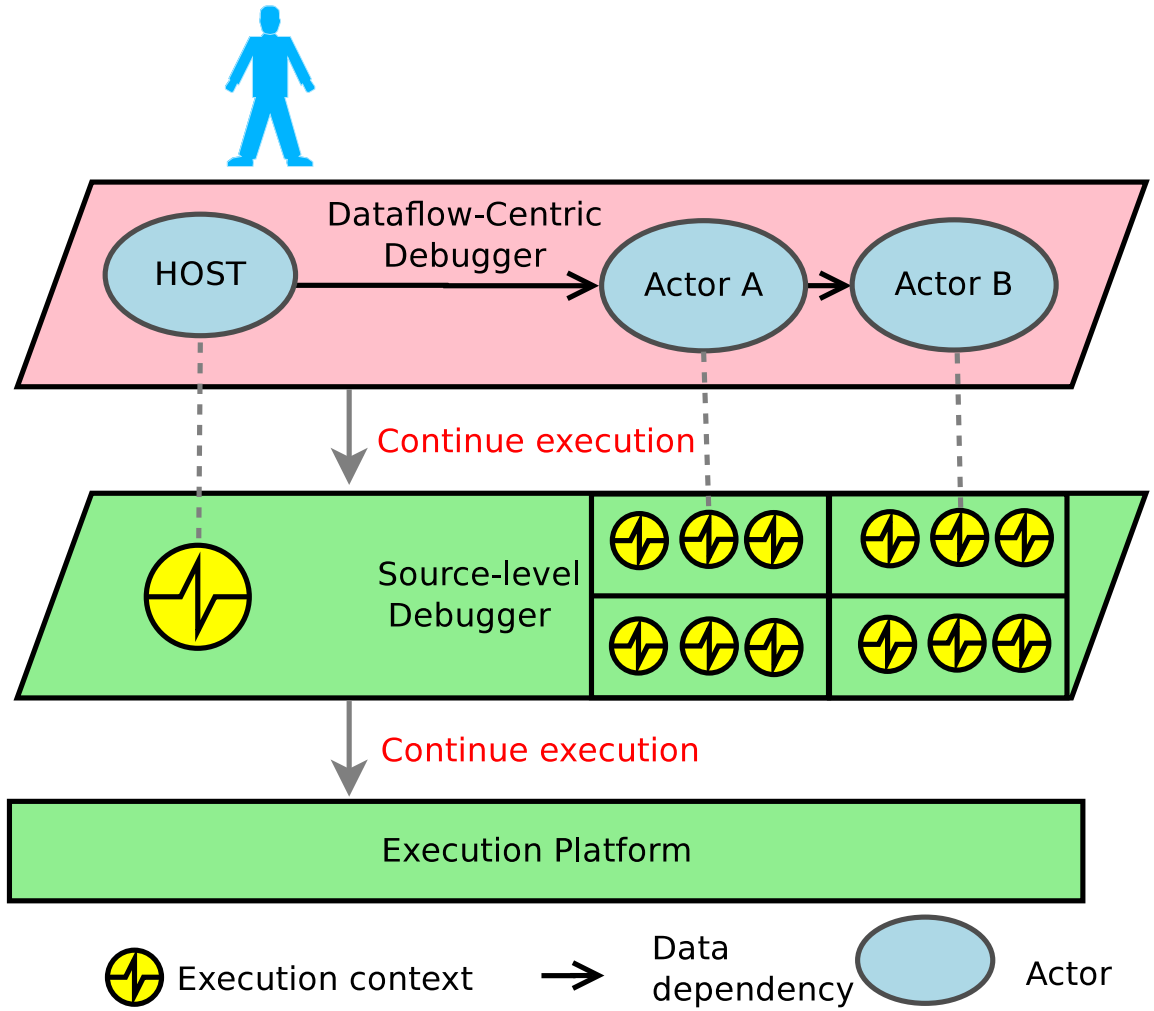
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



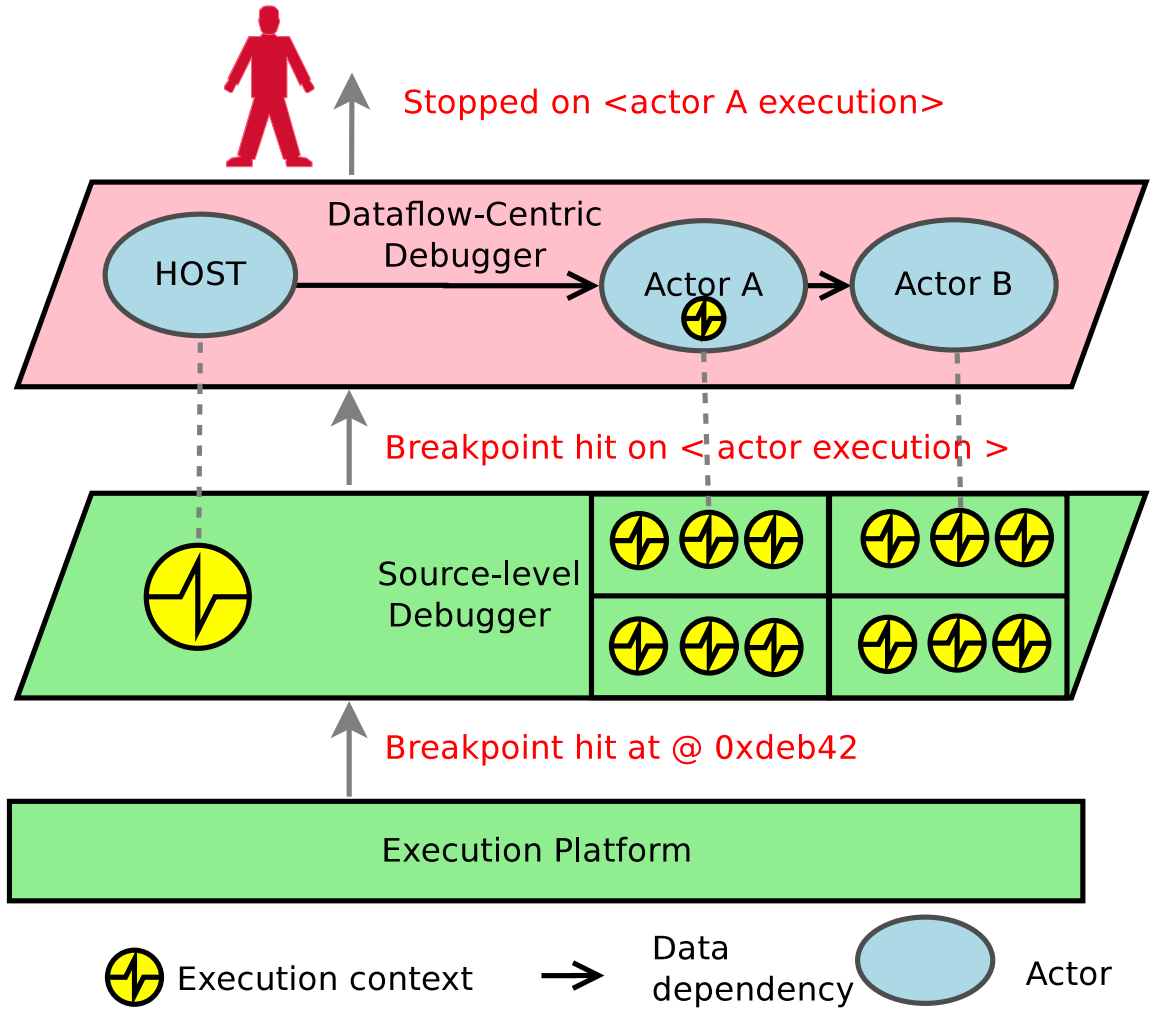
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



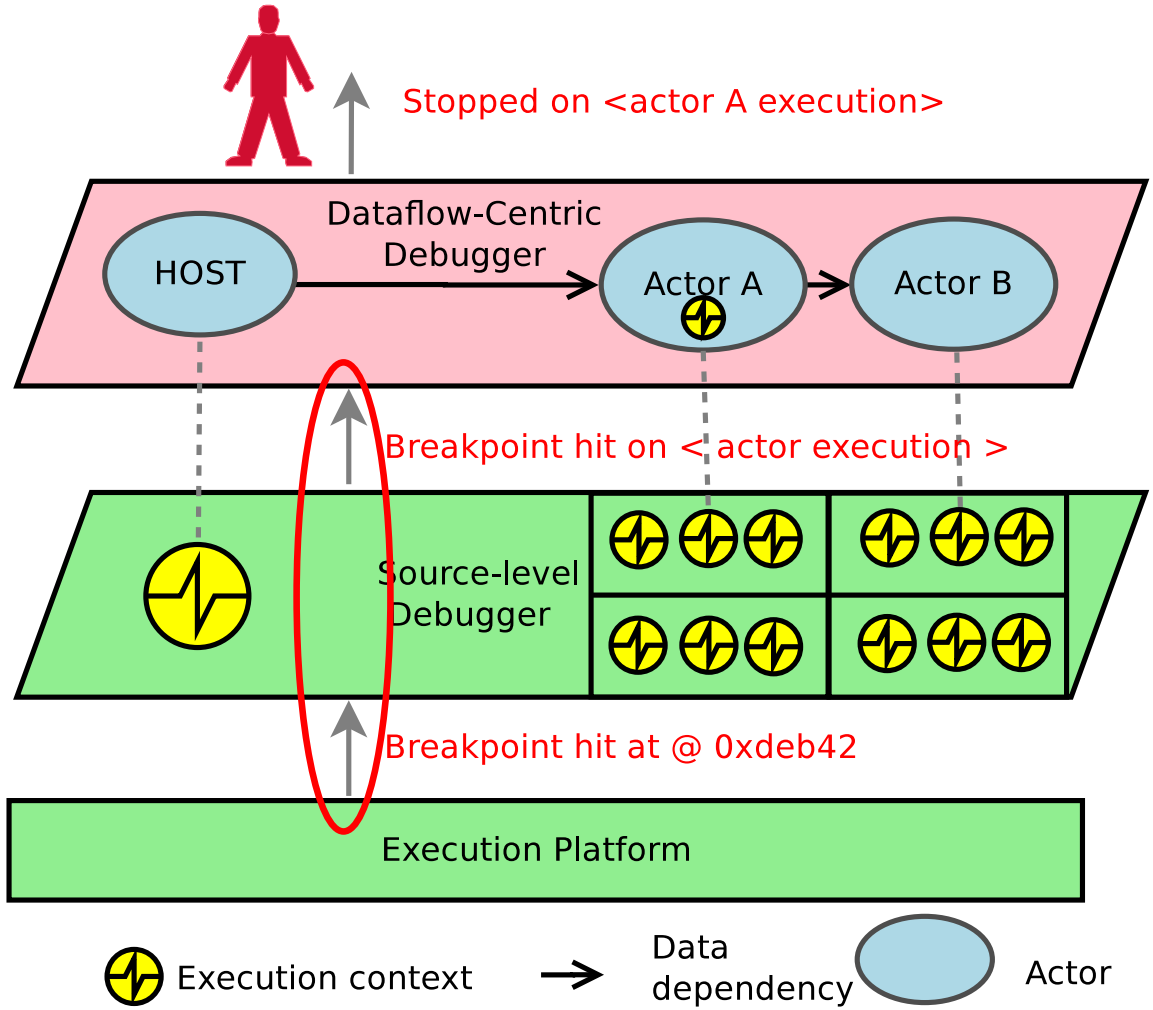
# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework

