STMicroelectronics
University of Grenoble

Debugging Component-Based Embedded Applications.

Kevin Pouget, Miguel Santana, Vania Marangozova-Martin
and Jean-François Mehaut

Context

Embedded System Development

- High-resolution multimedia app. $\Rightarrow$ high performance expectations.
  - H.265 HEVC
  - augmented reality,
  - ...
- Sharp time-to-market constraints

$\Rightarrow$ Important demand for

- powerful parallel architectures
  - MultiProcessor on Chip (MPSoC)
- convenient programming methodologies
  - Component-Based Software Engineering
- efficient verification and validation tools
  - Our problematic

Context

MultiProcessor on Chip (MPSoC)

- Parallel architecture
  - more difficult to program
- Maybe heterogeneous
  - hardware accelerators,
  - GPU-like accelerators (OS-less)
- Embedded system
  - constrained environment,
  - on-board debugging complicated
    $\rightarrow$ performance debugging only
  - limited-scale functional debugging on simulators

Context

Component-Based Software Engineering

- Focus on design of independent building blocks
- Applications built with interconnected components
- Allows the adaptation of the application architecture according to runtime constraints
- Runnable components able to exploit MPSoC parallelism

Agenda

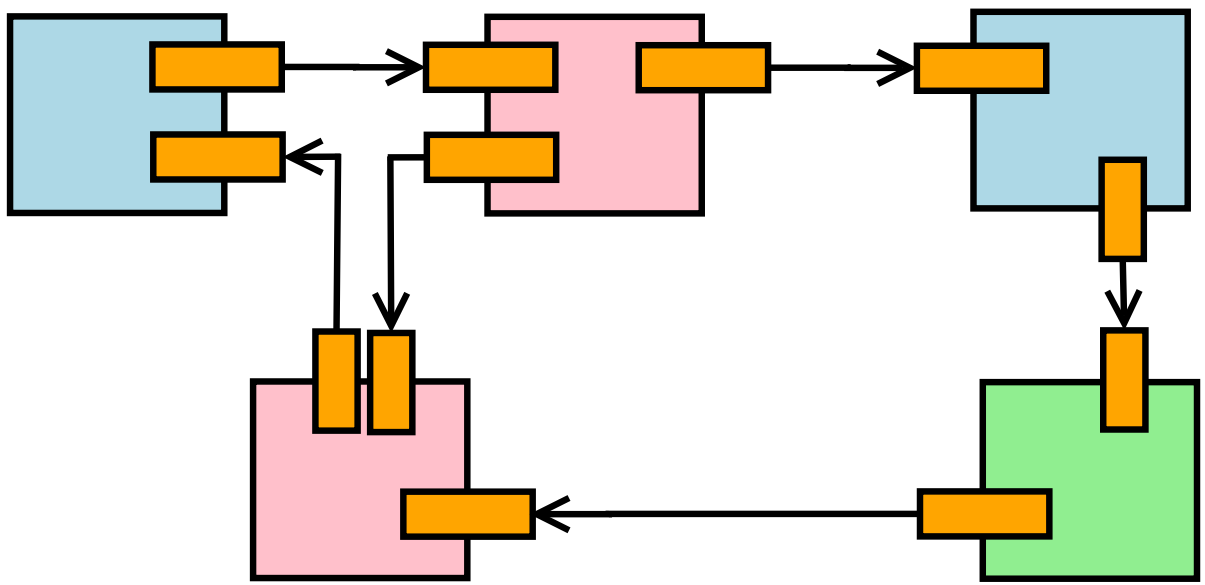
1. Component Debugging Challenges
2. Component-Aware Interactive Debugging
3. Feature Demonstration
4. Conclusion

