STMicroelectronics
LIG
University of Grenoble

life.augmented

Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget, *UJF-LIG, STMicroelectronics*
Miguel Santana, *STMicroelectronics*
Jean-François Méhaut, *UJF-CEA/LIG*

MAD Workshop'14, Athens, Greece — October 8th, 2014

Introduction

Embedded Systems and MPSoC

Consumer Electronics Devices

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

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

Introduction

Embedded Systems and MPSoC

Consumer Electronics Devices

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

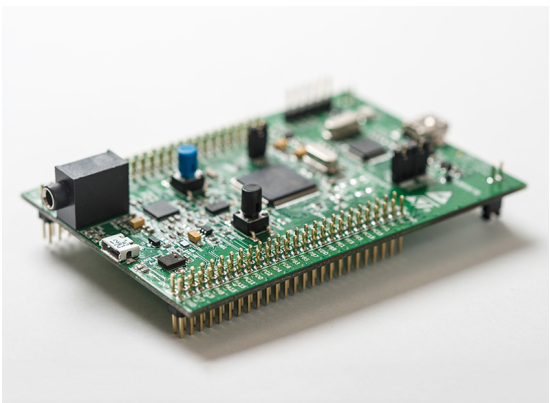
- High-resolution multimedia apps
  - H.265 HEVC
  - Augmented reality
  - 3D video games
  - ...

$\Rightarrow$ high performance expectations.

Introduction

Embedded Systems and MPSoC

Current applications have high performance expectations...



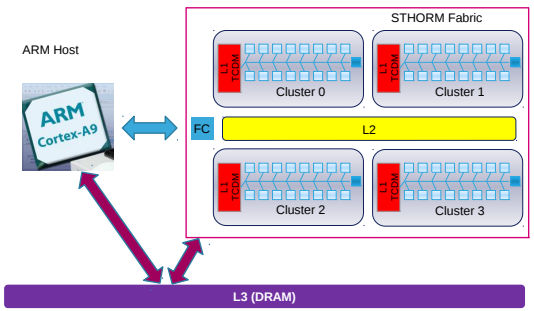
⇒ important demand for:

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

Introduction

Embedded Systems and MPSoC

Current applications have high performance expectations...



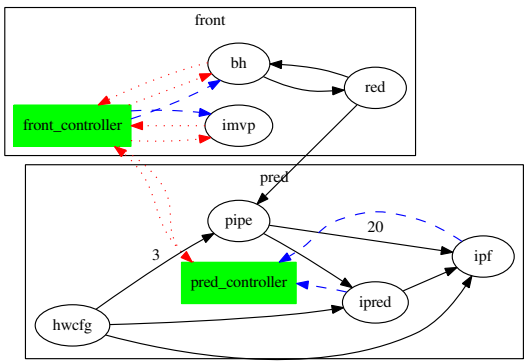
⇒ important demand for:

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

Introduction

Embedded Systems and MPSoC

Current applications have high performance expectations...



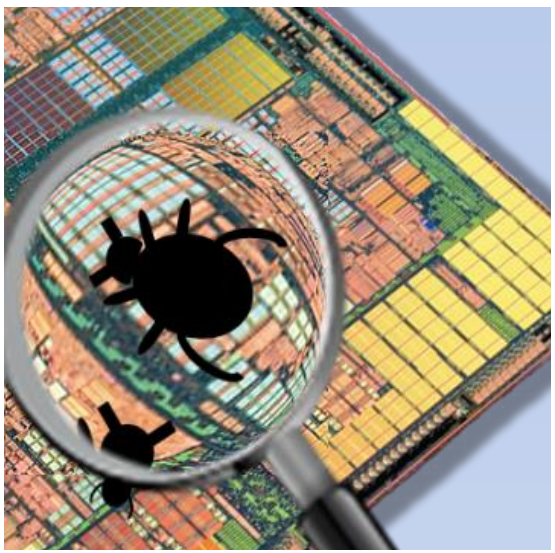
⇒ important demand for:

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

Introduction

Embedded Systems and MPSoC

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
  - Workshop and our research effort

Agenda

1. Background: MPSoC Programming and Debugging
2. Programming Model Centric Interactive Debugging
3. Model-Centric Debugger Case-Study

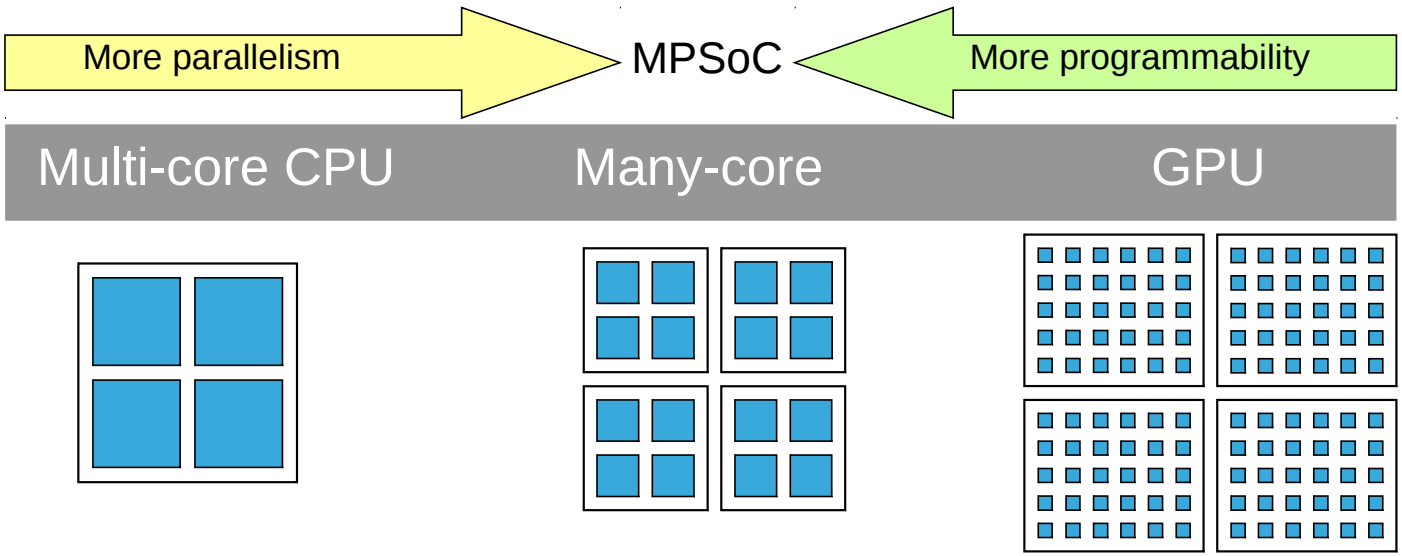
Agenda

1. Background: MPSoC Programming and Debugging
2. Programming Model Centric Interactive Debugging
3. Model-Centric Debugger Case-Study

Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

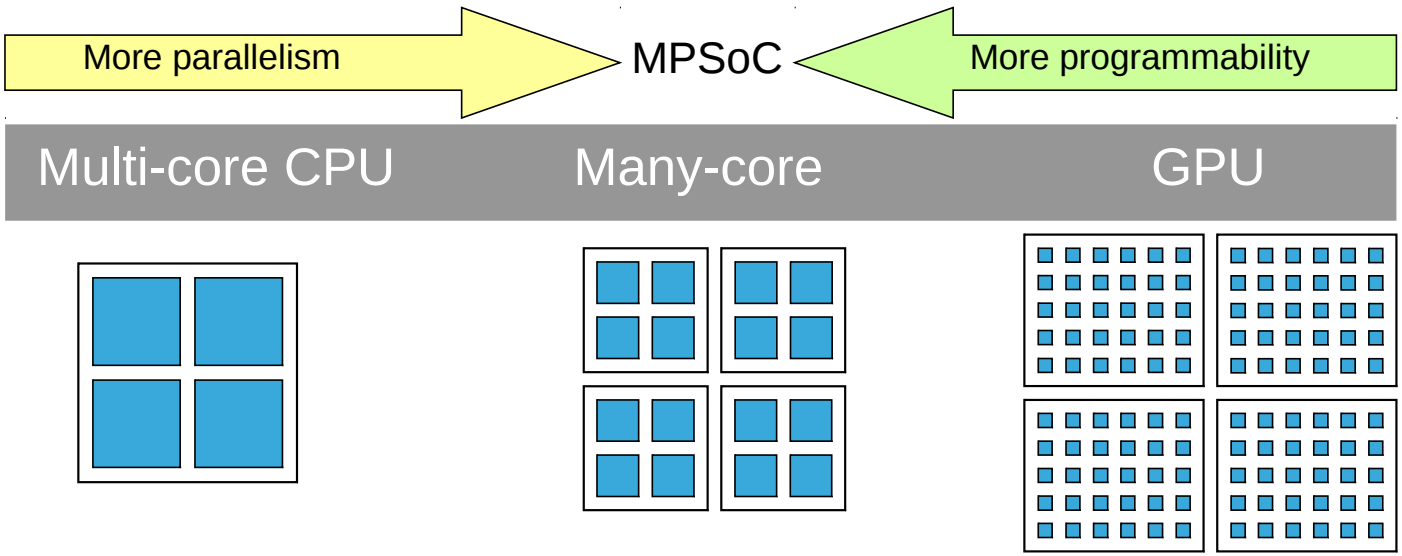
MultiProcessor System on-a-Chip



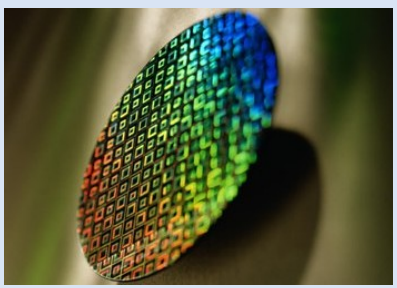
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System on-a-Chip



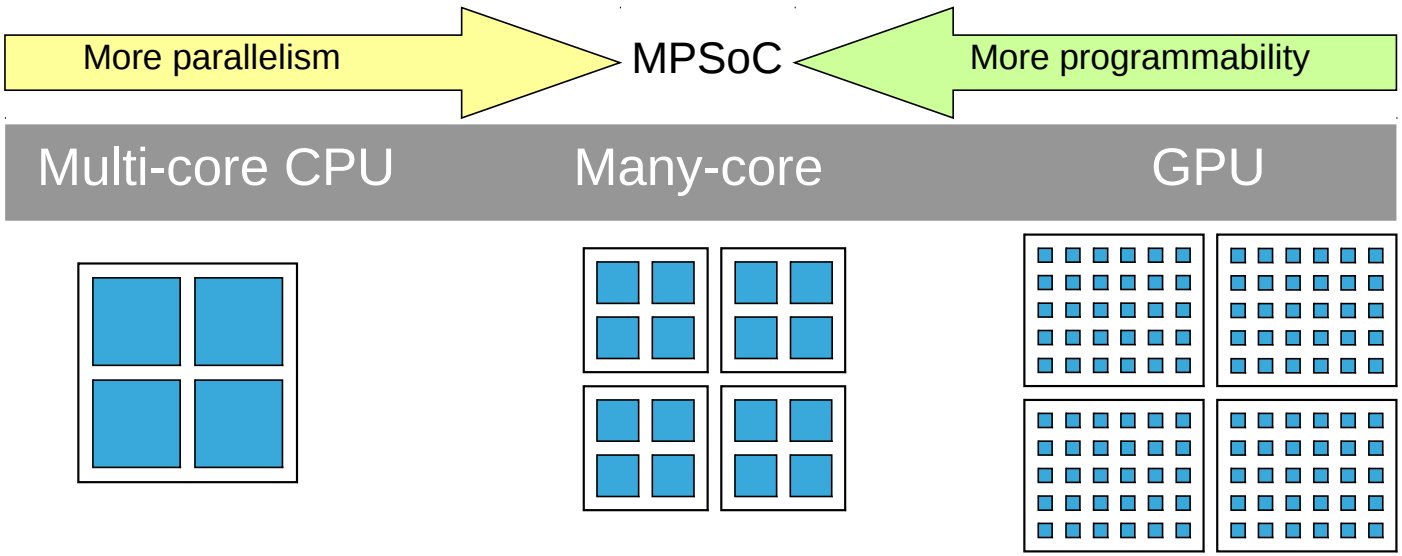
- Many-core processor for embedded systems
- Low energy-consumption
- Heterogeneous computing power



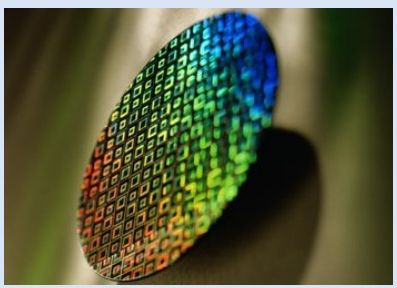
Background: MPSoC Programming and Debugging

MPSoC and GPU Systems

MultiProcessor System on-a-Chip



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



How to program such complex architectures?

Background: MPSoC Programming and Debugging

Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

Background: MPSoC Programming and Debugging

Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

$\rightarrow$ application written on top of an *abstract* machine

Background: MPSoC Programming and Debugging

Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

$\rightarrow$ application written on top of an *abstract* machine

... implemented by supportive environments:
programming frameworks, runtime libraries, APIs

Background: MPSoC Programming and Debugging

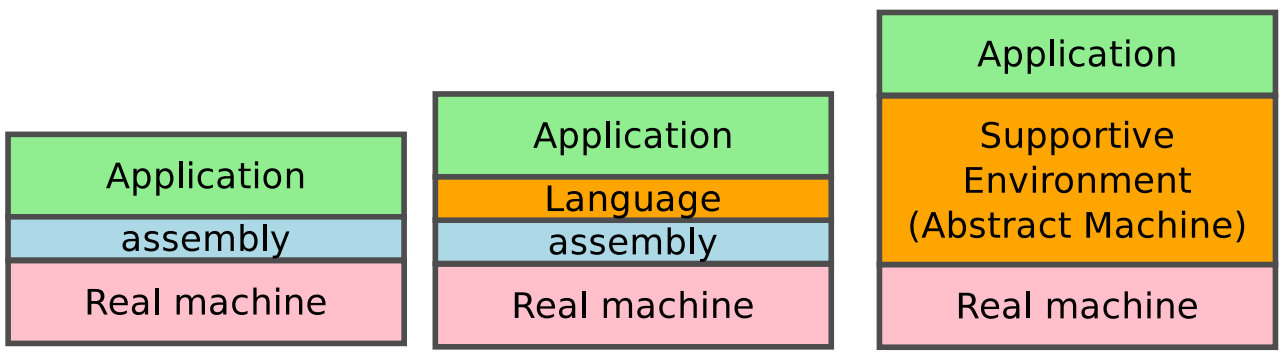
Programming Models and Supportive Environments

... with programming models!