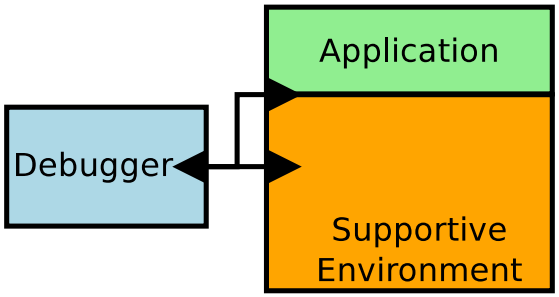


# Building Blocks of a Model-Centric Debugger

⇒ Detect and interpret the execution events of the runtime framework



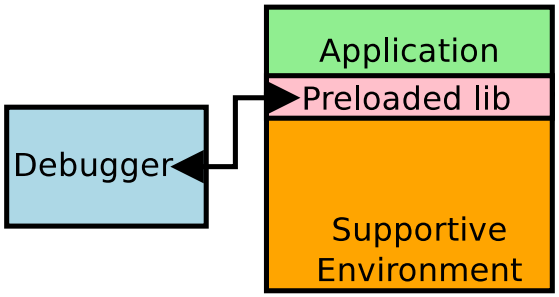
# Building Blocks of a Model-Centric Debugger




| | Breakpoints and Debug Information | | |
|---|---|---|---|
| Capturable Info. | **High** | | |
| Execution Overhead | Significant | | |
| Cooperation btw. Debug and Env. | **None** | | |
| Portability | Low | | |

# Building Blocks of a Model-Centric Debugger

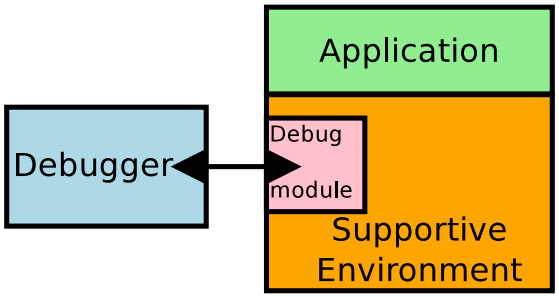
Compiler Optimization and Runtime SystEms




| | **Breakpoints and Debug Information** | **Preloaded Library** |
|---|---|---|
| Capturable Info. | **High** | Limited to API |
| Execution Overhead | Significant | Limited |
| Cooperation btw. Debug and Env. | **None** | Low |
| Portability | Low | **Very Good** |


# Building Blocks of a Model-Centric Debugger

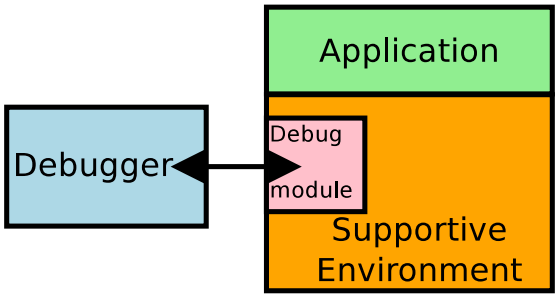
Compiler Optimization and Runtime SystEms




| | Breakpoints and Debug Information | Preloaded Library | Specialized Debug Module |
|---|---|---|---|
| Capturable Info. | **High** | Limited to API | **Full** |
| Execution Overhead | Significant | Limited | Limited |
| Cooperation btw. Debug and Env. | **None** | Low | Strong |
| Portability | Low | **Very Good** | Vendor Specific |


# Building Blocks of a Model-Centric Debugger

Compiler Optimization and Runtime SystEms



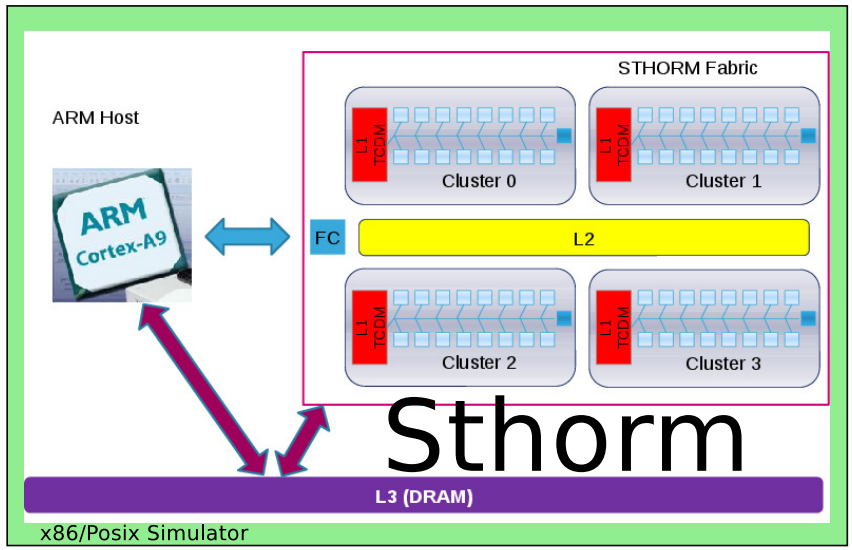

| | Breakpoints and Debug Information | Preloaded Library | Ayudame Debug Module |
|---|---|---|---|
| Capturable Info. | **High** | Limited to API | **Good** |
| Execution Overhead | Significant | Limited | Limited |
| Cooperation btw. Debug and Env. | **None** | Low | Moderate |
| Portability | Low | **Very Good** | Good+ Vendor |


# Agenda

Compiler Optimization and Runtime SystEms


# Case-Study Illustrations

Compiler Optimization and Runtime SystEms




## STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators

# Case-Study Illustrations

Compiler Optimization and Runtime SystEms

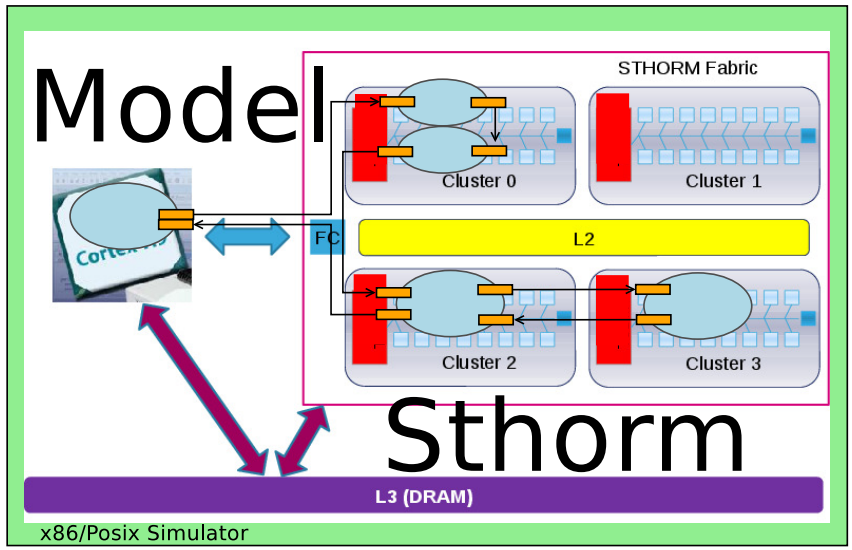
## STHORM Progr. Environments

- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

## STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



# Case-Study Illustrations

Compiler Optimization and Runtime SystEms

## The GNU Debugger

- Adapted to interactive debugging
- Large user community

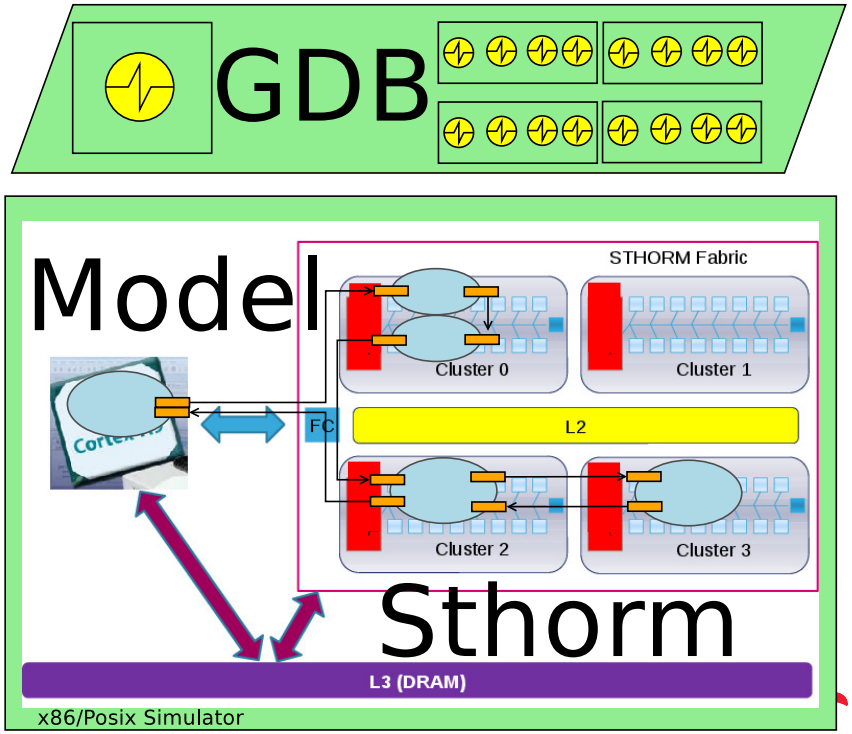
## STHORM Progr. Environments

- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

## STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators




# Case-Study Illustrations

Compiler Optimization and Runtime SystEms

## The GNU Debugger

- Adapted to interactive debugging
- Large user community
- Extendable with Python API