Agenda

Component Debugging Challenges

Component-based applications are dynamic

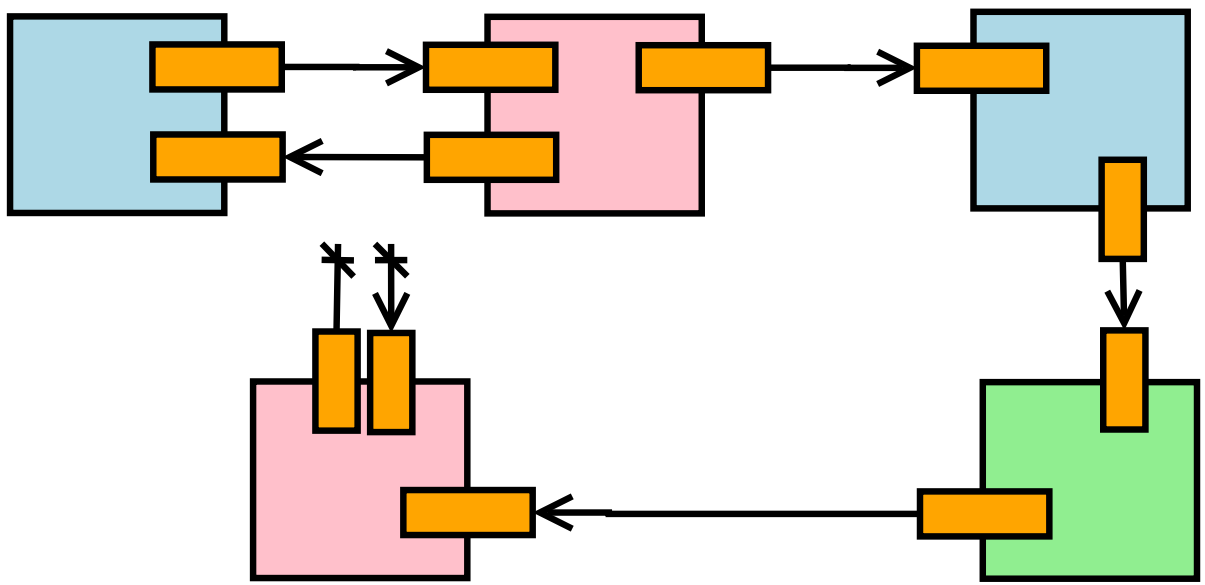
- various set of components deployed during the execution
- components are dynamically inter-connected



Component Debugging Challenges

Component-based applications are dynamic

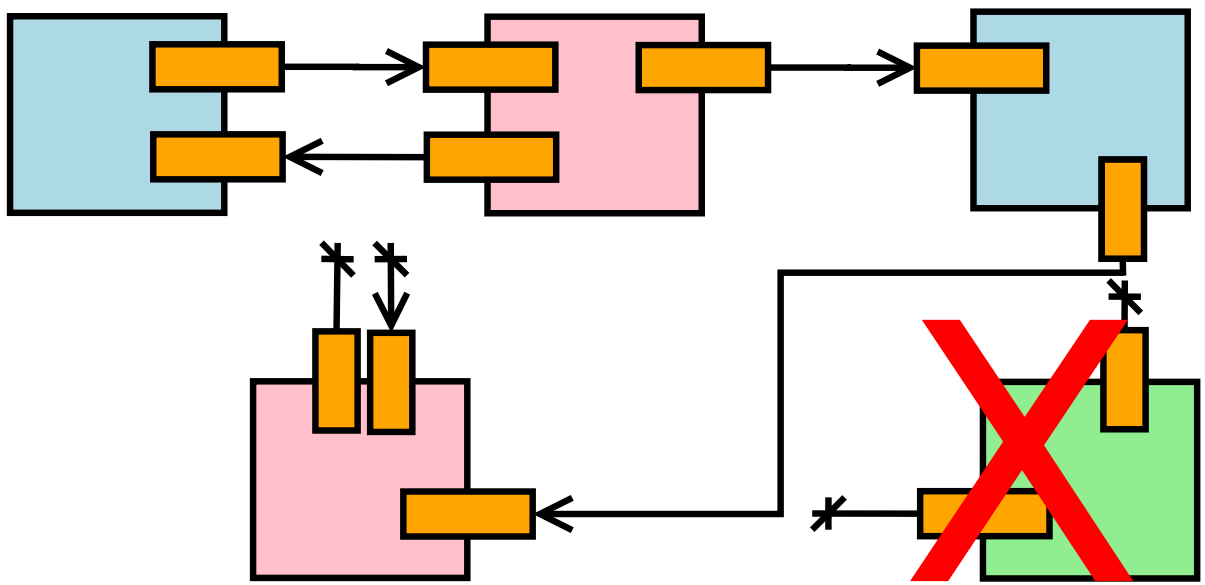
- various set of components deployed during the execution
- components are dynamically inter-connected