- **Programmability** with high-level abstractions
- **Portability** thanks to an hardware-independent interface
- **Separation of concerns** between application / lower levels

→ application written on top of an *abstract* machine

... implemented by supportive environments:
programming frameworks, runtime libraries, APIs



Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

Components	Dataflow

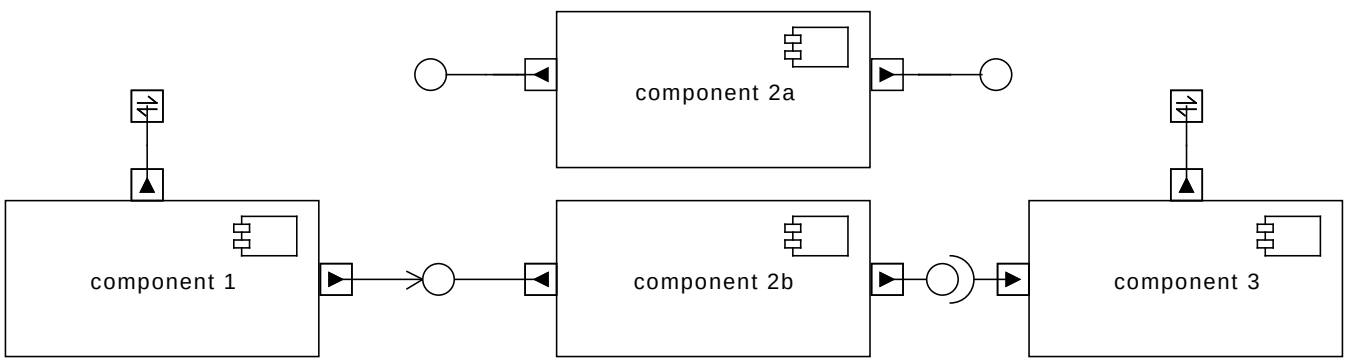
Background: MPSoC Programming and Debugging

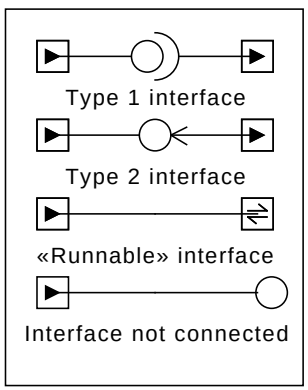
Programming Models for ST MPSoCs

Components

- code/data encapsulation
- lang.-free interfaces

Dataflow





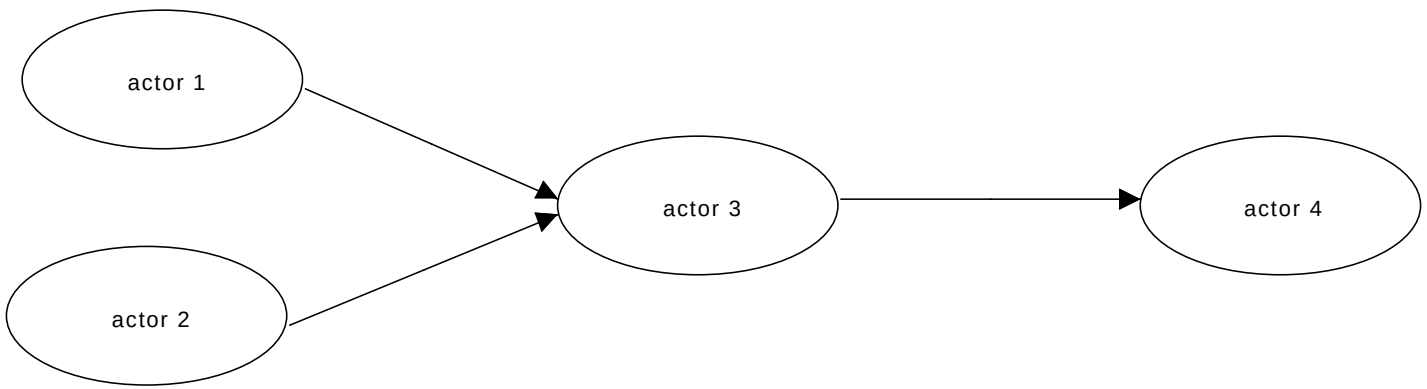
Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

Components

Dataflow

- streams of data
- implicit parallelism



Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

Components	Dataflow
• code/data encapsulation	• streams of data
• lang.-free interfaces	• implicit parallelism

large programming domain coverage...

Background: MPSoC Programming and Debugging

Programming Models for ST MPSoCs

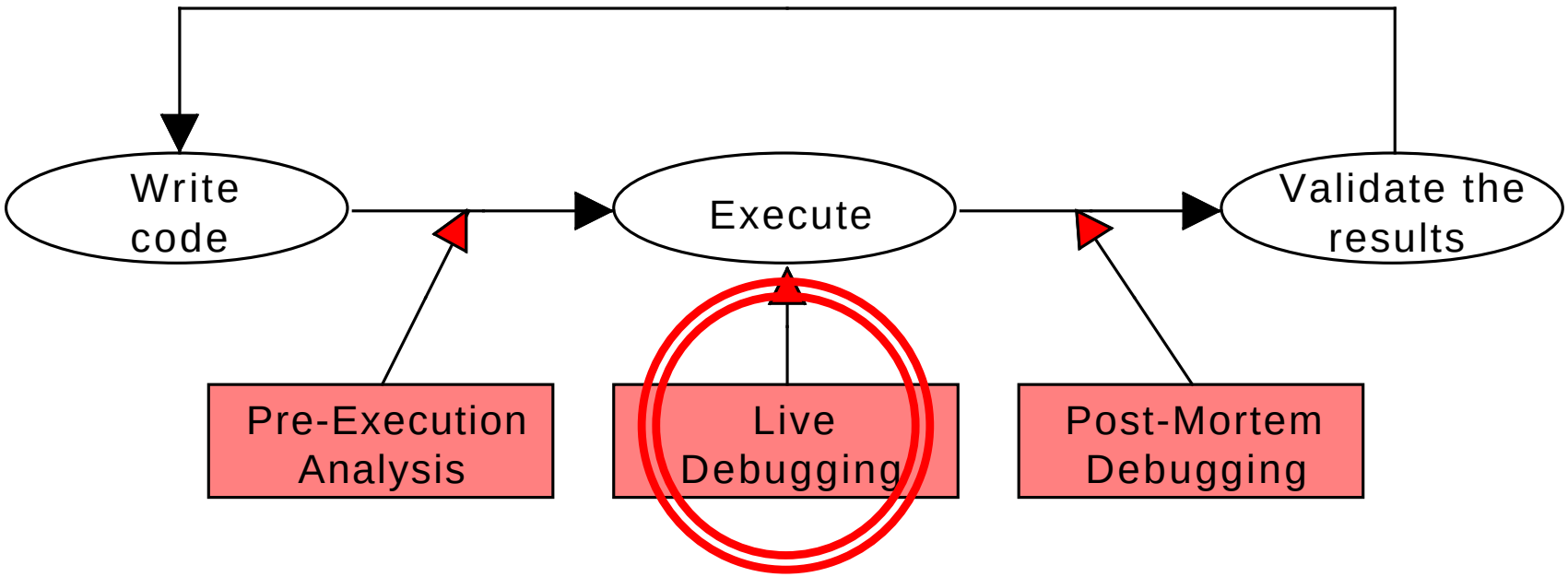
Components	Dataflow
• code/data encapsulation	• streams of data
• lang.-free interfaces	• implicit parallelism

large programming domain coverage...

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

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging



Interactive Debugging (eg.: GDB)

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

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)

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

- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)

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

- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection

... but nothing related to Supportive Environments ...

Background: MPSoC Programming and Debugging

Tools and Techniques, Advantages of Interactive Debugging

Interactive Debugging (eg.: GDB)

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

- Breakpoints, memory watchpoints, event catchpoints, ...
- Step-by-step execution
- Memory and processor inspection

... but nothing related to Supportive Environments ...

⇒ Debuggers cannot access the *abstract* machine!

Background: MPSoC Programming and Debugging

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

1. Background: MPSoC Programming and Debugging
2. Programming Model Centric Interactive Debugging
3. Model-Centric Debugger Case-Study

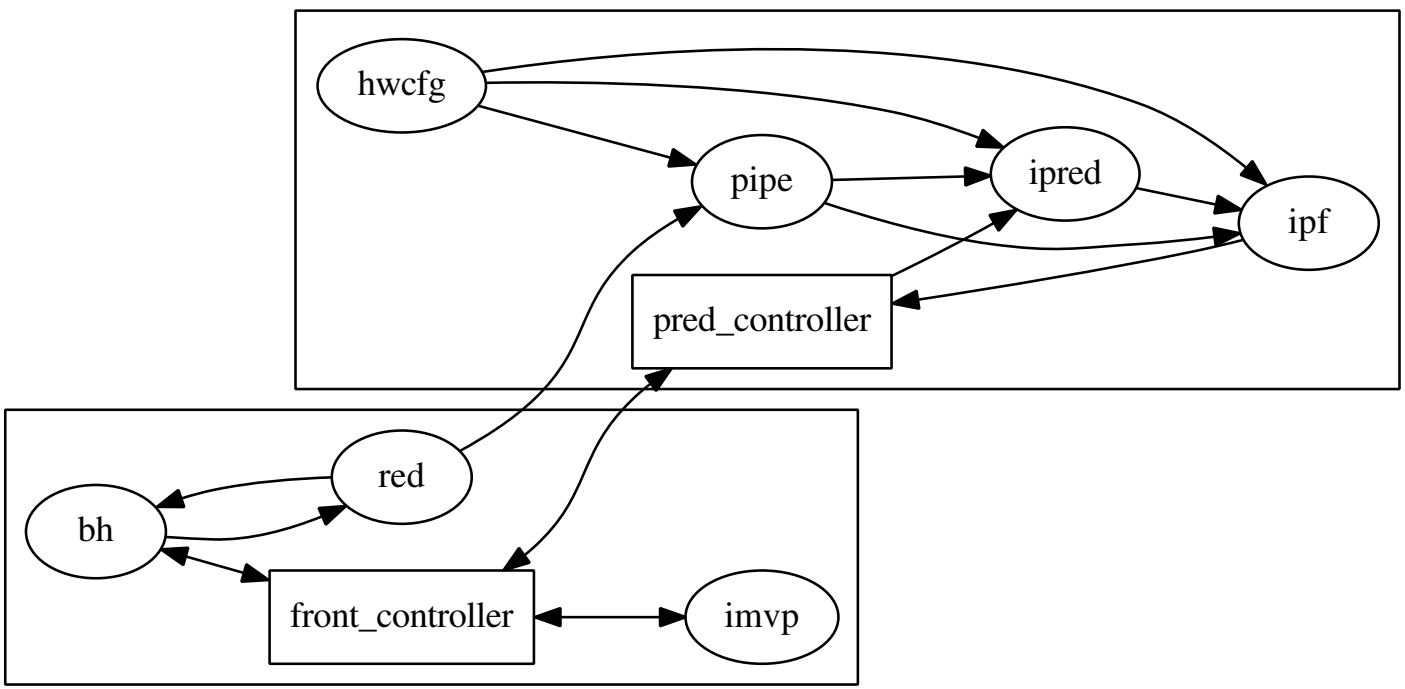
Programming Model Centric Interactive Debugging