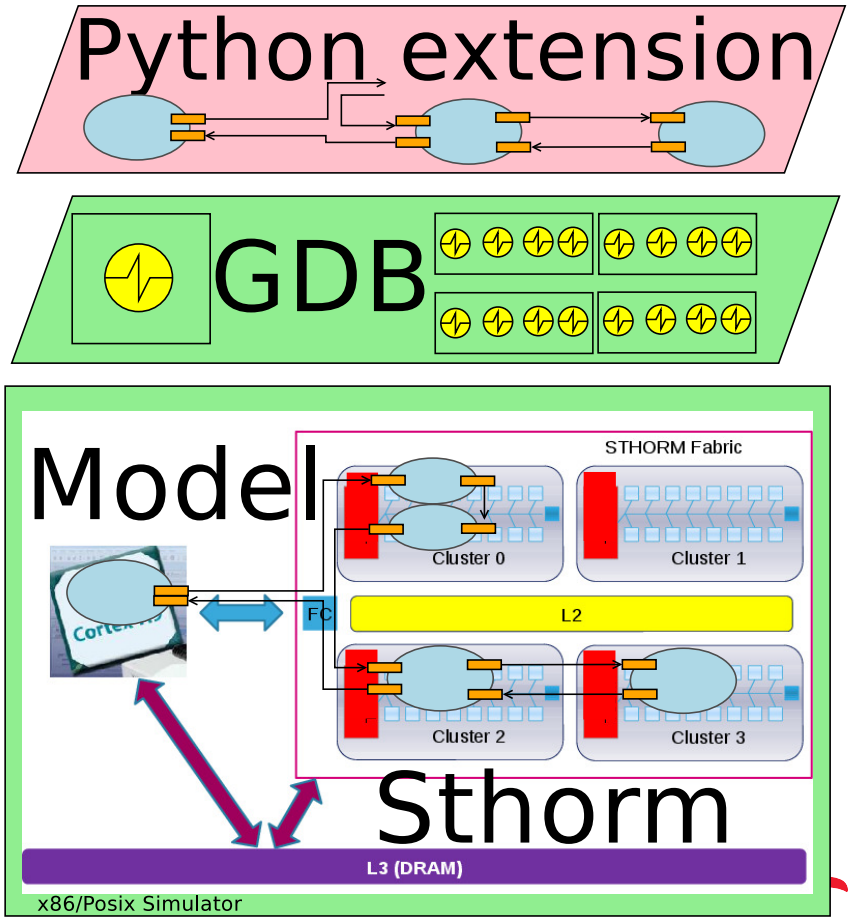
## STHORM Progr. Environments

- Dataflow (PEDF)
- Components (NPM)
- Kernels (OpenCL)

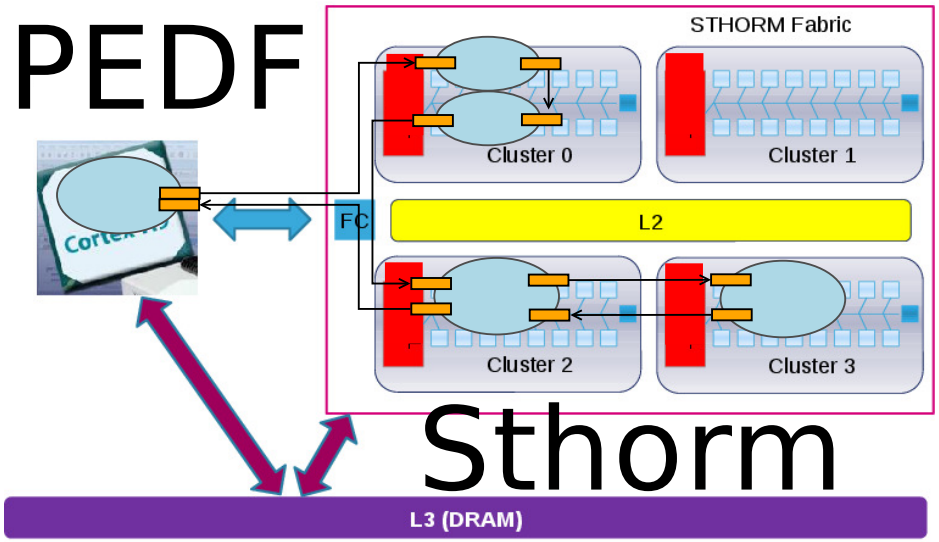
## STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



# Case-Study Illustrations: Dataflow Video Decoder

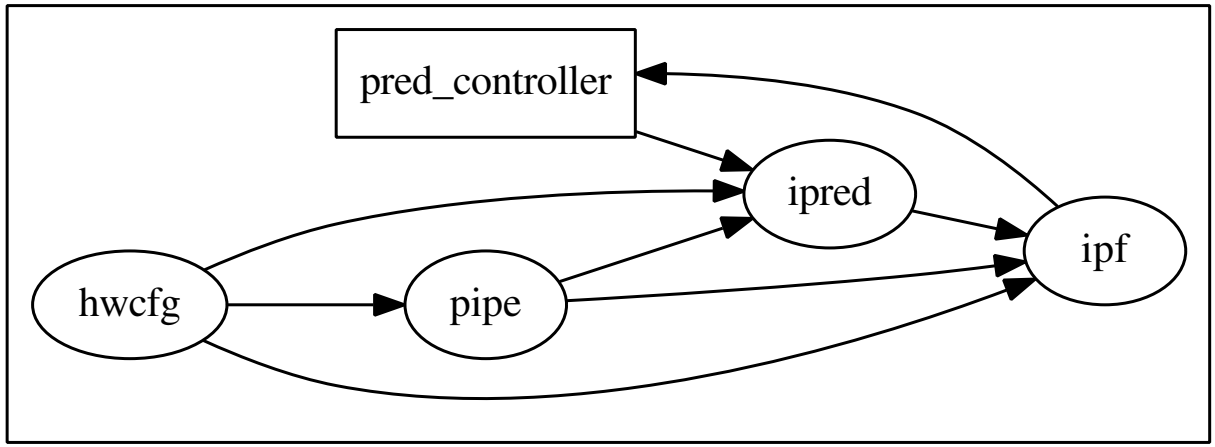




logo by bullboykennels

## The application is frozen, how can GDB help us?

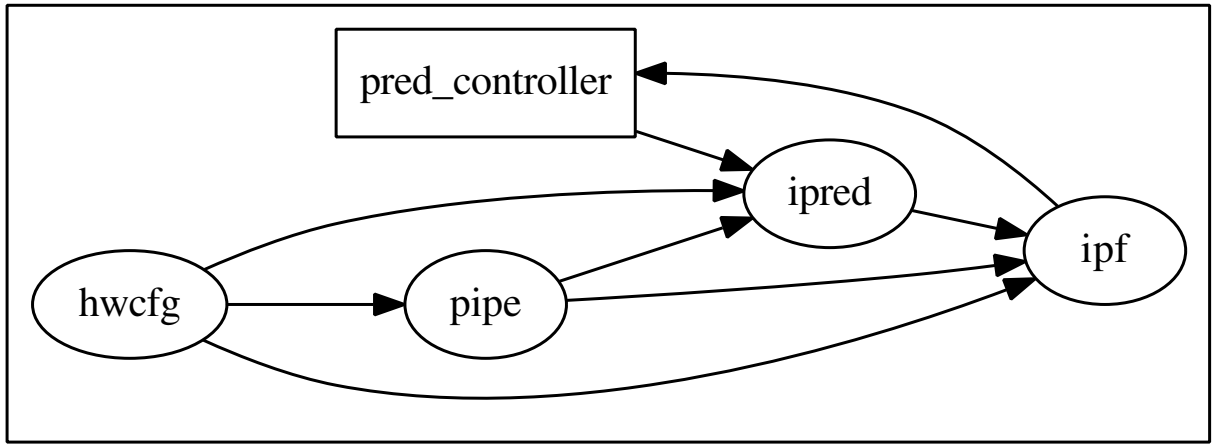
*hint: not much!*




(static graph provided by the compiler)

# Case-Study Illustrations: Dataflow Video Decoder

## The application is frozen, how can GDB help us?

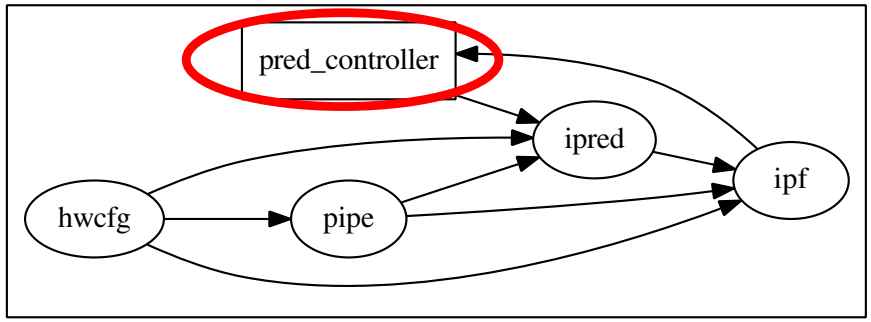



### (gdb) info threads

```
 Id   Target Id         Frame
  1   Thread 0xf7e77b  0xf7ffd430 in __kernel_vsyscall ()
* 2   Thread 0xf7e797  operator= (val=..., this=0xa0a1330)
```