Component Debugging Challenges

Component-based applications are dynamic

- various set of components deployed during the execution
- components are dynamically inter-connected



Component Debugging Challenges

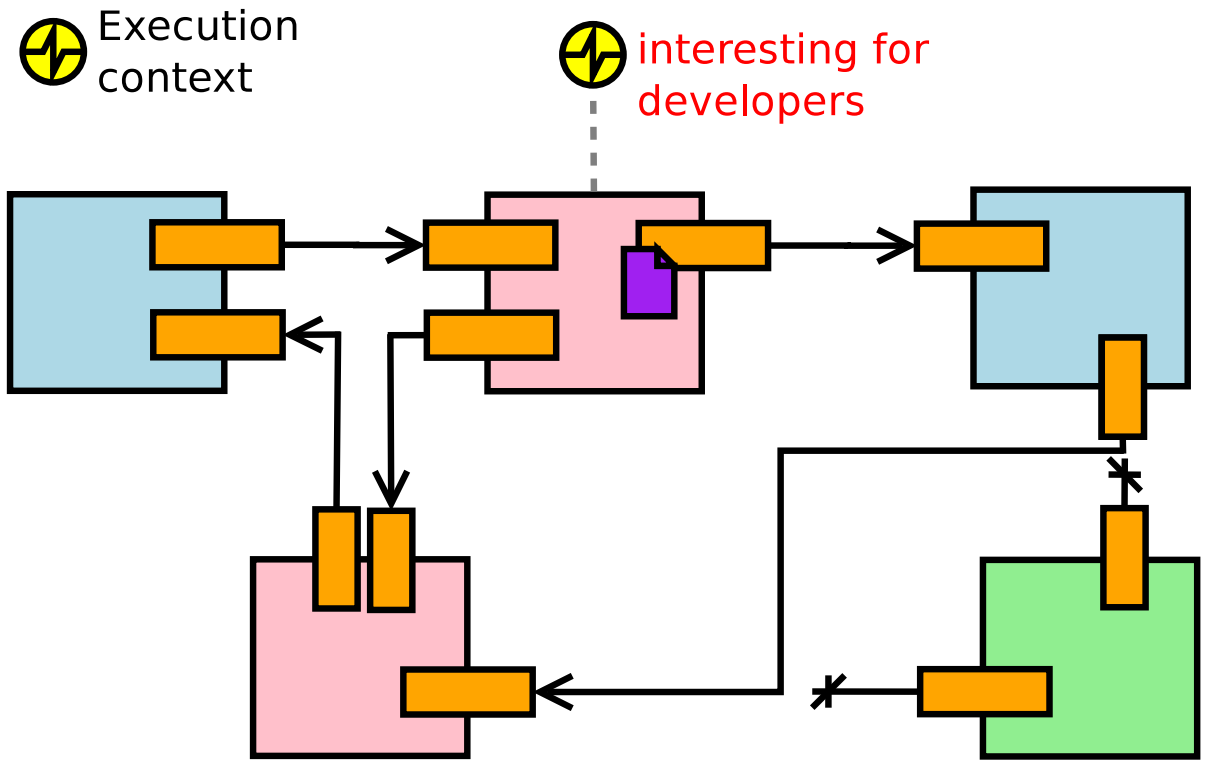