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

Programming Model Centric Interactive Debugging

1 Provide a Structural Representation

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

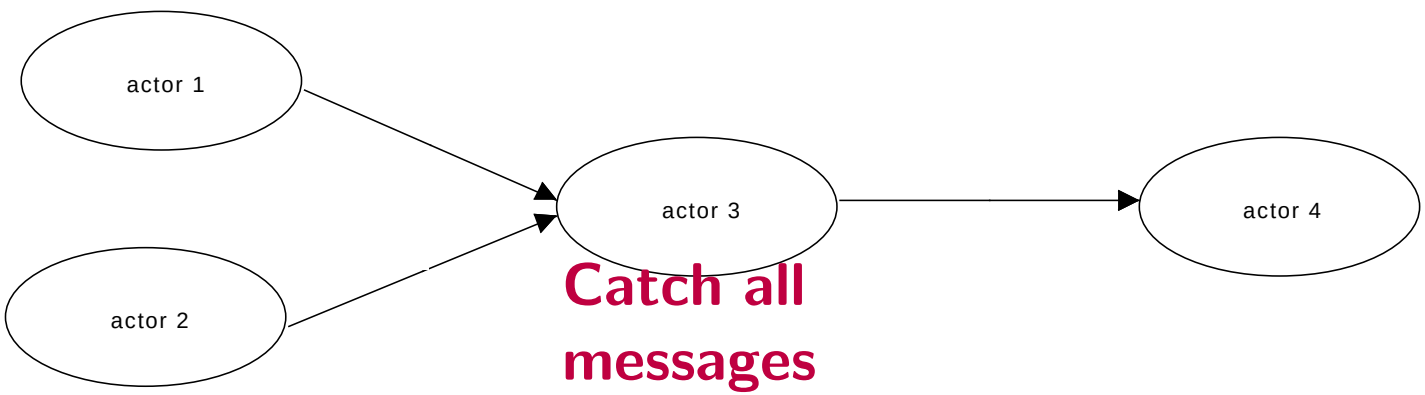


Dataflow graph from the case-study

Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

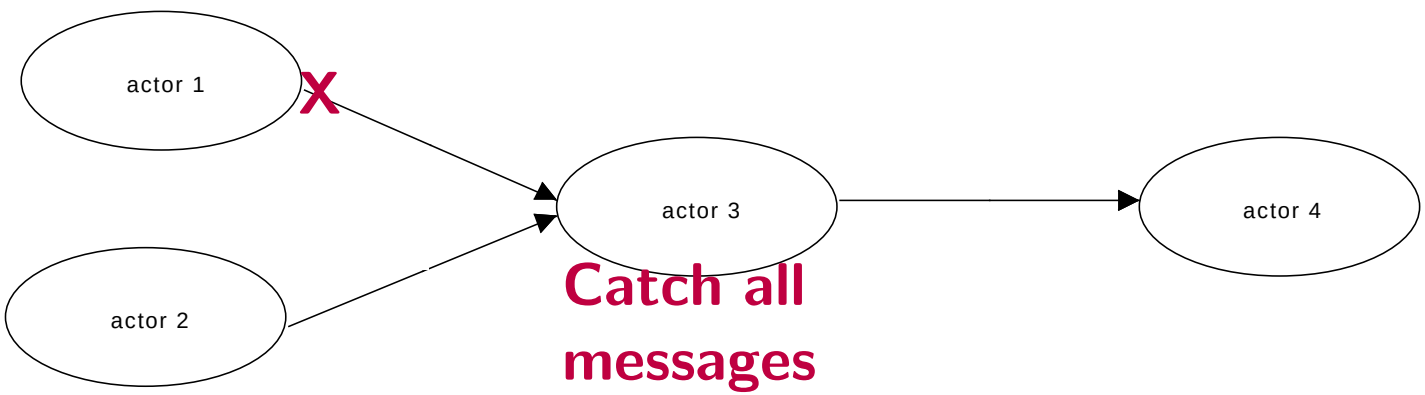
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

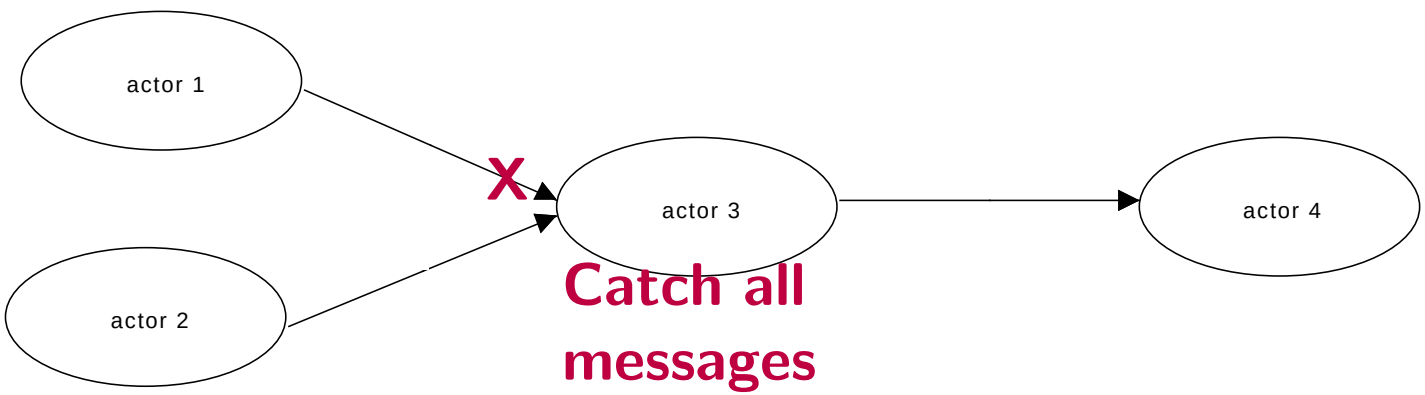
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

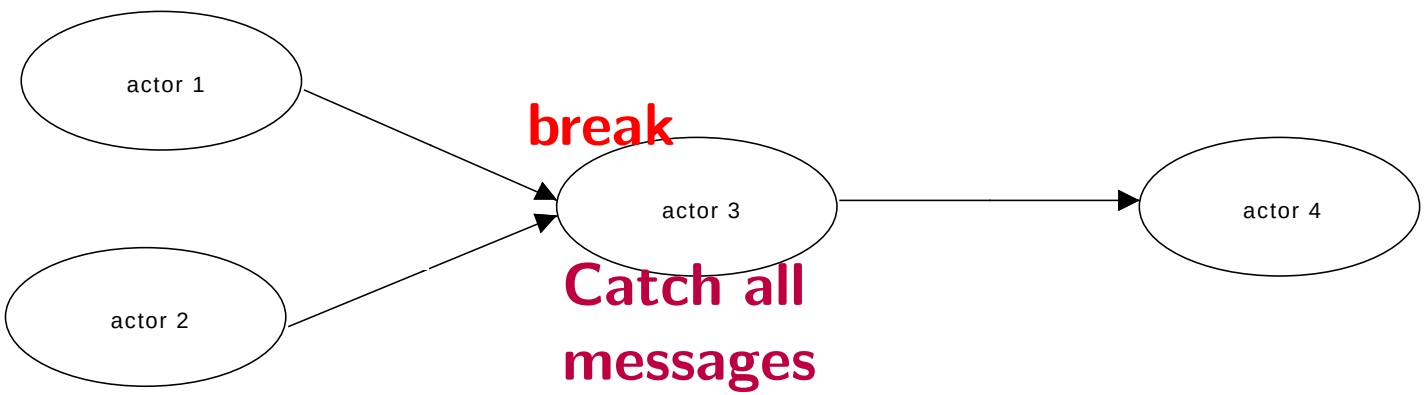
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

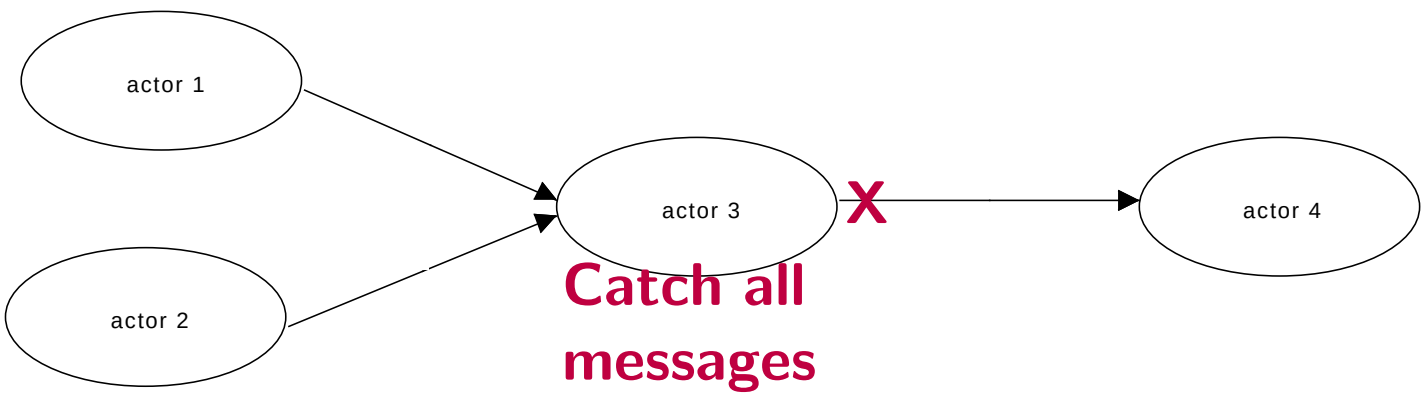
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

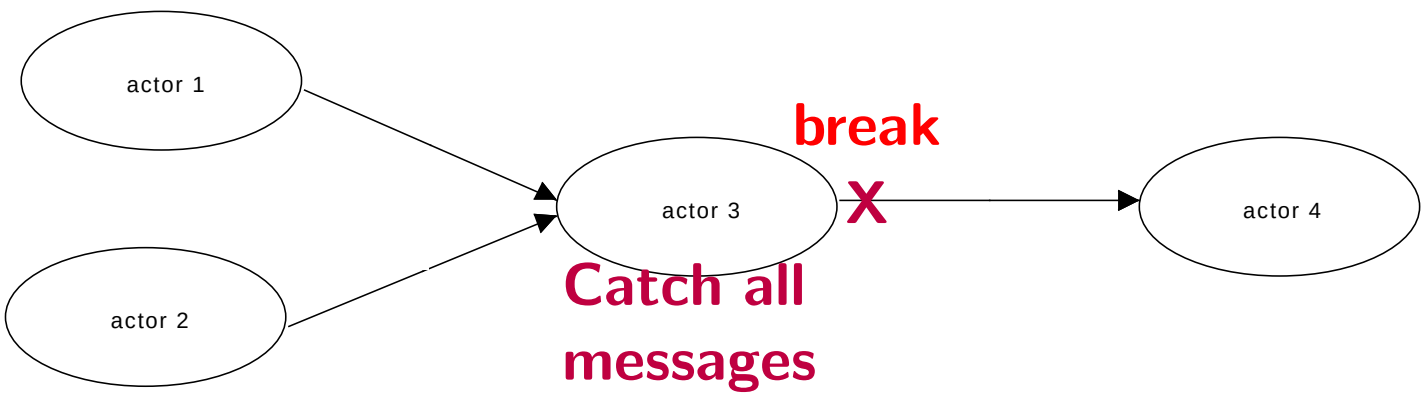
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

2 Monitor Dynamic Behaviors

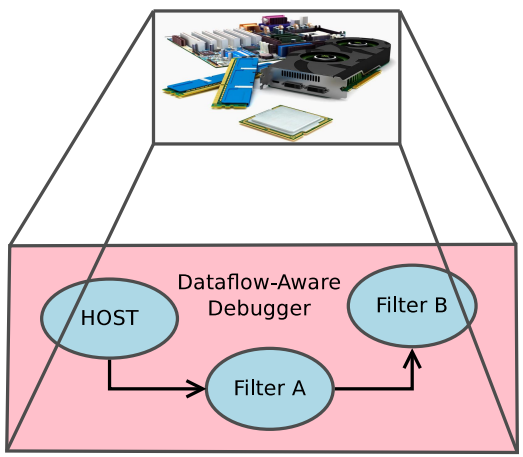
- Monitor the collaboration between the tasks
- Detect communication, synchronization events
  - interpret their pattern and semantics (one-to-one, one-to-many, global or local barriers)
- Offer communication-aware catchpoint mechanisms



Programming Model Centric Interactive Debugging

3 Interact with the Abstract Machine

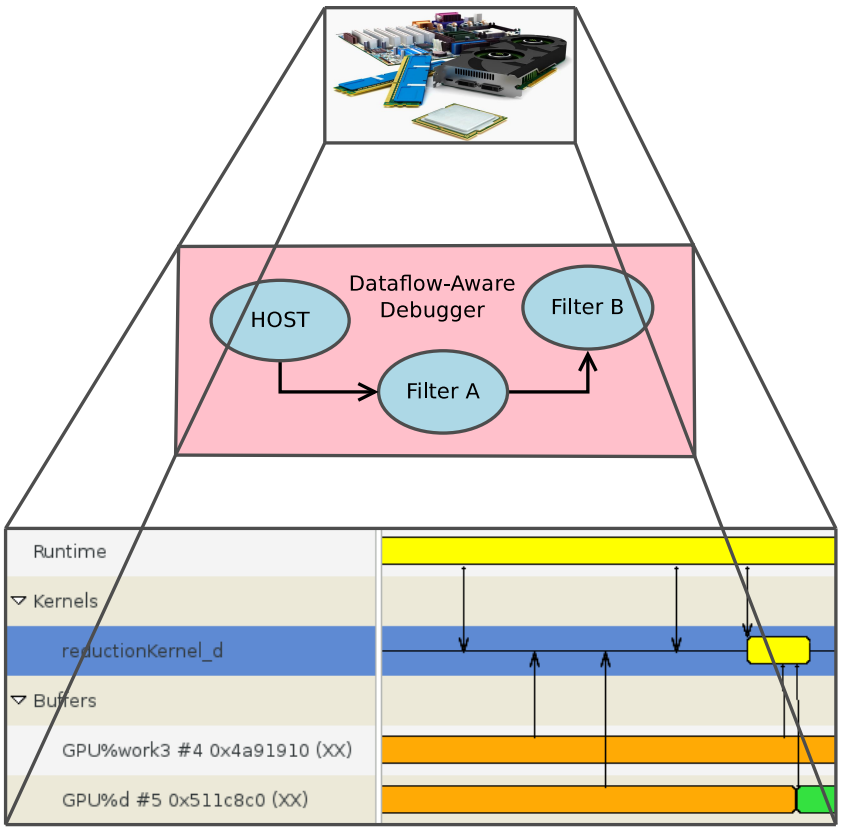
- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