# Case-Study Illustrations: Dataflow Video Decoder

## The application is frozen, how can GDB help us?

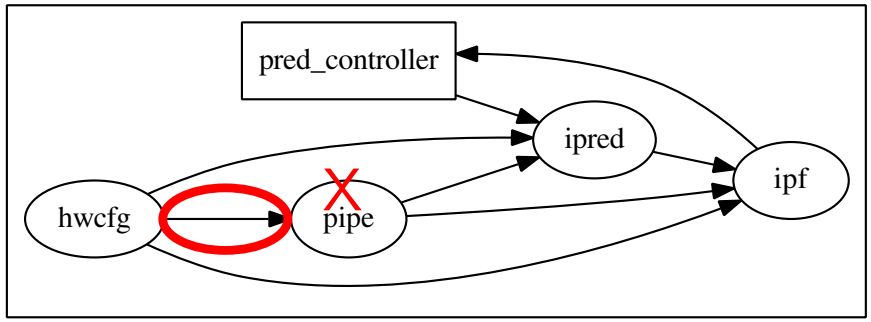



## (gdb) thread apply all where

```
Thread 1 (Thread 0xf7e77b):
#0  0xf7ffd430 in __kernel_vsyscall ()
#1  0xf7fcd18c in pthread_cond_wait@ ()
#2  0x0809748f in wait_for_step_completion(struct... *)
#3  0x0809596e in pred_controller_work_function()
#4  0x08095cbc in entry(int, char**) ()
#5  0x0809740a in host_launcher_entry_point ()
```

# Case-Study Illustrations: Dataflow Video Decoder

## The application is frozen, how can GDB help us?



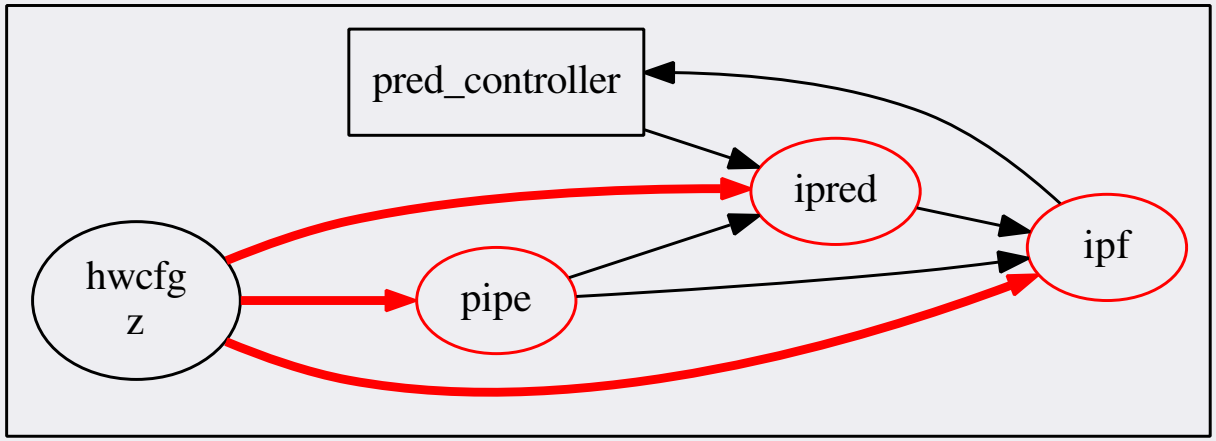

### (gdb) thread apply all where

```
Thread 2 (Thread 0xf7e797):
#0 operator= (val=..., this=0xa0a1330)
#1 pipeRead (data=0) at pipeFilter.c:154
                154   Smb = pedf.io.hwcfgSmb[count];
#2 0x0804da63 in PipeFilter_work_function () at pipe.c:361
#3 0x080a4132 in PedfBaseFilter::controller (this=0xa0d18)
#4 0x080c12f0 in sc_core::sc_thread_cor_fn (arg=0xa0a3598)
```

# Case-Study Illustrations: Dataflow Video Decoder

## The application is frozen, how can mcGDB help us?

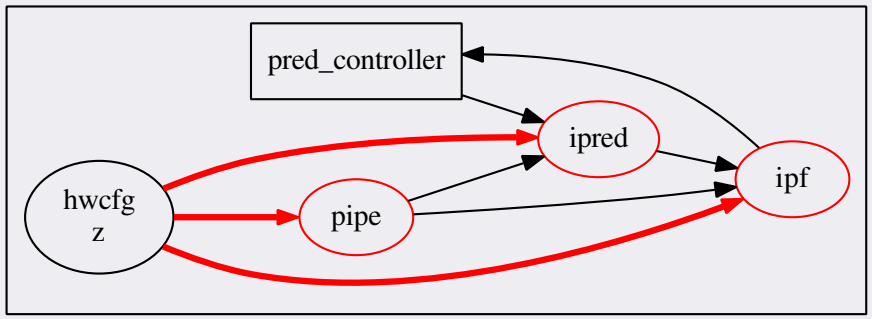
### (mcgdb) info graph



# Case-Study Illustrations: Dataflow Video Decoder

**The application is frozen, how can mcGDB help us?**

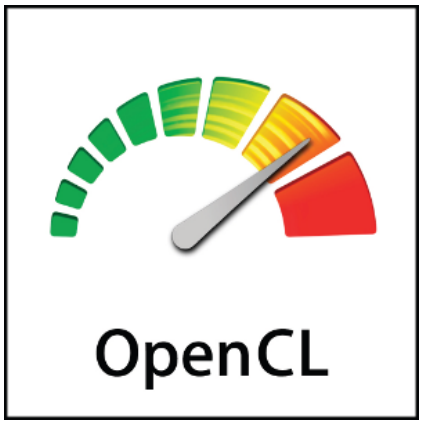
## (mcgdb) info graph

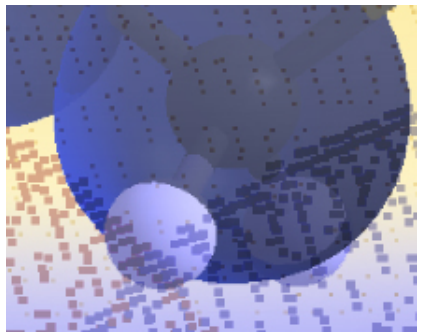



## (mcgdb) info actors +state

```
#0 Controller 'pred_controller':
    Blocked, waiting for step completion
#1/2/3 Actor 'pipe/ipref/ipf':
    Blocked, reading from #4 'hwcfg'
#4 Actor 'hwcfg':
    Asleep, Step completed
```

# Case-Study Illustrations: OpenCL Kernel Programming



**BigDFT**

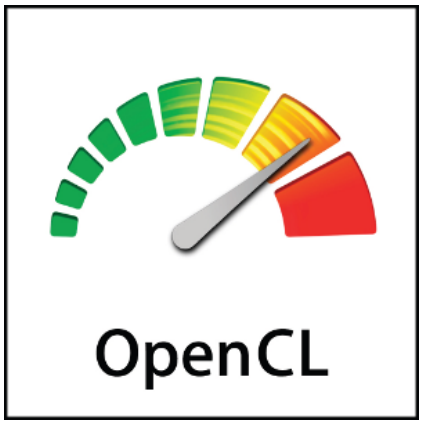
## OpenCL (and Cuda)

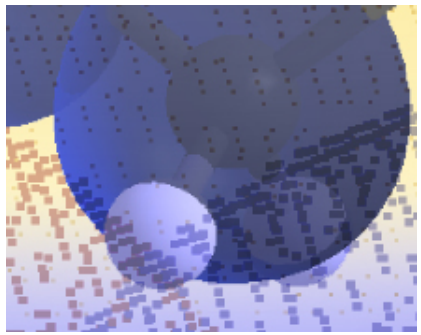
- Running on STHORM, but primarily used with GPU

Density functional theory solver.

- High performance computing
- Hybrid CPU/GPU
- MPI - OpenCL (Fortran / C)

# Case-Study Illustrations: OpenCL Kernel Programming



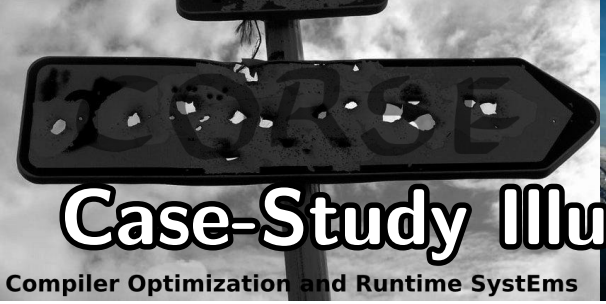
**BigDFT**

## OpenCL (and Cuda)

- Running on STHORM, but primarily used with GPU
- Host-side debugging only

Density functional theory solver.

- High performance computing
- Hybrid CPU/GPU
- MPI - OpenCL (Fortran / C)




# Case-Study Illustrations: OpenCL Kernel Programming

Compiler Optimization and Runtime SystEms

Why Execution Visualization ?

let's consider an example ...