Components interact with one another

- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution

Component Debugging Challenges

Components interact with one another

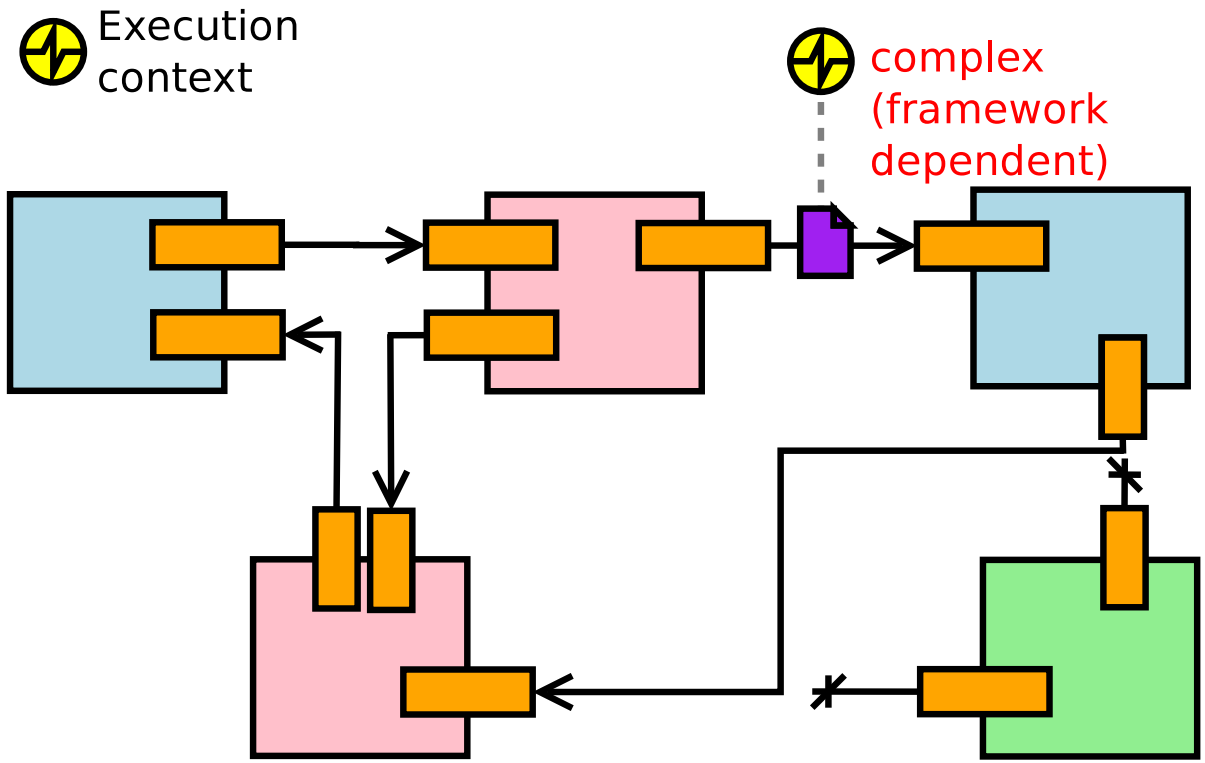
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