Programming Model Centric Interactive Debugging

3 Interact with the Abstract Machine

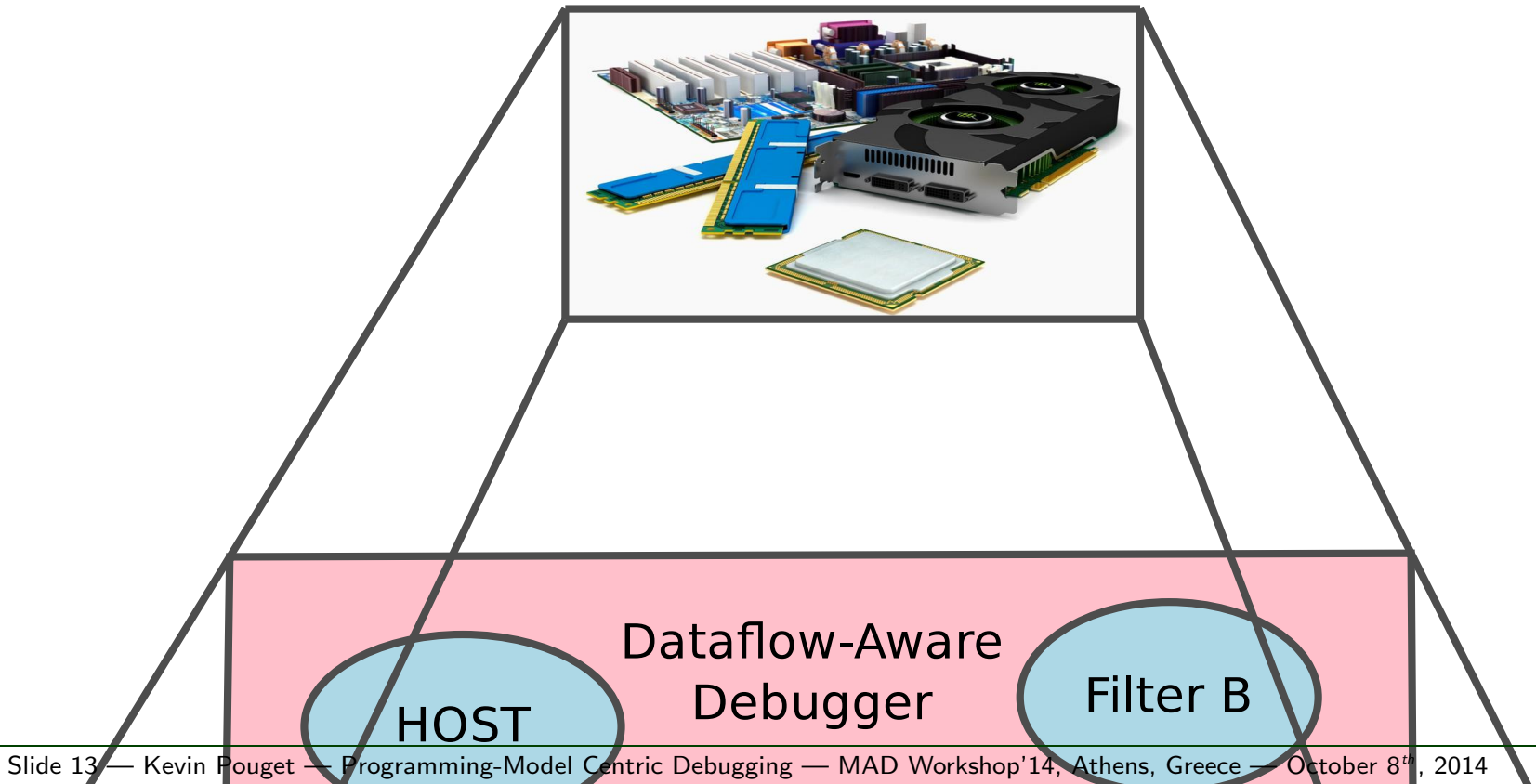
- Recognize the different entities of the model
- Provide details about their state, schedulability, callstack, ...
- Provide support to understand how they reached their current state



Programming Model Centric Interactive Debugging

3 Interact with the ~~Abstract~~ Machine

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



Agenda

1. Background: MPSoC Programming and Debugging
2. Programming Model Centric Interactive Debugging
3. Model-Centric Debugger Case-Study

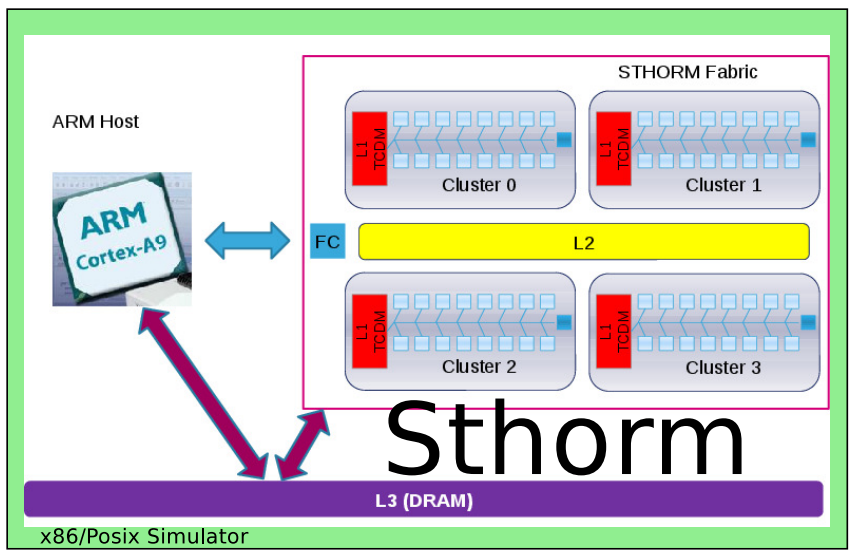
Model-Centric Debugger Case-Study

Proof-of-concept Environment

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Model-Centric Debugger Case-Study

Proof-of-concept Environment

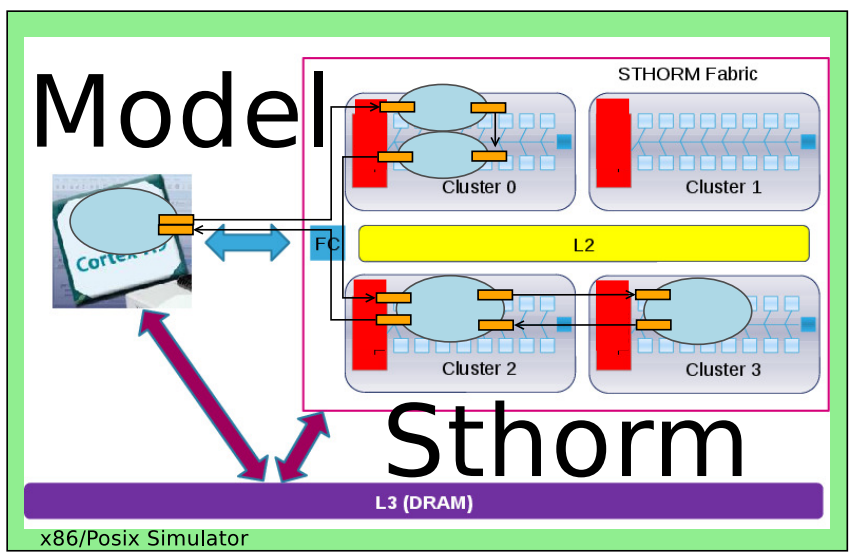
STHORM Progr. Environments

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

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Model-Centric Debugger Case-Study

Proof-of-concept Environment

The GNU Debugger

- Adapted to low level/C debugging
- Large user community

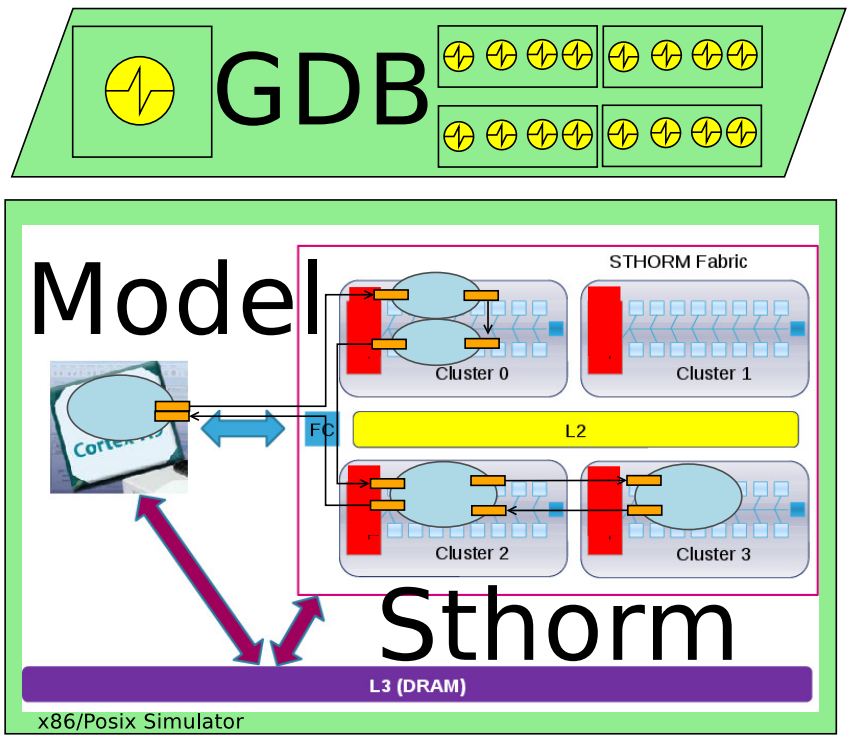
STHORM Progr. Environments

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

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Model-Centric Debugger Case-Study

Proof-of-concept Environment

The GNU Debugger

- Adapted to low level/C debugging
- Large user community
- Extendable with Python API