## C code

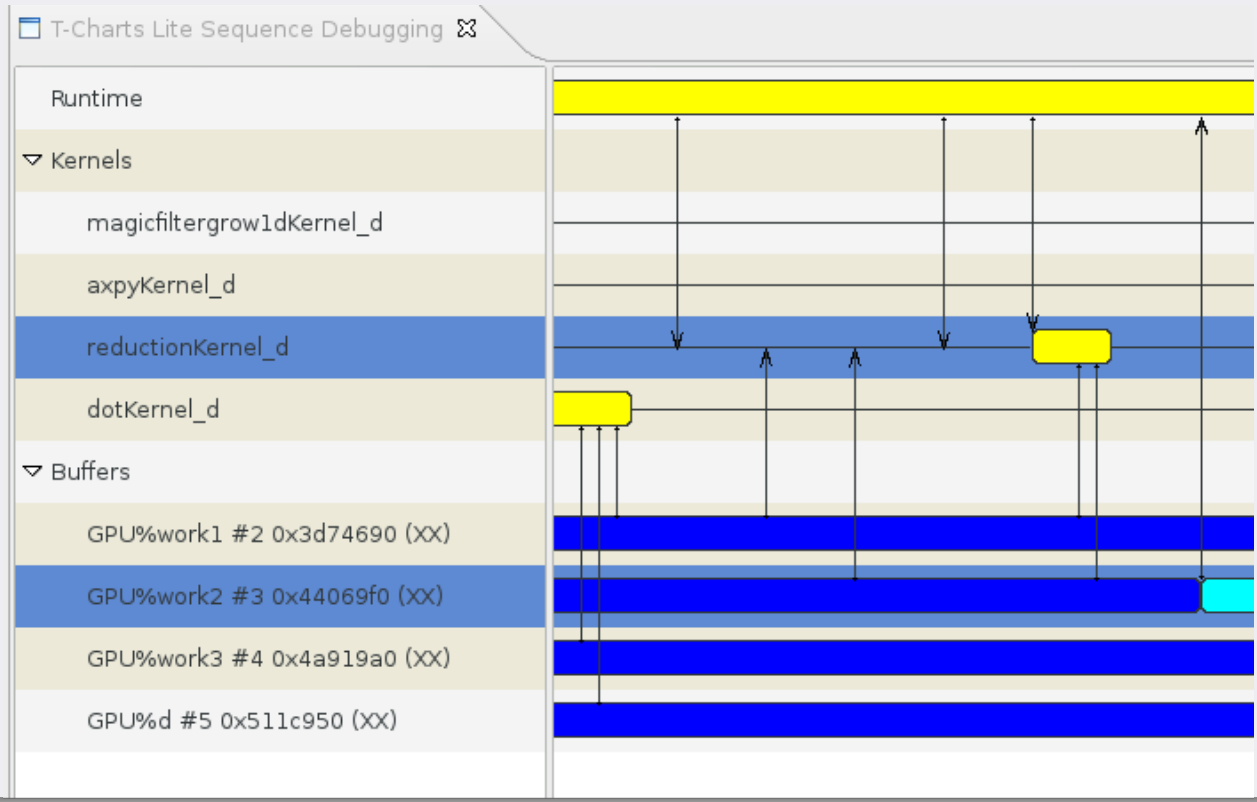
```
reductionKernel (int n, double *in, double *out){...}
checkStatus(int *ptr, char *msg) { if(ptr == 0) exit(-1);}

void main()  {
 double *in = malloc(...) ; checkStatus(in,"in failed");
 double *out = malloc(...); checkStatus(out,"out failed");

 initialize(in);
 reductionKernel(N, in, out);
 // free ...
}
```


## OpenCL equivalent:

```
/* Instantiate the runtime.  */
command_queue = clCreateCommandQueue((*context)->context, aDevices[0], 0, &ciErrNum);
kerns->reduction_kernel_d=clCreateKernel(reductionProgram, "reductionKernel_d",&ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create kernel!");
/* Allocate the buffers on the GPU.   */
*buff_ptr = clCreateBuffer((*context)->context,  CL_MEM_READ_ONLY, *size, NULL, &ciErrNum);
oclErrorCheck(ciErrNum,"Failed to create read buffer!");
/* Push the initial values to the GPU memory.  */
cl_int ciErrNum = clEnqueueWriteBuffer((*command_queue)->command_queue,  *buffer, CL_TRUE, 0, *size, p...
oclErrorCheck(ciErrNum,"Failed to enqueue write buffer!");
/* Set the kernel parameters.  */
clSetKernelArg(kernel, i++,sizeof(*ndat), (void*)ndat); clSetKernelArg(kernel, i++,sizeof(*in), (void*...
clSetKernelArg(kernel, i++,sizeof(*out), (void*)out);   clSetKernelArg(kernel, i++,sizeof(cl_dbl)*blk_...
/* Trigger the kernel execution.  */
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_queue, kernel, 1, NULL, globalWorkSz, localWo...
oclErrorCheck(errNum,"Failed to enqueue reduction kernel!");
/* Get the result back.  */
cl_int ciErrNum = clEnqueueReadBuffer((*command_queue)->command_queue, *input, CL_TRUE, 0, sizeof(cl_d...
oclErrorCheck(ciErrNum,"Failed to enqueue read buffer!");
/* Then release the memory ... */
```

# Case-Study Illustrations: OpenCL Kernel Programming

## (mcgdb) print_flow (an Eclipse visualization engine)

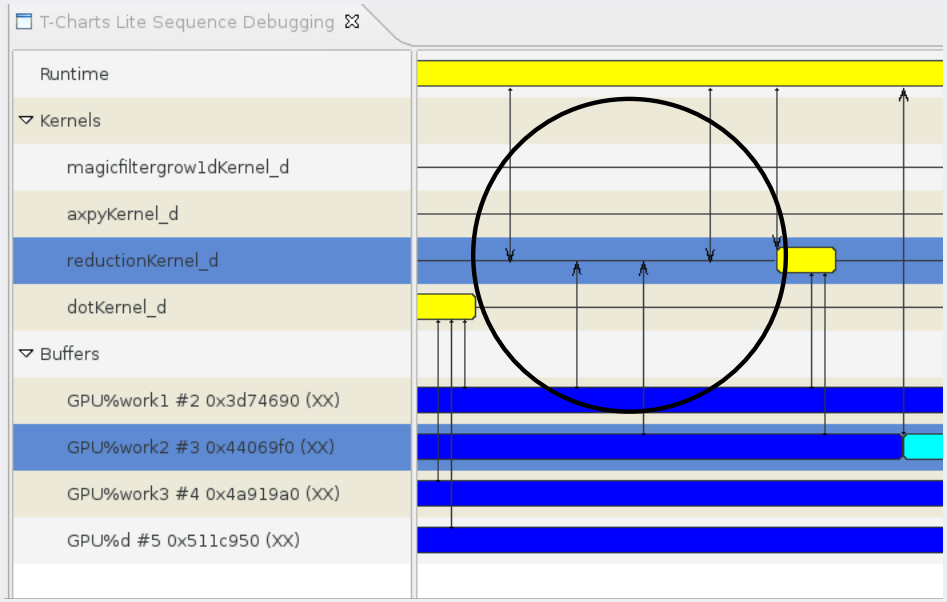



- updated on user request, or
- automatically on execution stops, step-by-step, ...

# Case-Study Illustrations: OpenCL Kernel Programming

## (mcgdb) print_flow

(an Eclipse visualization engine)



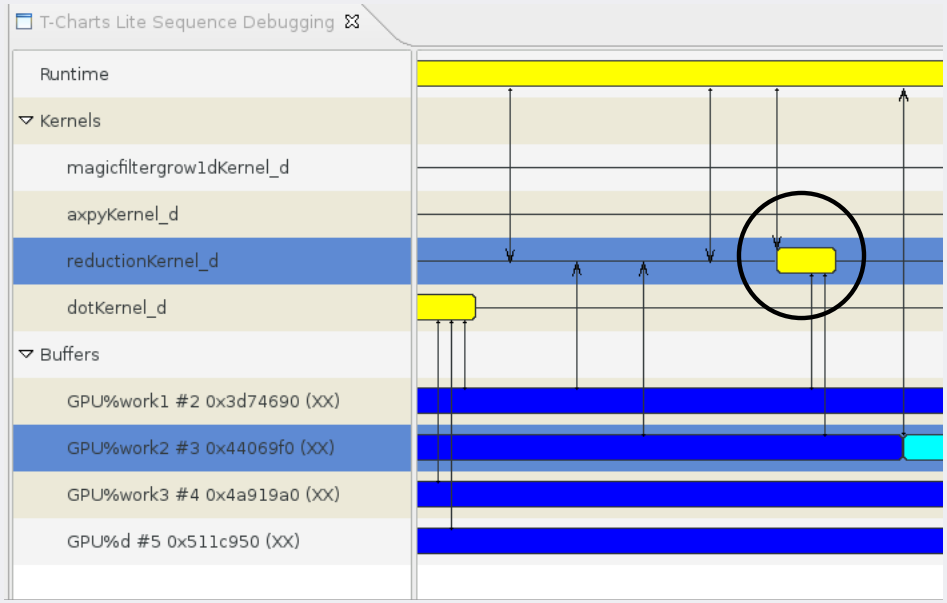

- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers

```
clSetKernelArg(kernel, i++, sizeof(*ndat),(void*)ndat);
clSetKernelArg(kernel, i++, sizeof(*in), (void*)in);
clSetKernelArg(kernel, i++, sizeof(*out), (void*)out);
clSetKernelArg(kernel, i++, sizeof(*sz), (void*)sz);
```

# Case-Study Illustrations: OpenCL Kernel Programming

## (mcgdb) print_flow

(an Eclipse visualization engine)



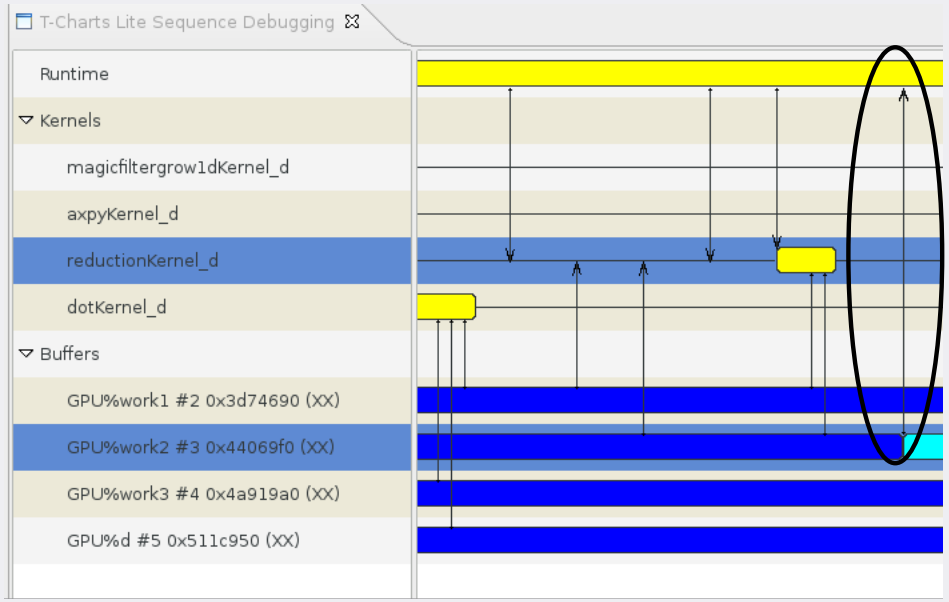

- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved

```
ciErrNum = clEnqueueNDRangeKernel(command_queue->command_q,
                                  kernel, 1, NULL, globalWorkSz,
                                  localWorkSize, 0, NULL, NULL);
```