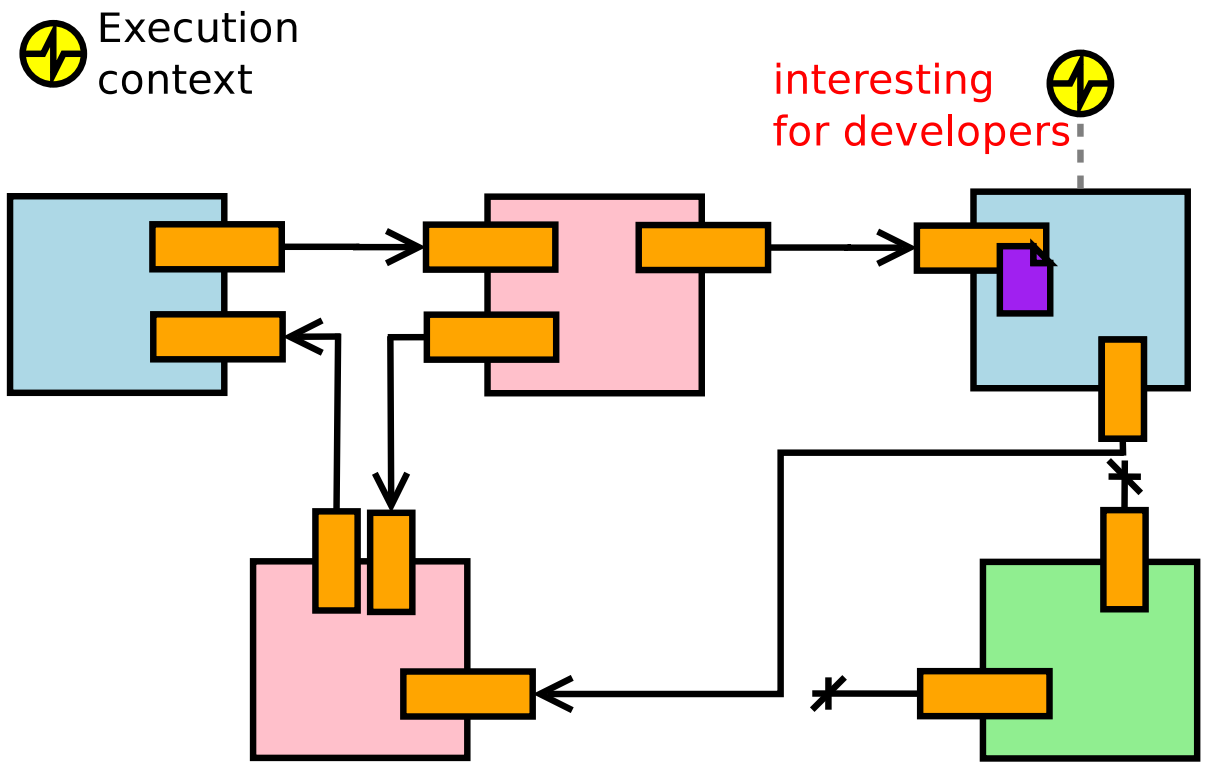
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

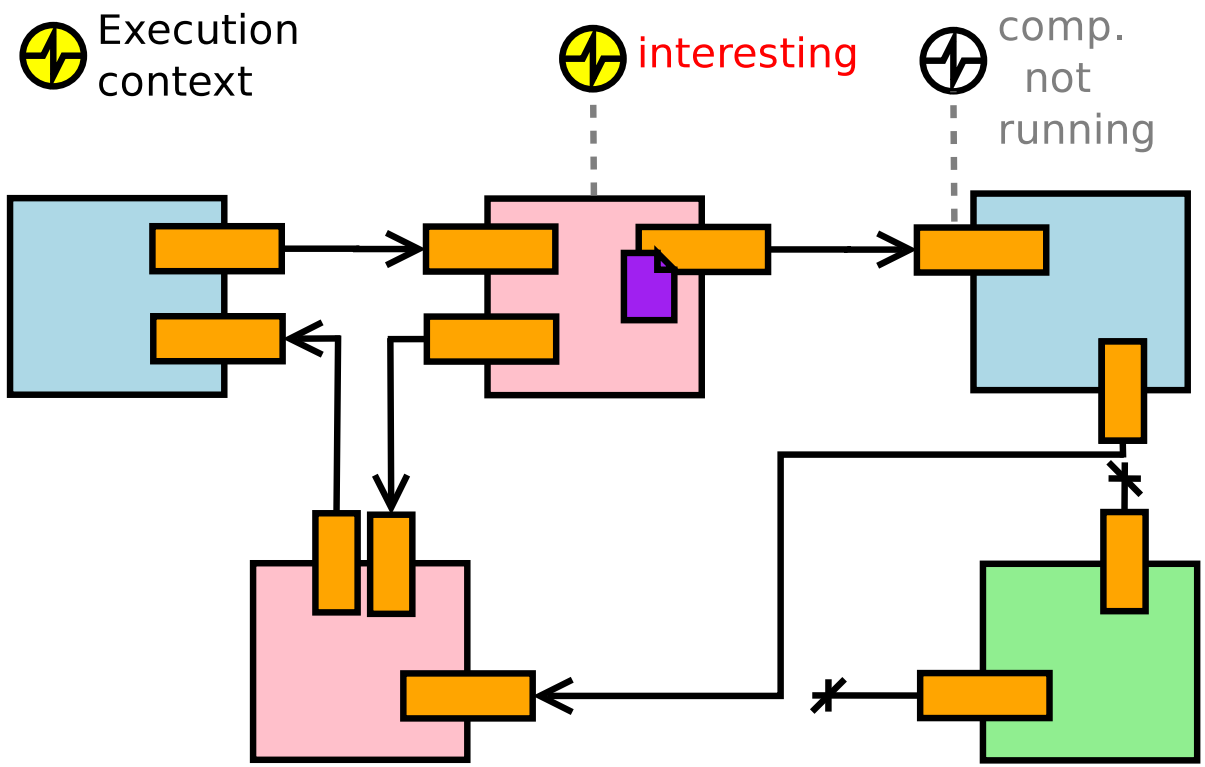
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