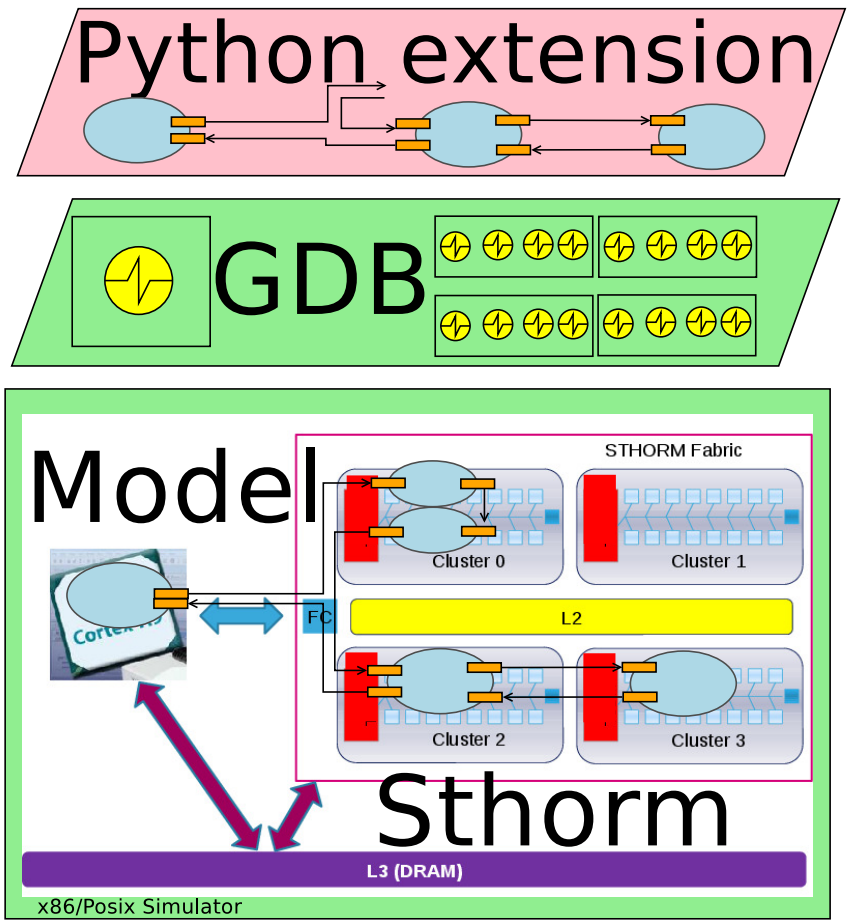
STHORM Progr. Environments

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

STHORM / Platform 2012

ST/CEA MPSoC research platform

- x86 platform simulators



Model-Centric Debugger Case-Study

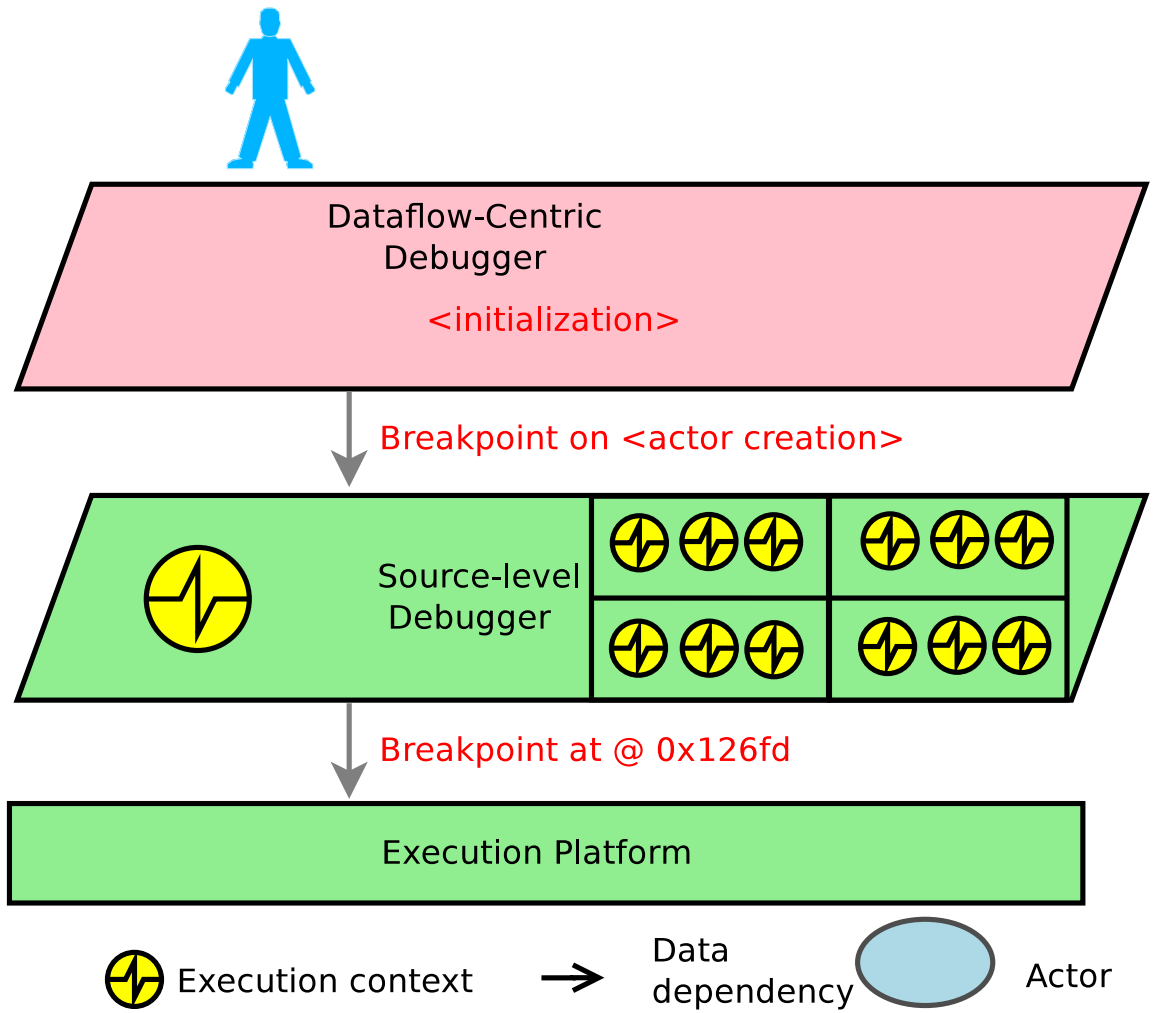
Interpreting Execution Events

$\Rightarrow$ Detect and interpret the exec. events of the runtime framework

Model-Centric Debugger Case-Study

Interpreting Execution Events

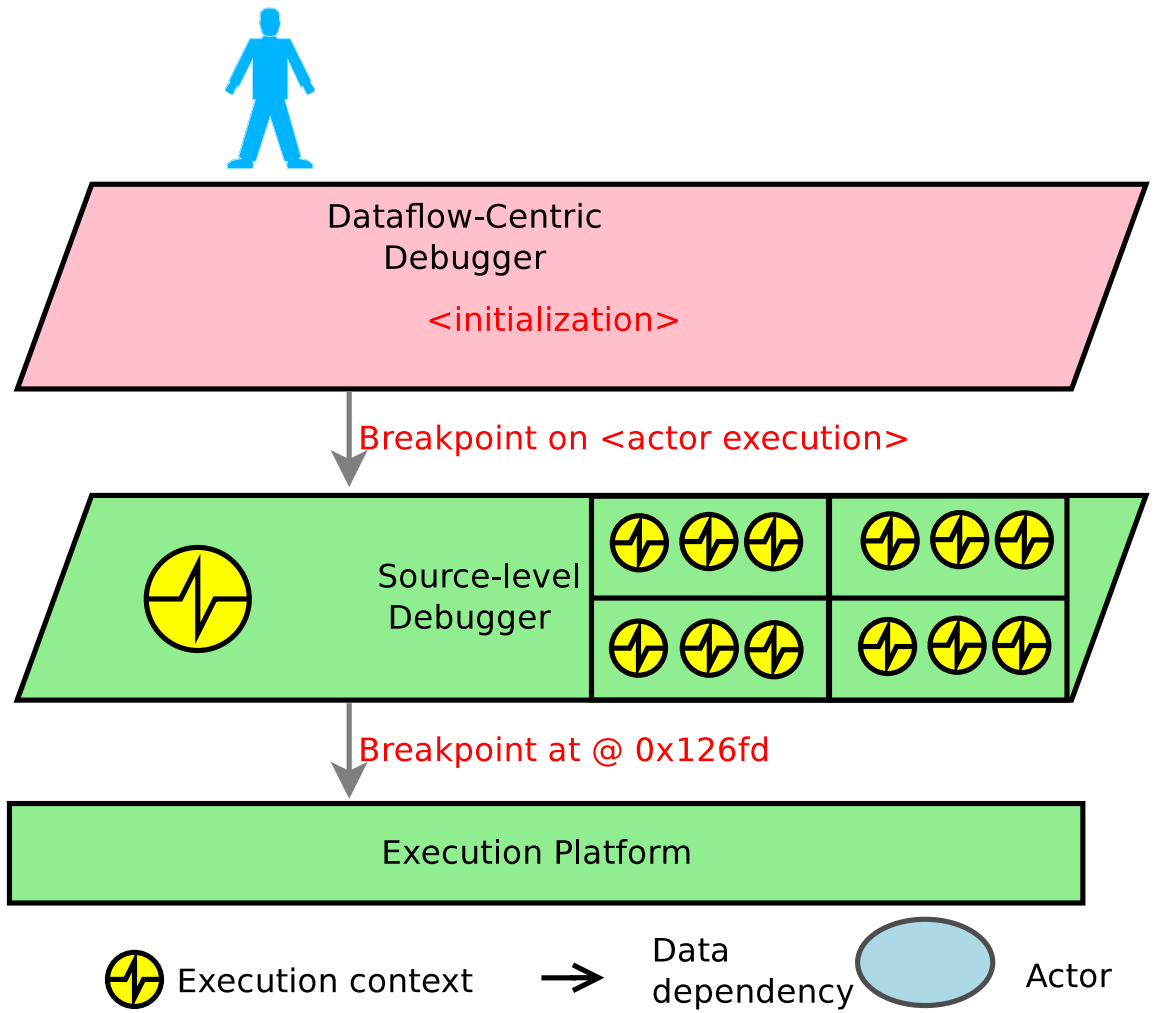
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

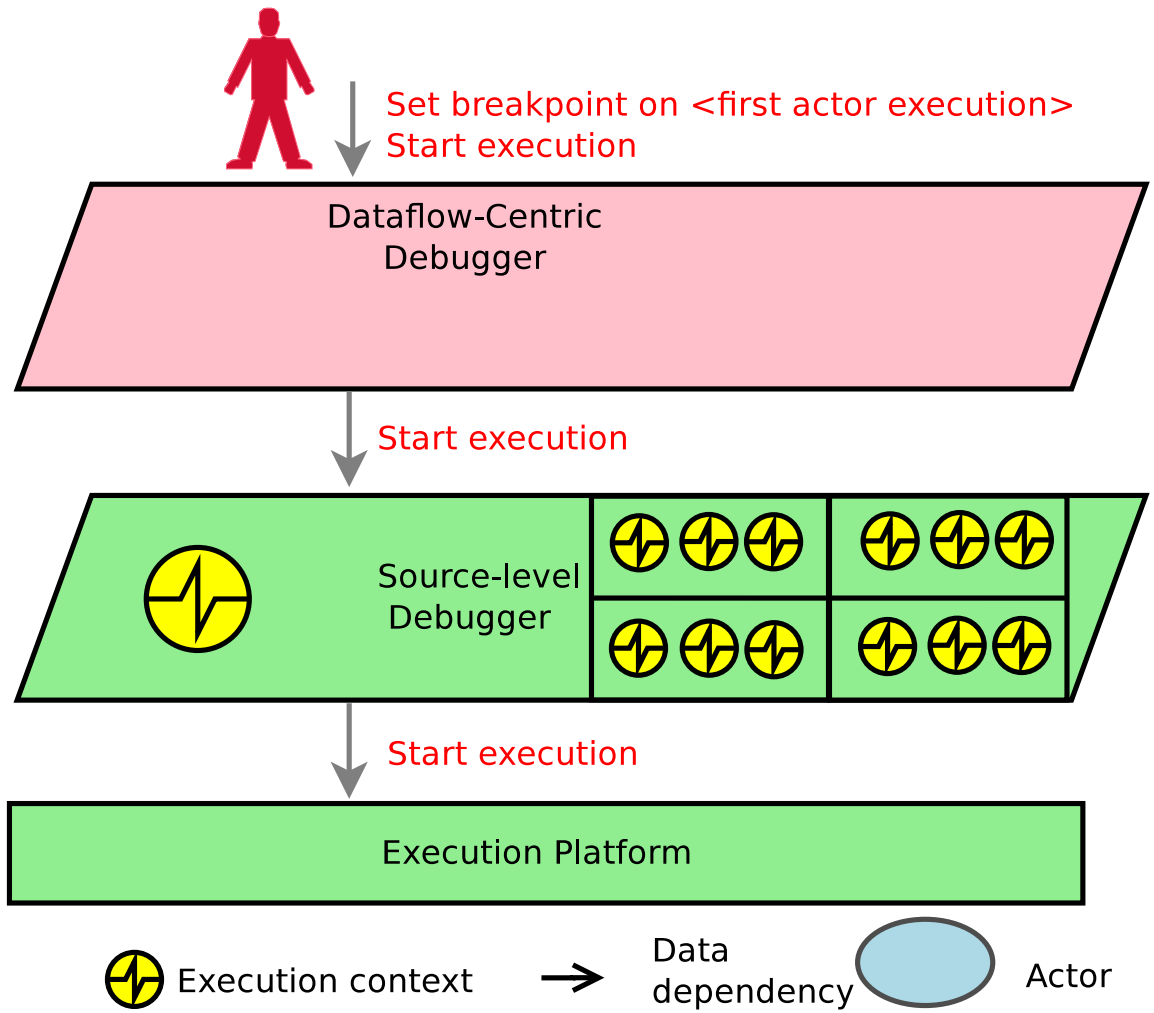
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

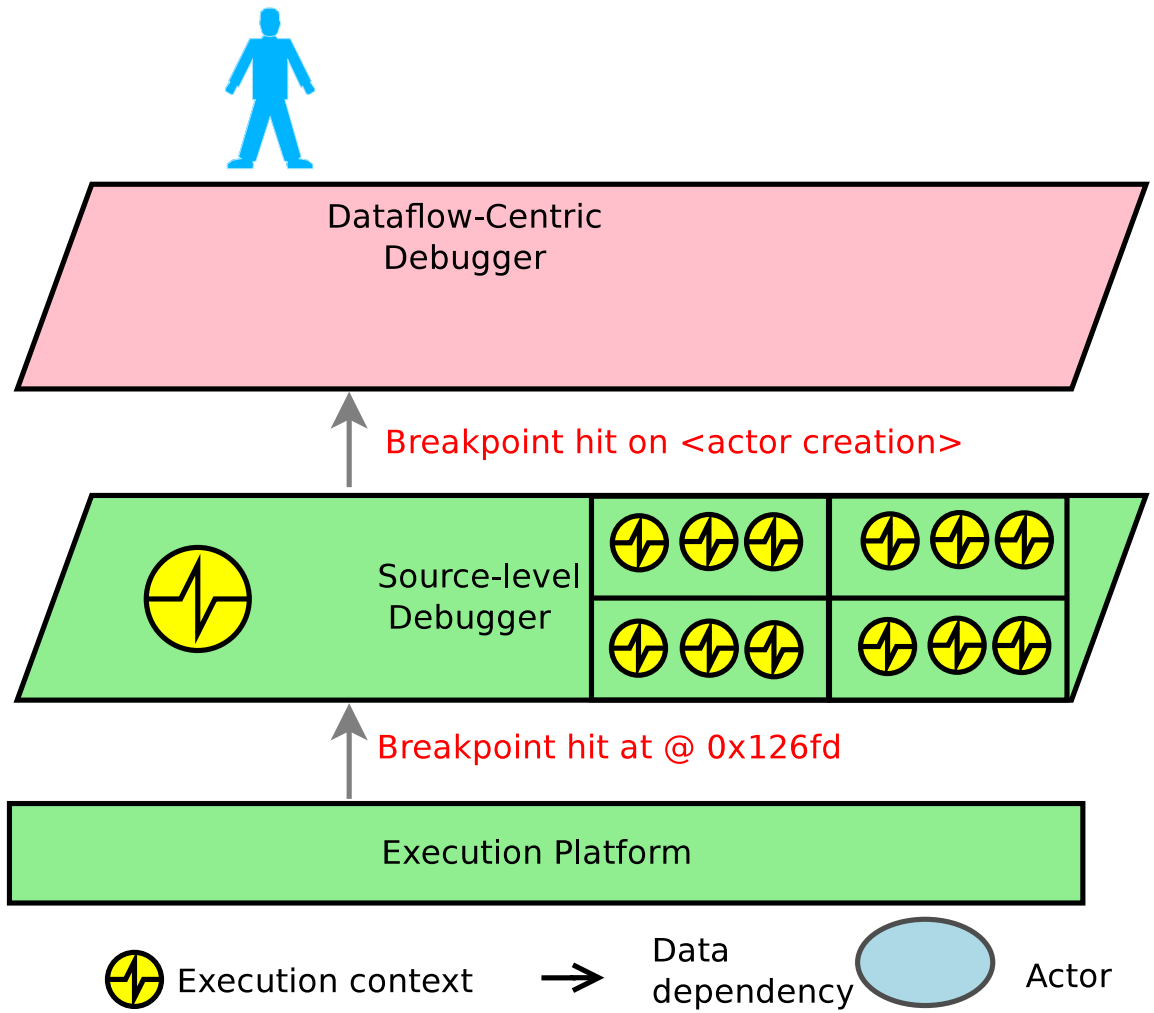
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