# Case-Study Illustrations: OpenCL Kernel Programming

## (mcgdb) print_flow

(an Eclipse visualization engine)




- Set the kernel arguments.
  - 2 scalars
  - 2 GPU buffers
- Trigger the kernel execution
  - 2 buffers involved
- Retrieve the result
  - buffer content is saved

```
cl_int ciErrNum = clEnqueueReadBuffer(
                    (*command_queue)->command_queue,
                    *input, CL_TRUE, 0, sizeof(cl_double),
                    out, 0, NULL, NULL);
```


## Ongoing Work

OpenMP and Temanejo


# Ongoing Work: OpenMP and Temanejo

Compiler Optimization and Runtime SystEms

## OpenMP and GDB

$\Rightarrow$ No high-level vision of the application by GDB

```
(gdb) list
17  /* <---- current thread is here ----> */
18  #pragma omp critical
19  {
20     printf("@%d Inside critical zone", id);
21  }
(gdb) next
@4 Inside critical zone
@2 Inside critical zone
20  printf("@%d Inside critical zone", id);
(gdb) # I wanted to be the first :‘-(
```

# Ongoing Work: OpenMP constructs

Compiler Optimization and Runtime SystEms

## Taking OpenMP constructs into account

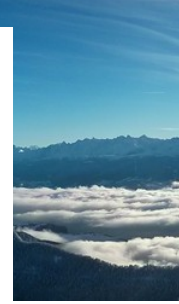
```
#pragma omp parallel
  {
    int id = omp_get_thread_num() + 1;

    #pragma omp single
    { ... }

    #pragma omp critical
    { ... }
  }
```

$\Rightarrow$ distinction of parallel zones, parallelism level, etc. in mcGDB

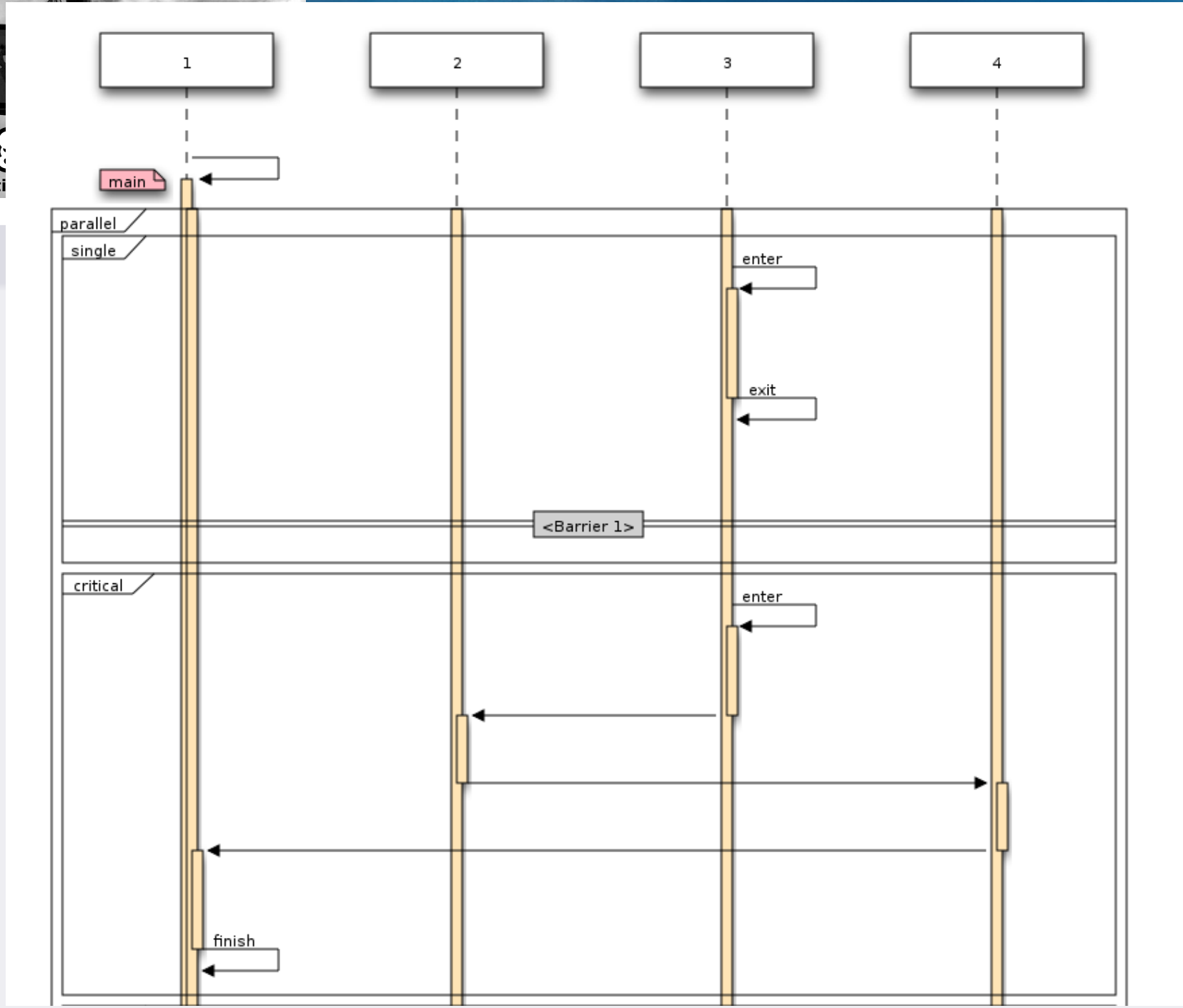

## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution


Ong
Compiler Opti



# Taki


1
2
3
4
main
parallel
single
enter
exit
<Barrier 1>
critical
enter
finish


## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution


## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution
- GUI integration into GDB ... (in progress)

```
(mcgdb) gui start
(mcgdb) gui control
(mcgdb) gui quit
```


## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution
- GUI integration into GDB ... (in progress)

```
(mcgdb) gui start
```

  - Qt-window popup controlled with Javascript
  - Auto-refresh on prompt display


# Ongoing Work: OpenMP constructs

Compiler Optimization and Runtime SystEms

## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution
- GUI integration into GDB ... (in progress)

```
(mcgdb) gui start
```

```
(mcgdb) gui control
```

  - Allows interactivity (= control of GDB) in the GUI
    - GDB is not thread-safe ⇒ CLI + GUI in a thread == segfault
  - Switch threads by clicking on the boxes
  - Stack-trace on mouse hover
  - ...


# Ongoing Work: OpenMP constructs

Compiler Optimization and Runtime SystEms

## Taking OpenMP constructs into account

- Sequence-diagram-like visualization of OpenMP execution
- GUI integration into GDB ... (in progress)

```
(mcgdb) gui start
```

```
(mcgdb) gui control
```

  - Allows interactivity (= control of GDB) in the GUI
    - GDB is not thread-safe ⇒ CLI + GUI in a thread == segfault
  - Switch threads by clicking on the boxes
  - Stack-trace on mouse hover
  - ...

- Extending the CLI with execution-control commands ...


# Ongoing Work: Execution Control Commands

Compiler Optimization and Runtime SystEms

## (mcgdb) omp start

Continues the execution until the beginning of the first parallel zone.


# Ongoing Work: Execution Control Commands

Compiler Optimization and Runtime SystEms

## (mcgdb) omp start

Continues the execution until the beginning of the first parallel zone.

## (mcgdb) omp next <zone>

Continues the execution until the next OpenMP <zone>.

zone $\in$ {`single`, `critical`, `task`, `section`, `barrier`, `master`}


# Ongoing Work: Execution Control Commands

Compiler Optimization and Runtime SystEms

## (mcgdb) omp start

Continues the execution until the beginning of the first parallel zone.

## (mcgdb) omp next <zone>

Continues the execution until the next OpenMP <zone>.

`zone` $\in$ {`single, critical, task, section, barrier, master`}

## (mcgdb) omp step

Continues the exec. until one thread starts working on the current zone.


# Ongoing Work: Execution Control Commands

Compiler Optimization and Runtime SystEms

## (mcgdb) omp start

Continues the execution until the beginning of the first parallel zone.

## (mcgdb) omp next <zone>

Continues the execution until the next OpenMP <zone>.

`zone` $\in$ {`single, critical, task, section, barrier, master`}

## (mcgdb) omp step

Continues the exec. until one thread starts working on the current zone.

## (mcgdb) omp all_out

Continues the exec. until all the threads are right after the current zone.


# Ongoing Work: OpenMP and Temanejo

Compiler Optimization and Runtime SystEms

## Next step?

OpenMP 4.0 task dependencies


## Next step?

OpenMP 4.0 task dependencies
and cooperation with Temanejo!