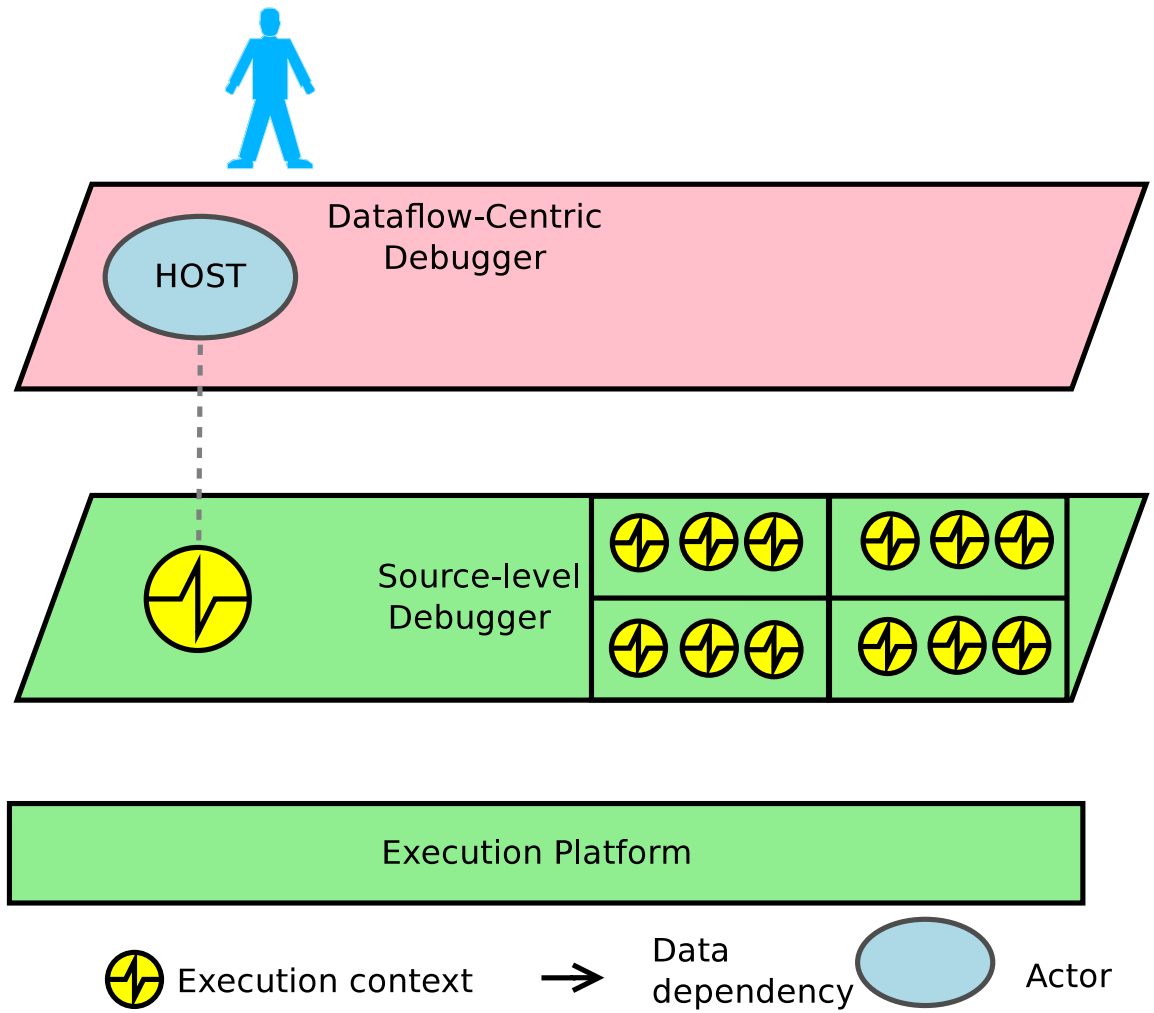
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

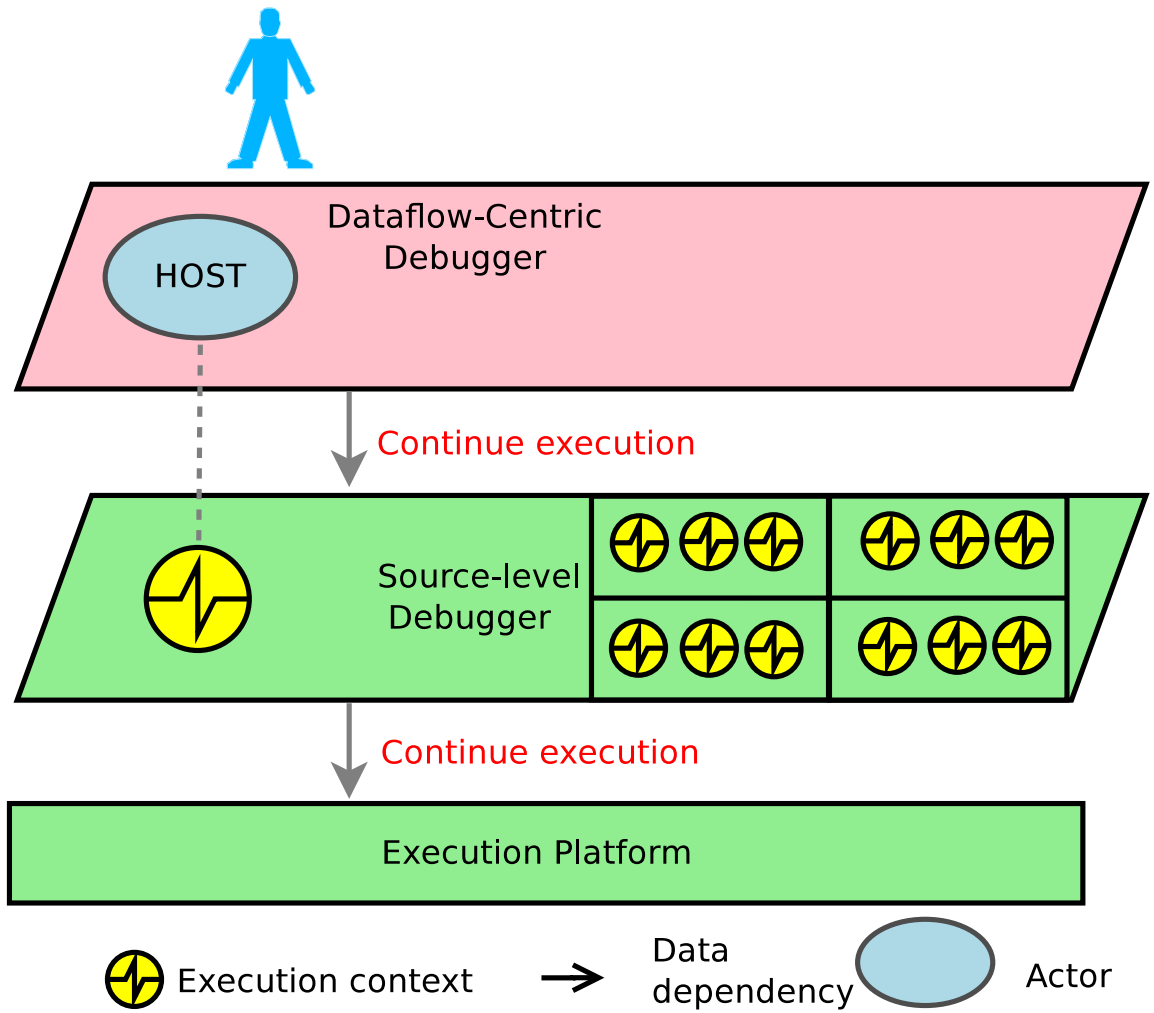
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

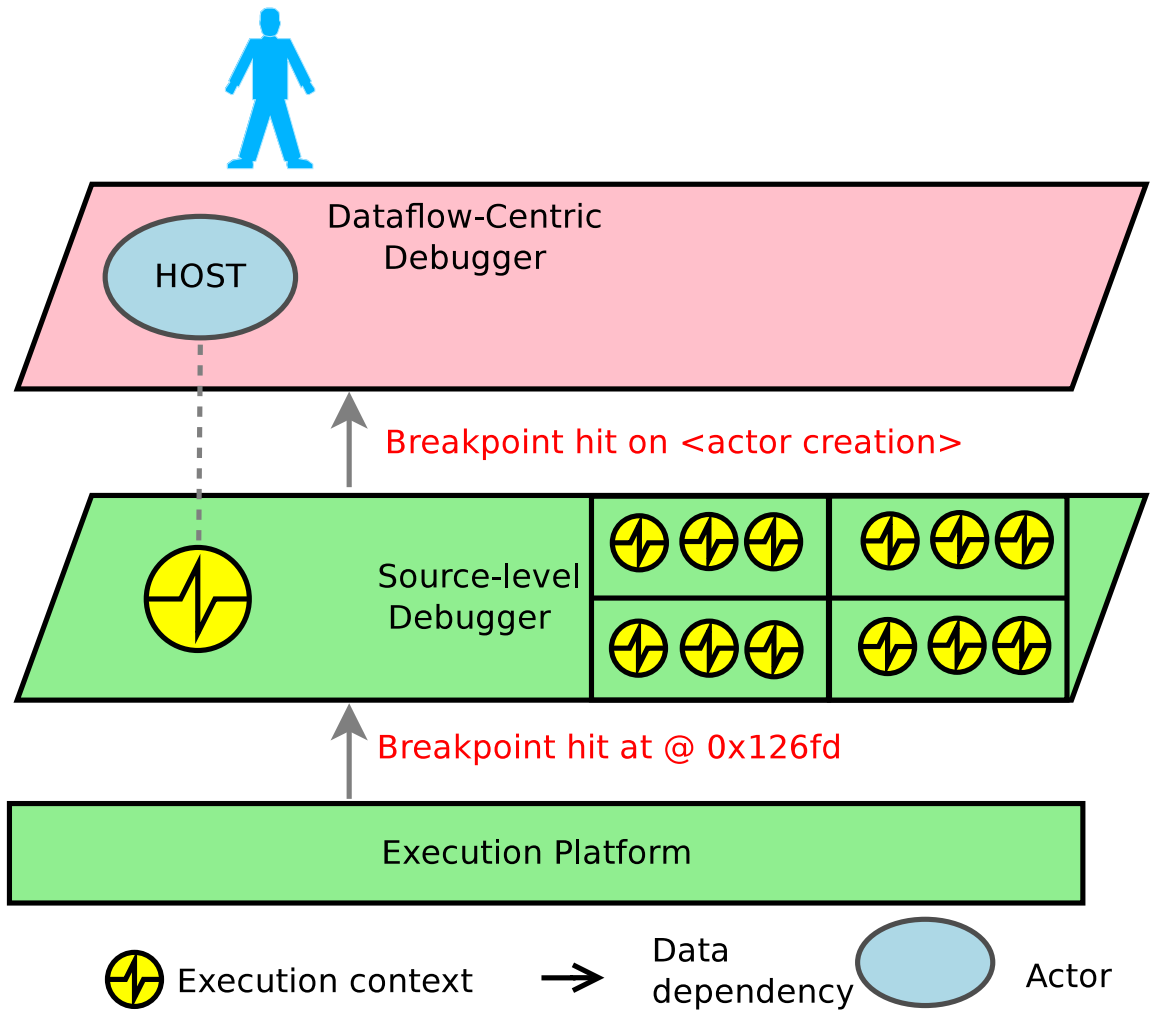
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

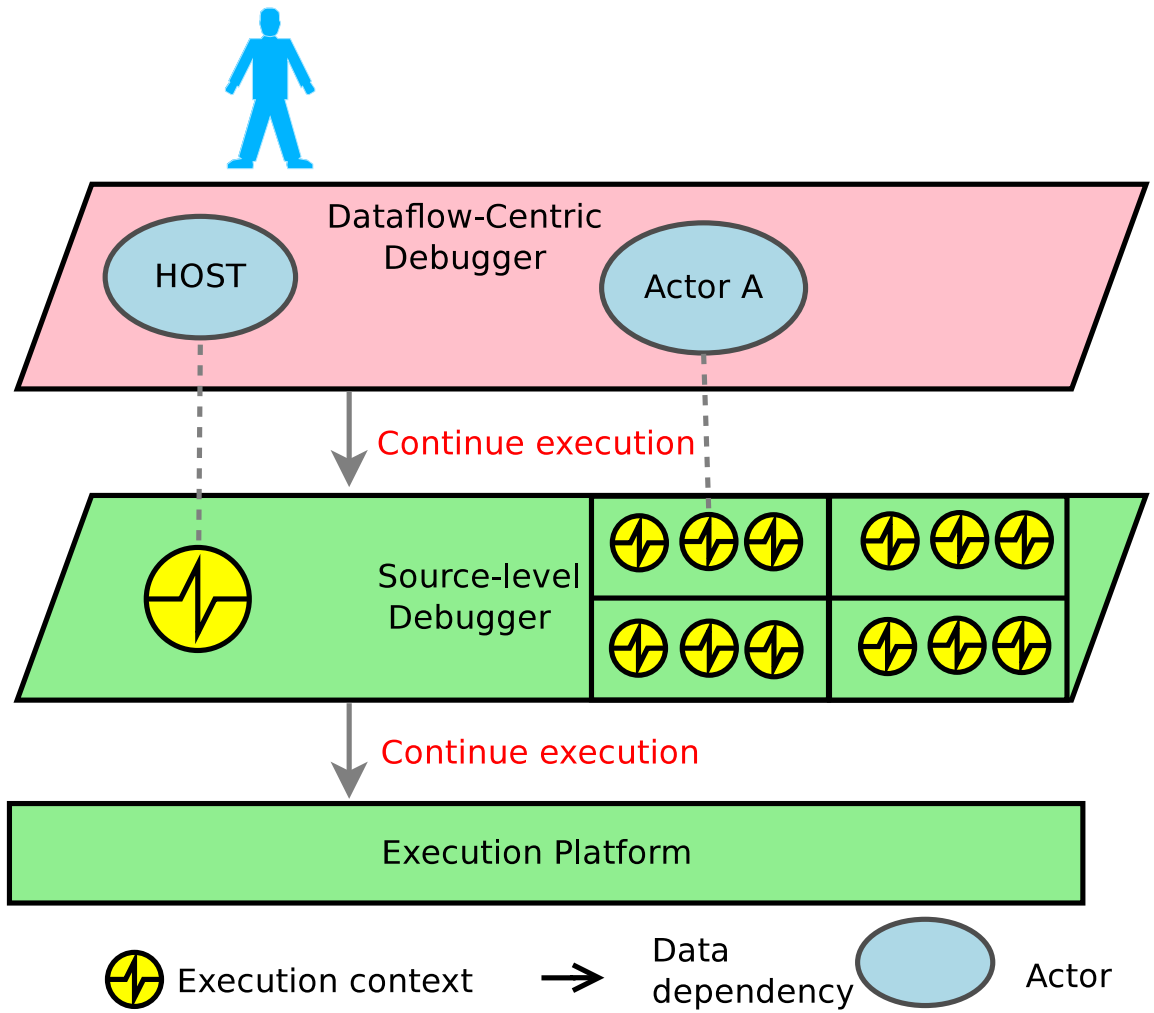
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