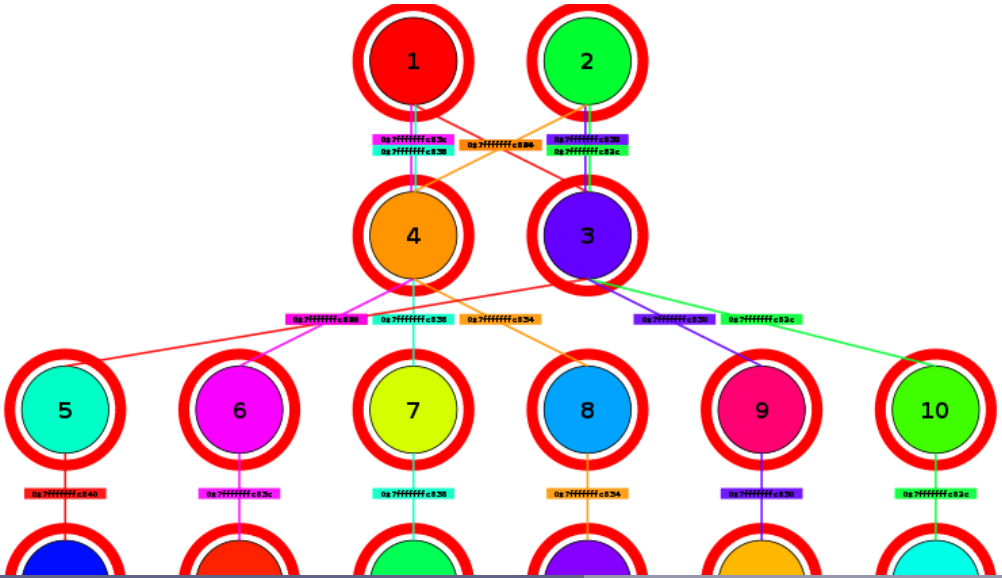

# Ongoing Work: OpenMP and Temanejo

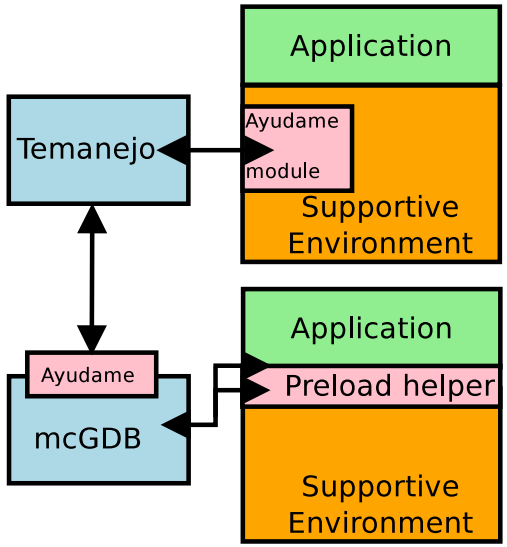
Compiler Optimization and Runtime SystEms

## Next step?

OpenMP 4.0 task dependencies
and cooperation with Temanejo!

- mcGDB feeds Temanejo with task graph
- Temanejo provides the task graph visualization and model UI
- mcGDB and GDB provides model and source user interaction




# Conclusions and Future Work


# Conclusions and Future Work



- Debugging high-level applications is challenging
- Lack of information about programming models and frameworks

**Our contribution:** model-centric interactive debugging (PhD Thesis'14), applied to:

- Component-software engineering (SCOPES '12)
- Dataflow programming (SAC and HIPS '13)
- Kernels for accelerator programming
- OpenMP on its way


# Conclusions and Future Work



- Debugging high-level applications is challenging
- Lack of information about programming models and frameworks

**Our contribution:** model-centric interactive debugging (PhD Thesis'14), applied to:

- Component-software engineering (SCOPES '12)
- Dataflow programming (SAC and HIPS '13)
- Kernels for accelerator programming
- OpenMP on its way

**Proof-of-concept:** MCGDB, a prototype for STHORM platform

- Extends GDB and its Python interface:
  - Framework for model-centric debugging
  - Interface patches contributed to the community
- Usage studied through embedded and scientific applications


# Conclusions and Future Work



**Perspectives** with programming-model centric debugging:

- Industrial side
  - Strengthen the implementation for production
  - Conduct extensive impact studies
  - Integrate within graphical debugging and visualization environments


# Conclusions and Future Work



**Perspectives** with programming-model centric debugging:

- Industrial side
  - Strengthen the implementation for production
  - Conduct extensive impact studies
  - Integrate within graphical debugging and visualization environments
- Research side
  - Apply to different programming models
    - ★ multi-level of abstraction for embedded systems,
    - ★ hardware (ARM big.LITTLE architecture)
  - Continue the study on visualization-assisted interactive debugging
  - Cooperation with Temanejo!

# Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget
Jean-François Méhaut, Miguel Santana

University Joseph Fourier / LIG, STMicroelectronics, Grenoble, France
Nano2017-DEMA project

HLRS Institute, Stuttgart, Germany
August 31$^{st}$, 2015


# Publications

Compiler Optimization and Runtime SystEms


Kevin Pouget.
*Programming-Model Centric Debugging for Multicore Embedded Systems*. PhD thesis, Université de Grenoble, École Doctorale MSTII, feb 2014.

Kevin Pouget, Marc Pérache, Patrick Carribault, and Hervé Jourdren.
User level DB: a debugging API for user-level thread libraries. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–7, 2010.

Kevin Pouget, Miguel Santana, Vania Marangozova-Martin, and Jean-François Mehaut.
Debugging Component-Based Embedded Applications. In *Joint Workshop Map2MPSoC (Mapping of Applications to MPSoCs) and SCOPES (Software and Compilers for Embedded Systems)*, St Goar, Germany, may 2012. Published in the ACM library.

Kevin Pouget, Patricia López Cueva, Miguel Santana, and Jean-François Méhaut.
Interactive Debugging of Dynamic Dataflow Embedded Applications. In *Proceedings of the 18th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, Boston, Massachusetts, USA, may 2013. Held in conjunction of IPDPS.

Kevin Pouget, Patricia López Cueva, Miguel Santana, and Jean-François Méhaut.
A novel approach for interactive debugging of dynamic dataflow embedded applications. In *Proceedings of the 28th Symposium On Applied Computing (SAC)*, pages 1547–1549, Coimbra, Portugal, apr 2013.


## (mcgdb) info workers

```
> Worker #1: ParallelJob #1 > CriticalJob #1
  Worker #2: ParallelJob #1 > Barrier #1
  Worker #3: ParallelJob #1
  Worker #4: ParallelJob #1 > Barrier #1
```


## (gdb) where

```
#0  #pragma   omp critical_start ()
#1  0x0400a1a in  ParallelJob #1::main<0> () at parallel-demo.
#3  #pragma   omp parallel ()
#5  0x4009cb  in  main () at parallel-demo.c:6
```


# Current-State Information

Compiler Optimization and Runtime SystEms

## (gdb) where

```
#0  #pragma   omp critical_start ()
#1  0x0400a1a in  ParallelJob #1::main<0> () at parallel-demo.
#3  #pragma   omp parallel ()
#5  0x4009cb  in  main () at parallel-demo.c:6
```

## (gdb) where no-filter

```
#0 GOMP_critical_start ()          at libgomp/critical.c:36
#1 0x0400a1a in main._omp_fn.0 () at parallel-demo.c:18
#2 0x7df94dc in GOMP_parallel_tramp () at omp_preload.c:125
#3 0x7bb4caf in GOMP_parallel ()  at libgomp/parallel.c:168
#4 0x7df953c in GOMP_parallel ()  at omp_preload.c:136
#5 0x04009cb in main ()           at parallel-demo.c:6
```


# Implementation Challenges

Compiler Optimization and Runtime SystEms

## Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!


## Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!
- GDB/Python is bad at switch-and-continuing threads:
  e.g., to stop after a barrier:
  - set a BP on barrier function
  - continue until (all the threads -1) hit the barrier
  - when the last thread arrives:
    - activate scheduler-locking (= run only one thread at a time)
    - for all the threads:
      switch to the thread
      continue until the end of the barrier function
- should work in theory, but too hacky in practice.

---

*a*


# Implementation Challenges

Compiler Optimization and Runtime SystEms

## Controling Parallel Threads is Hard ...

... and I never did it before !

- Dataflow, components, etc. are not SPMD/SIMD!
- GDB/Python is bad at switch-and-continuing threads:
  e.g., to stop after a barrier:
  - set a BP on barrier function
  - continue until (all the threads -1) hit the barrier
  - when the last thread arrives:
    - activate scheduler-locking (= run only one thread at a time)
    - for all the threads:
      switch to the thread ← forbidden ↓ :-( [a]
      continue until the end of the barrier function
- should work in theory, but too hacky in practice.

[a] *Thou shalt not alter any data within gdb or the inferior* (gdbdoc 23,2,2,20)

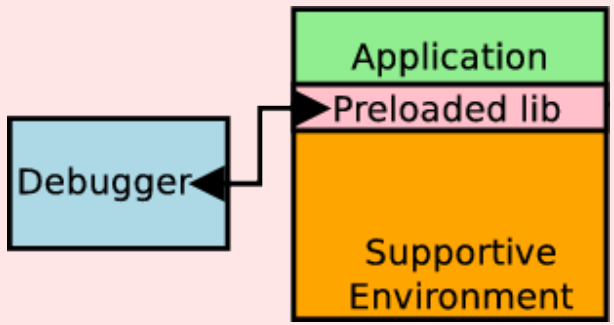
# Implementation Challenges

Compiler Optimization and Runtime SystEms

## Controling Parallel Threads is Hard ...

... and I never did it before !