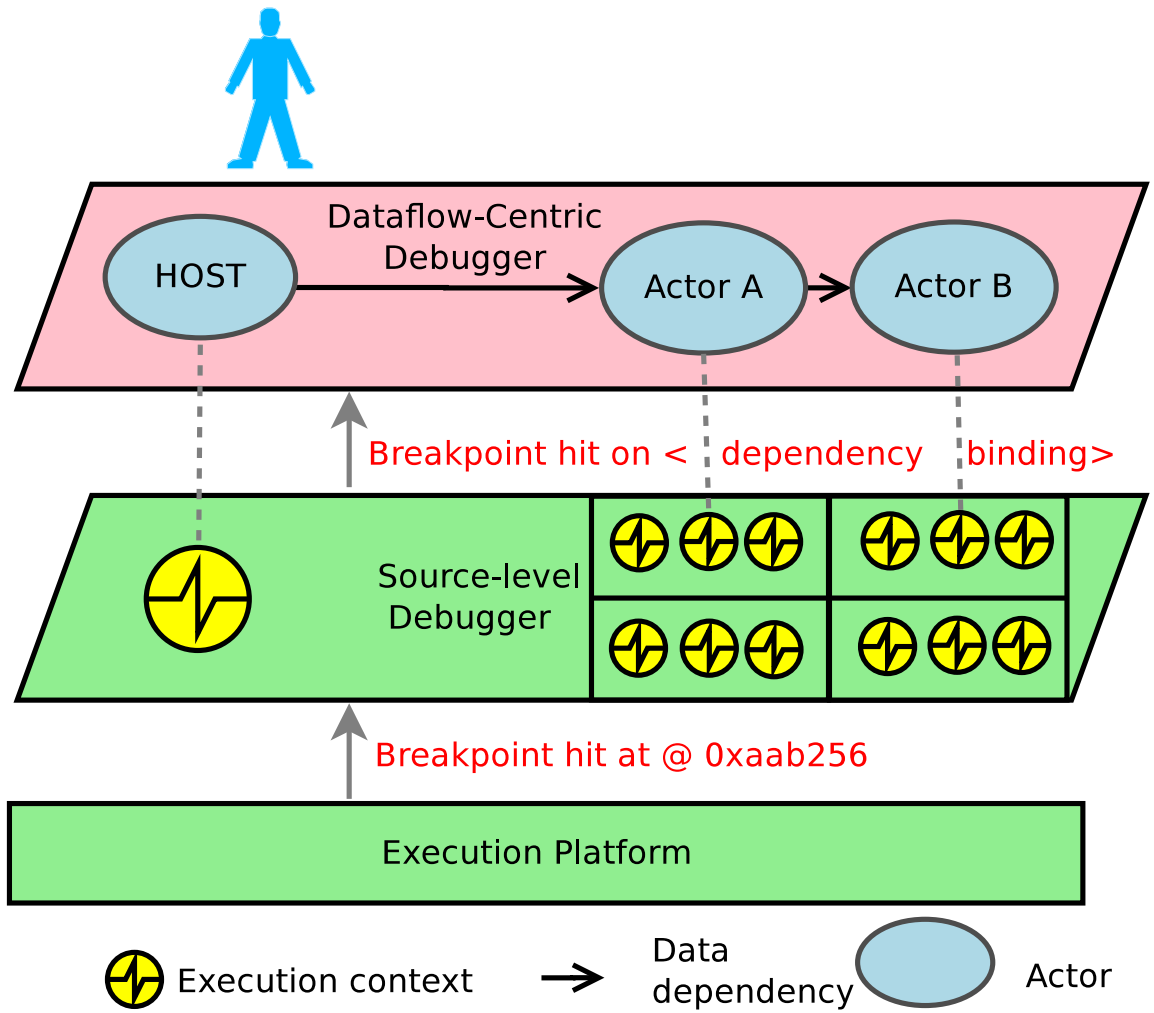
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Interpreting Execution Events

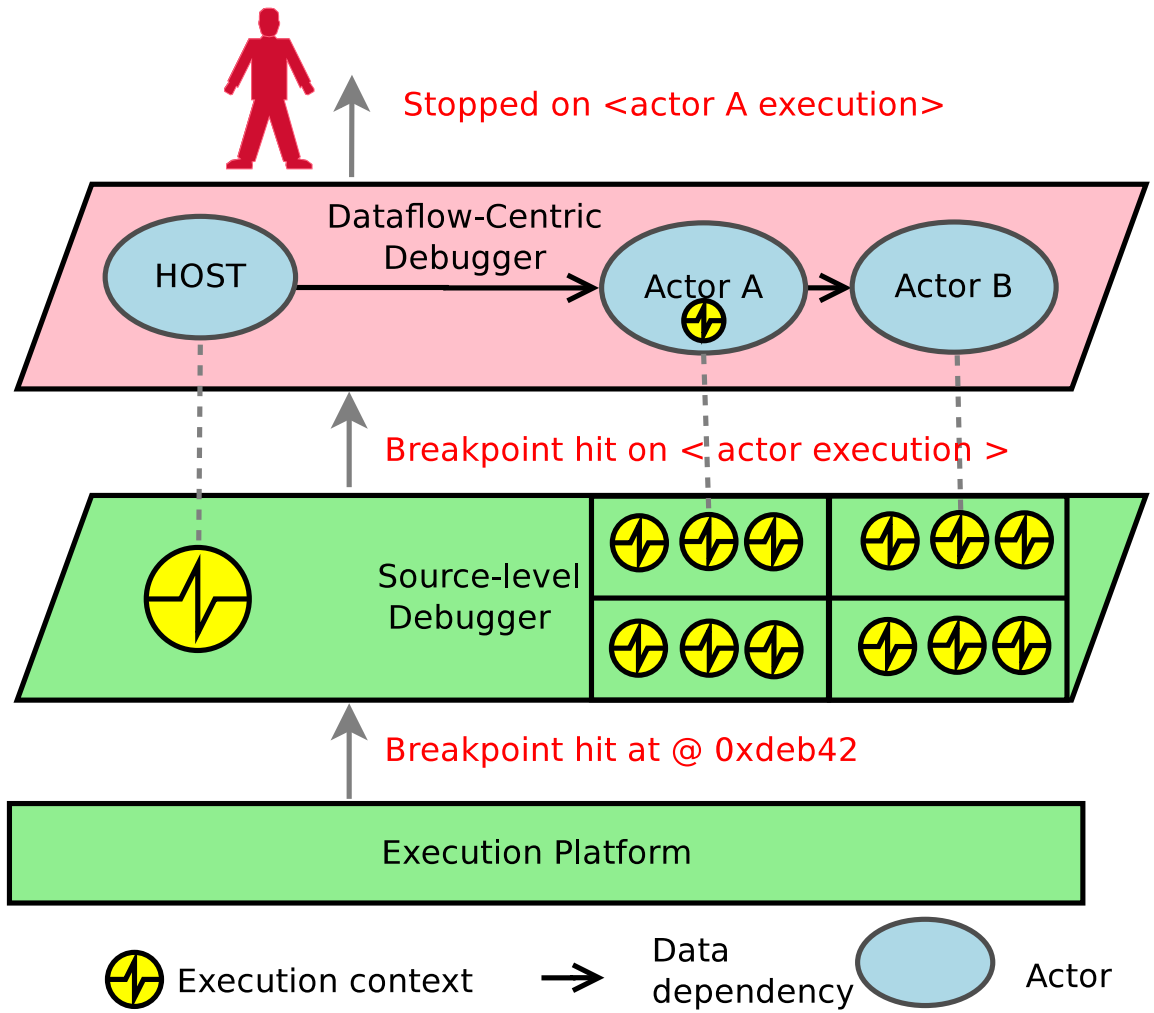
⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

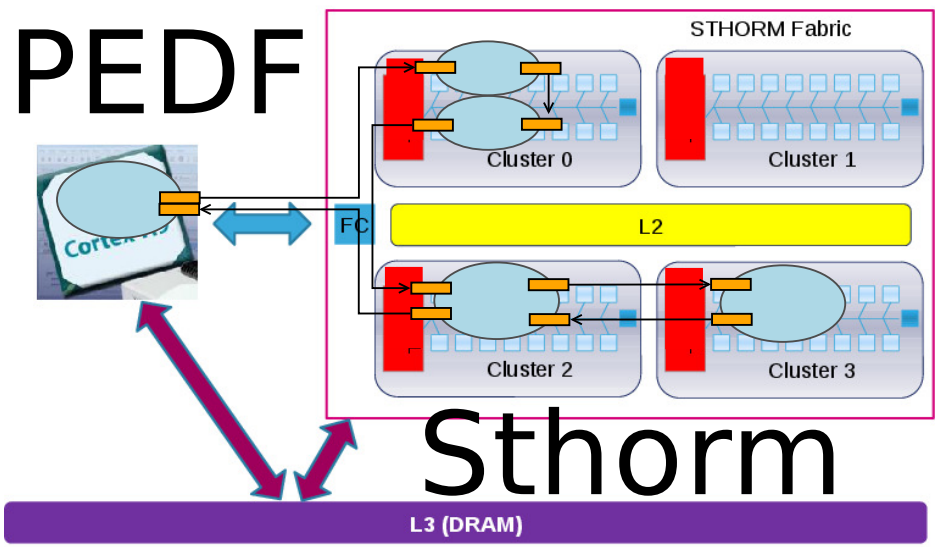
Interpreting Execution Events

⇒ Detect and interpret the exec. events of the runtime framework



Model-Centric Debugger Case-Study

Dataflow Video Decoder

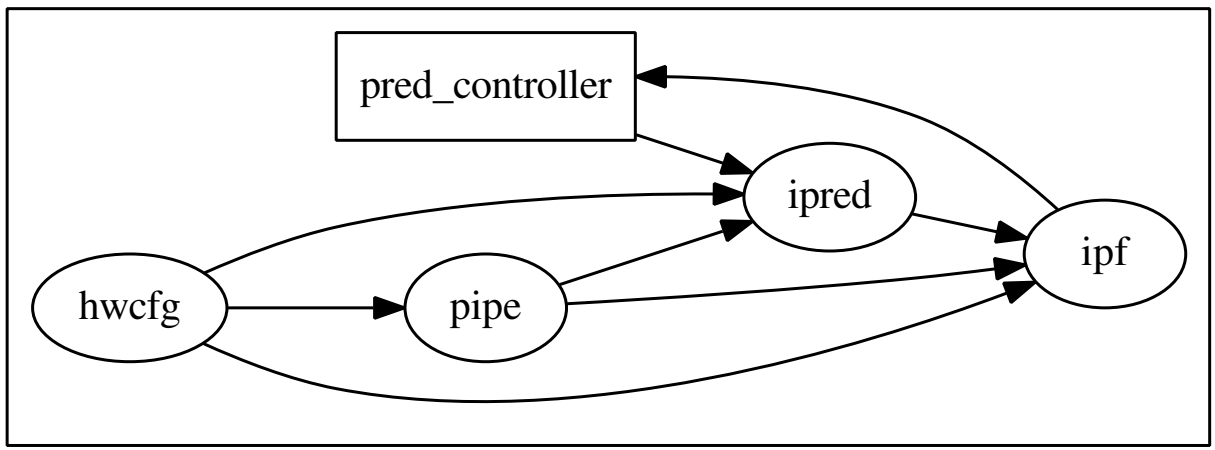


logo by bullboykennels

Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?

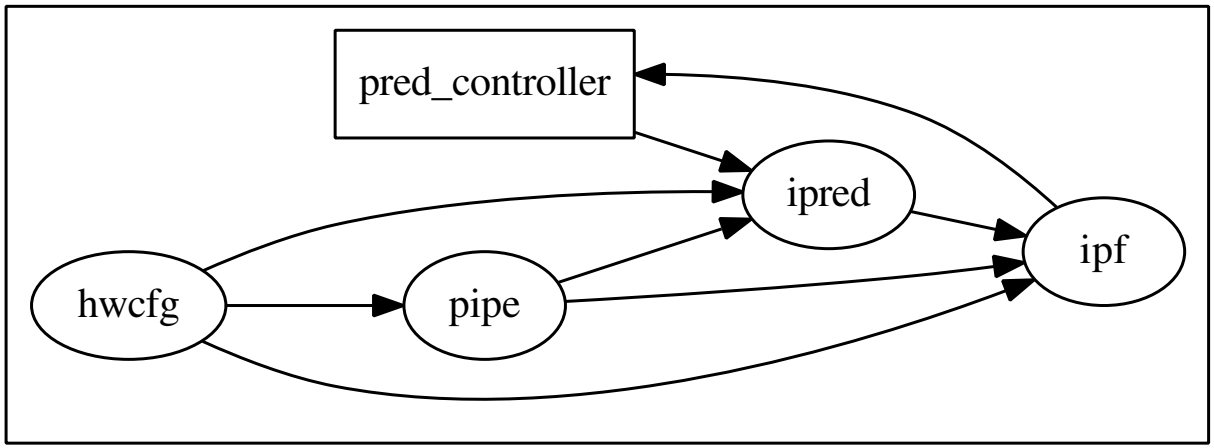
hint: not much!



(static graph provided by the compiler)

Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?

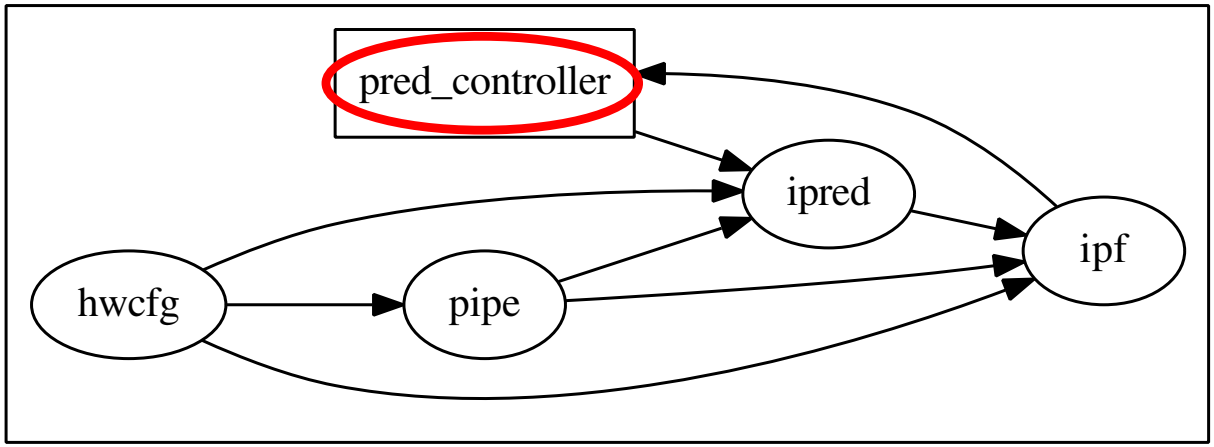


(gdb) info threads

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

Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?

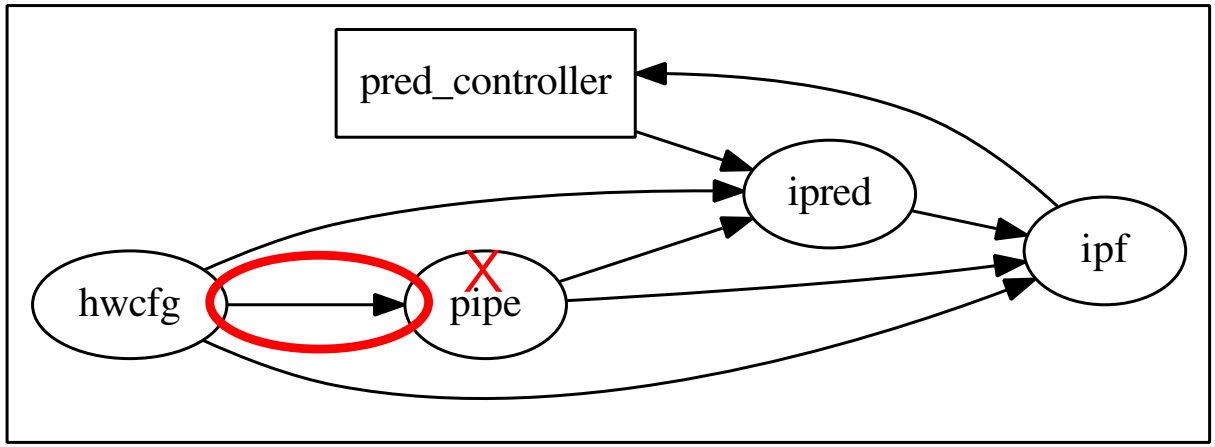


(gdb) thread apply all where

```
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 ()
```

Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can GDB help us?



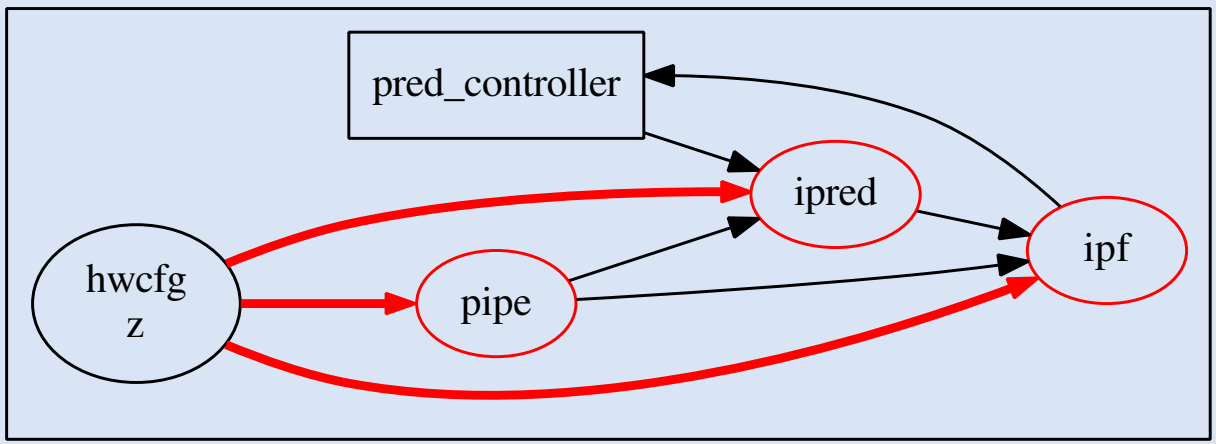
(gdb) thread apply all where

```
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)
```

Model-Centric Debugger Case-Study: Dataflow Video Decoder

The application is frozen, how can mcGDB help us?

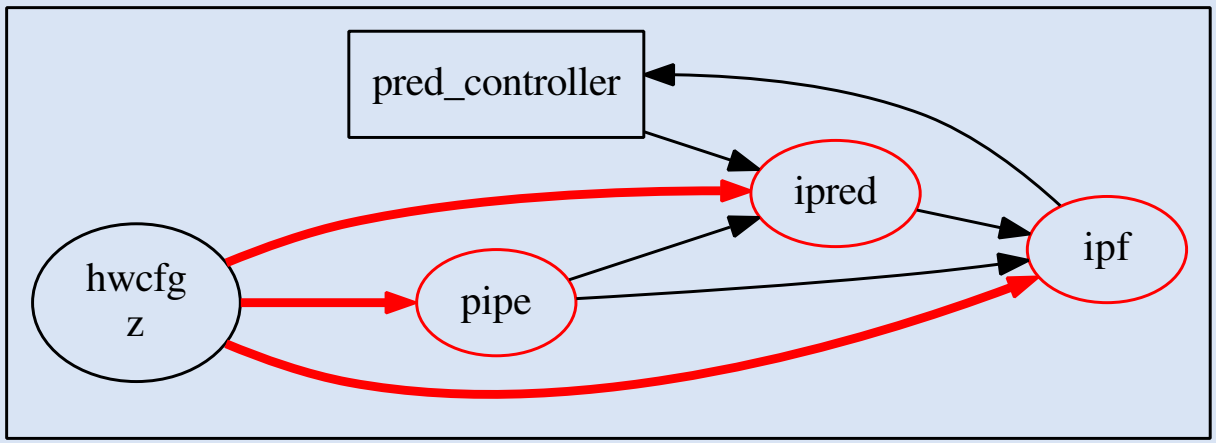
(mcgdb) info graph



Model-Centric Debugger Case-Study: 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
```

Agenda

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, applied to

- Component, dataflow and kernel (GPU) programming

Conclusions and Future Work

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

Our contribution: model-centric interactive debugging, applied to

- Component, dataflow and kernel (GPU) programming

Proof-of-concept: mcGDB, a prototype for STHORM platform

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

Conclusions and Future Work

Perspectives for programming-model centric debugging:

- Industrial side
  - Strengthen the implementation
  - Conduct extensive impact studies
- Research side
  - Apply to different programming models
  - Visualization-assisted interactive debugging
  - Enrich debugging information generated by compilers
    - work funded by NANO 2017 R&D project

STMicroelectronics
LIG
University of Grenoble

life.augmented

Programming-Model Centric Debugging for Multicore Embedded Systems

Kevin Pouget, *UJF-LIG, STMicroelectronics*
Miguel Santana, *STMicroelectronics*
Jean-François Méhaut, *UJF-CEA/LIG*

MAD Workshop'14, Athens, Greece — October 8th, 2014

Publications


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.

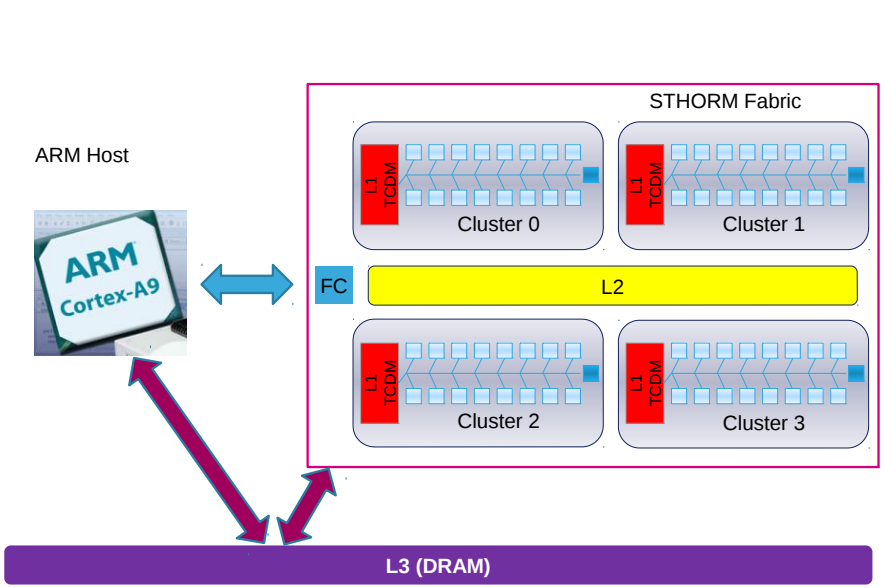

Kevin Pouget, Patricia López Cueva, Miguel Santana, and Jean-François Méhaut.
A novel approach for interactive debugging of dynamic dataflow embedded applications. In *Proceedings of the 28th Symposium On Applied Computing (SAC)*, pages 1547–1549, Coimbra, Portugal, apr 2013.

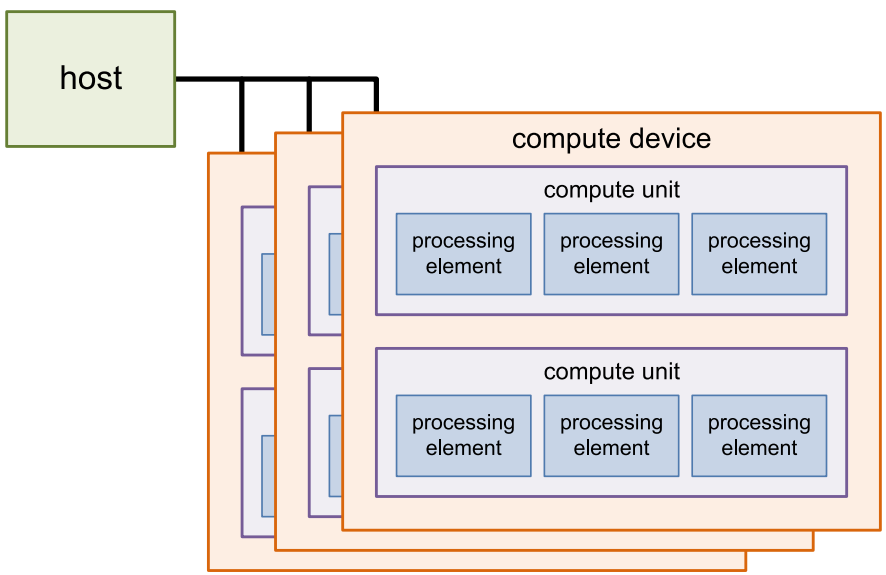
Conclusions and Future Work

Interact with the Abstract Machine

ST STHORM Platform — our reference MPSoC

- CPU + 4 clusters × 16 lightweight/energy-efficient cores
  - ± dedicated hardware accelerators
- GPU-like architecture



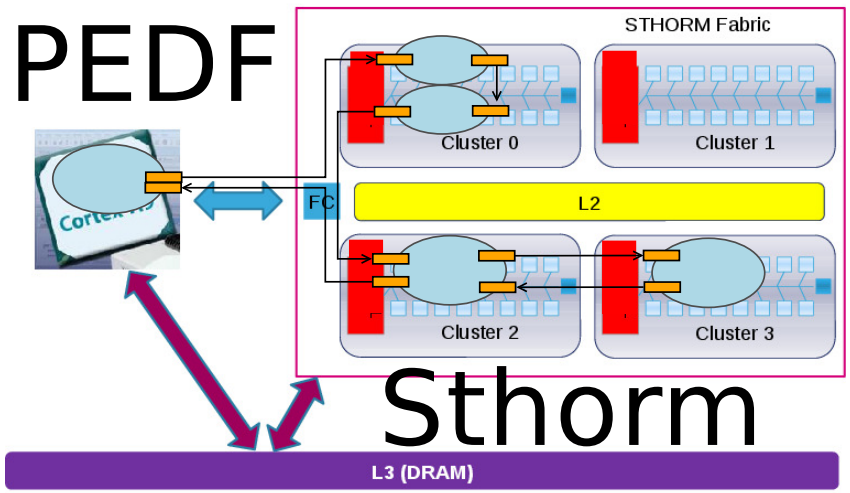


Björn König, Wikimedia

How to program such complex architectures?

Conclusions and Future Work

Dataflow Video Decoder



logo by bullboykennels

Dataflow Environment (PEDF)

- Dynamic dataflow programming
- Good for multimedia application
- No verification/validation help
- Heterogeneous computing:
  - actors ⇒ HW accelerators

- Flexible video decoding standard
  - for HD television, blu-ray disks, broadcast, telephony, ...
- Good dataflow decomposition
- Developed to validate PEDF design