## `libmcgdb_ldpreload_gomp.so` to the rescue!




- dynamically inserted btw app. and lib.
- transparent (mostly < gdb 7.8)
- tied to library implementation/ABI :-(


# Implementation Challenges

Compiler Optimization and Runtime SystEms

## Controling Parallel Threads is Hard ...

... and I never did it before !

libmcgdb_ldpreload_gomp.so to the rescue!

```
void GOMP_barrier (void) {
  real_GOMP_barrier();
  mcgdb_thread_can_run(&mcgdb_can_pass_barrier);
}
```

```
(mcgdb) set mcgdb_can_pass_barrier = 0
// wait for everybody
(mcgdb) set mcgdb_can_pass_barrier = 1
(mcgdb) thread apply all finish #(twice)
```


# Multithreading GDB ...

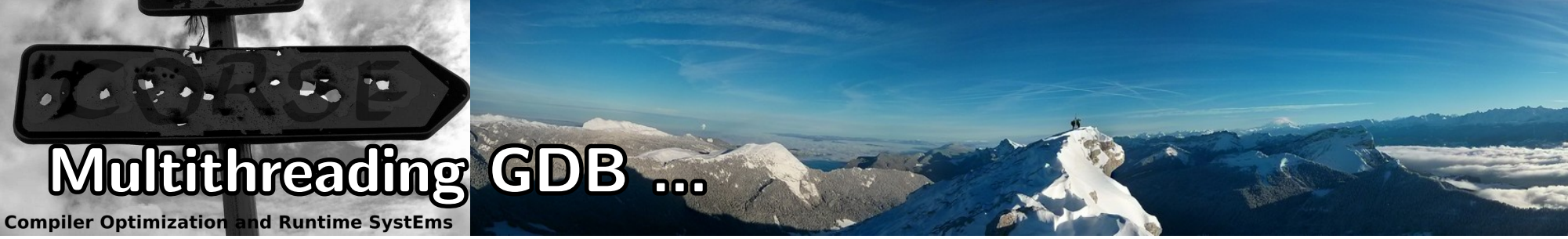
Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame




1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

```
(gdb) where
#0  sigsuspend () from /usr/lib/libc.so.6
#1  wait_lwp (lp=lp@entry=0x21f63b0) at ../../gdb/gdb/linux-na
#2  stop_wait_callback (lp=0x21f63b0, data=<optimized out>) at
#3  iterate_over_lwps (filter=..., callback=callback@entry=0x4
#4  linux_nat_wait_1 (ops=<optimized out>, target_options=1, c
#5  linux_nat_wait (ops=<optimized out>, ptid=..., ourstatus=0
#6  thread_db_wait (ops=<optimized out>, ptid=..., ourstatus=0
#7  delegate_wait (self=<optimized out>, arg1=..., arg2=<optim
#8  target_wait (ptid=..., status=status@entry=0x7fffffffe3b0
```


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

```
/* Wait for next SIGCHLD and try again.  This may let SIGCHLD
handler get invoked despite our caller had them intentionally
blocked by block_child_signal. This is sensitive only to the
loop of linux_nat_wait and there if we get called my_waitpid
gets called again before it gets to sigsuspend so we can
let the handlers get executed here.  (gdb/linux-nat.c) */

sigsuspend (&suspend_mask);
```


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

```
NAME
       sigsuspend, rt_sigsuspend - wait for a signal
SYNOPSIS
       int sigsuspend(const sigset_t *mask);
DESCRIPTION
 sigsuspend() temporarily replaces the signal mask of the
 calling process with the mask given by mask and then
 suspends the process until delivery of a signal whose action
 is to invoke a signal handler or to terminate a process.
```


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

   $\Rightarrow$ the `SIGCHLD` doesn't reach the right thread, and GDB's execution thread never wakes up


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

   $\Rightarrow$ the `SIGCHLD` doesn't reach the right thread, and GDB's execution thread never wakes up
   - understand how to fix it ...


# Multithreading GDB ...

Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

   $\Rightarrow$ the `SIGCHLD` doesn't reach the right thread,
   and GDB's execution thread never wakes up
   - understand how to fix it ...

   $\Rightarrow$ already done by GDB guys for Guile support! (PR 17247)


# Multithreading GDB ...

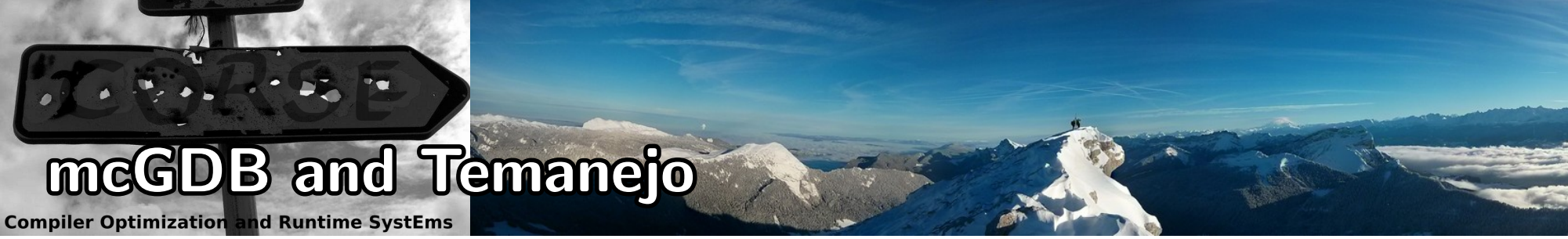
Compiler Optimization and Runtime SystEms

1. create thread **in** GDB (e.g. initialize `Ayudame`, or for a GUI ...)
2. start the application
3. see GDB hanging
   - try to 'fix' Ayudame
     OR
   - take a look at GDB internals

   $\Rightarrow$ the `SIGCHLD` doesn't reach the right thread,
   and GDB's execution thread never wakes up
   - understand how to fix it ...

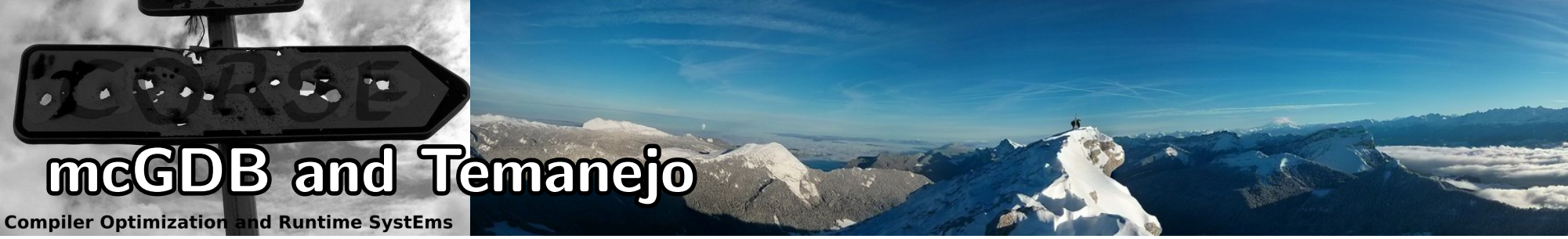
   $\Rightarrow$ already done by GDB guys for Guile support! (PR 17247)

```
import pysigset, signal

with pysigset.suspended_signals(signal.SIGCHLD):
     # start threads, they will inherit the  signal mask
     pass
```




## What I need/like from Temanejo:

- Ideas very close to mcGDB :)
- Graphical interface more intuitive than CLI
- Nice and interactive graph visualization
- Support of OMP task dependencies

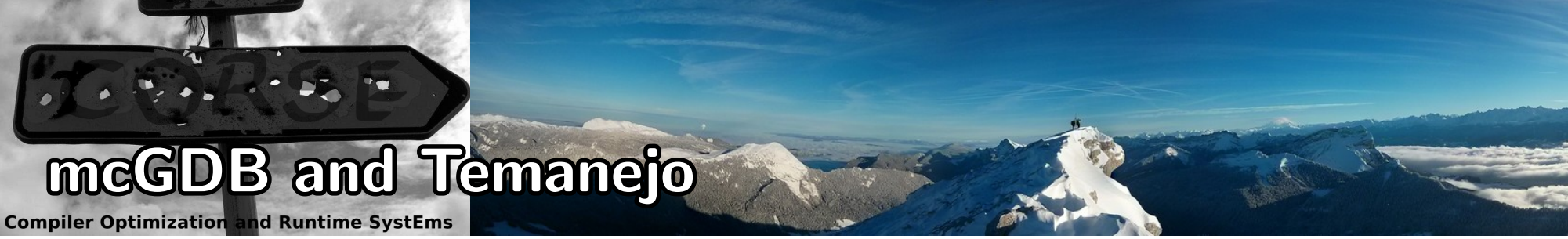



## What I need/like from Temanejo:

- Ideas very close to mcGDB :)
- Graphical interface more intuitive than CLI
- Nice and interactive graph visualization
- Support of OMP task dependencies

## What I like ... less:

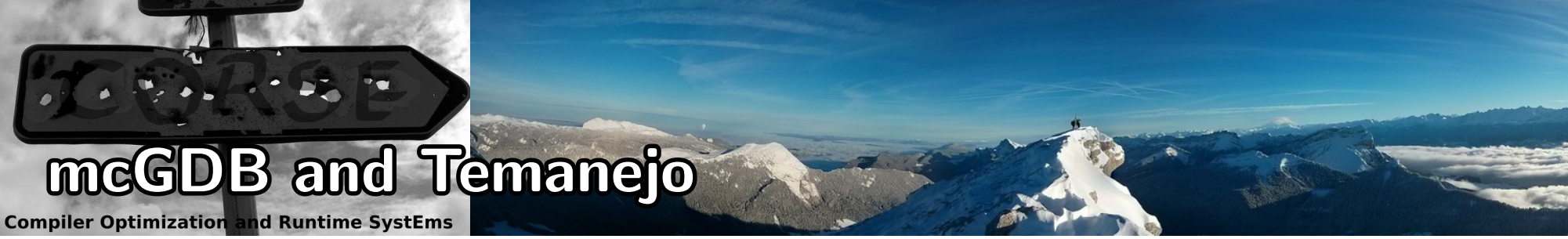
- No integration with GDB (AFAIK) / cooperation is difficult
  - GDB/DDT may block Temanejo/Ayudame commmunications
  - No 'model-level' knowledge in GDB/DDT




## What I'd like to do:

- Feed Ayudame with mcGDB information (or directly Temanejo)
- Translate Temanejo breakpoint orders into mcGDB catchpoints

  $\Longrightarrow$ use Temanejo to visualise, mcGDB to control and capture




## What I'd like to do:

- Feed Ayudame with mcGDB information (or directly Temanejo)
- Translate Temanejo breakpoint orders into mcGDB catchpoints

  $\Longrightarrow$ use Temanejo to visualise, mcGDB to control and capture

## What I can do today:

- GOMP/IOMP: 'complete' support (seqdiag and control)
- OmpSs: seqdiag and task dependencies (proof of concept)

$\Longrightarrow$ Choose an architecture,
understand where to hook,
discuss interesting features,
etc.