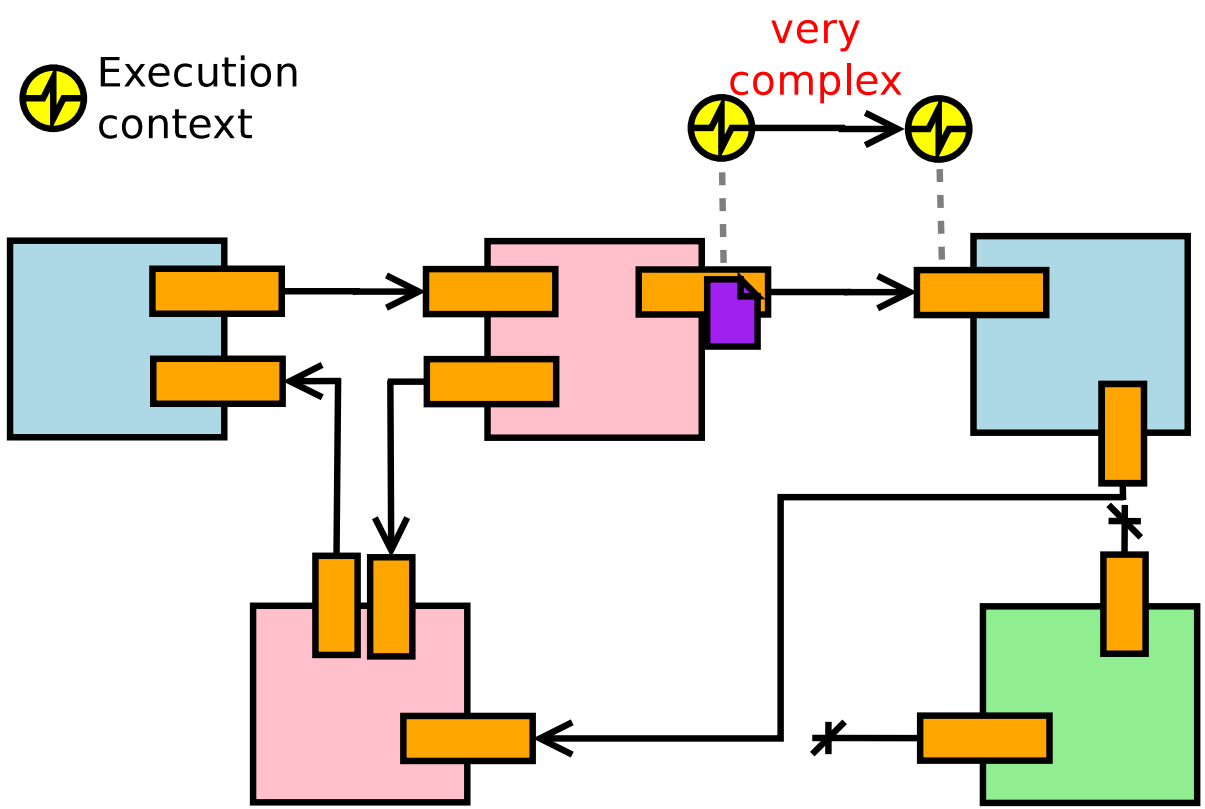
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

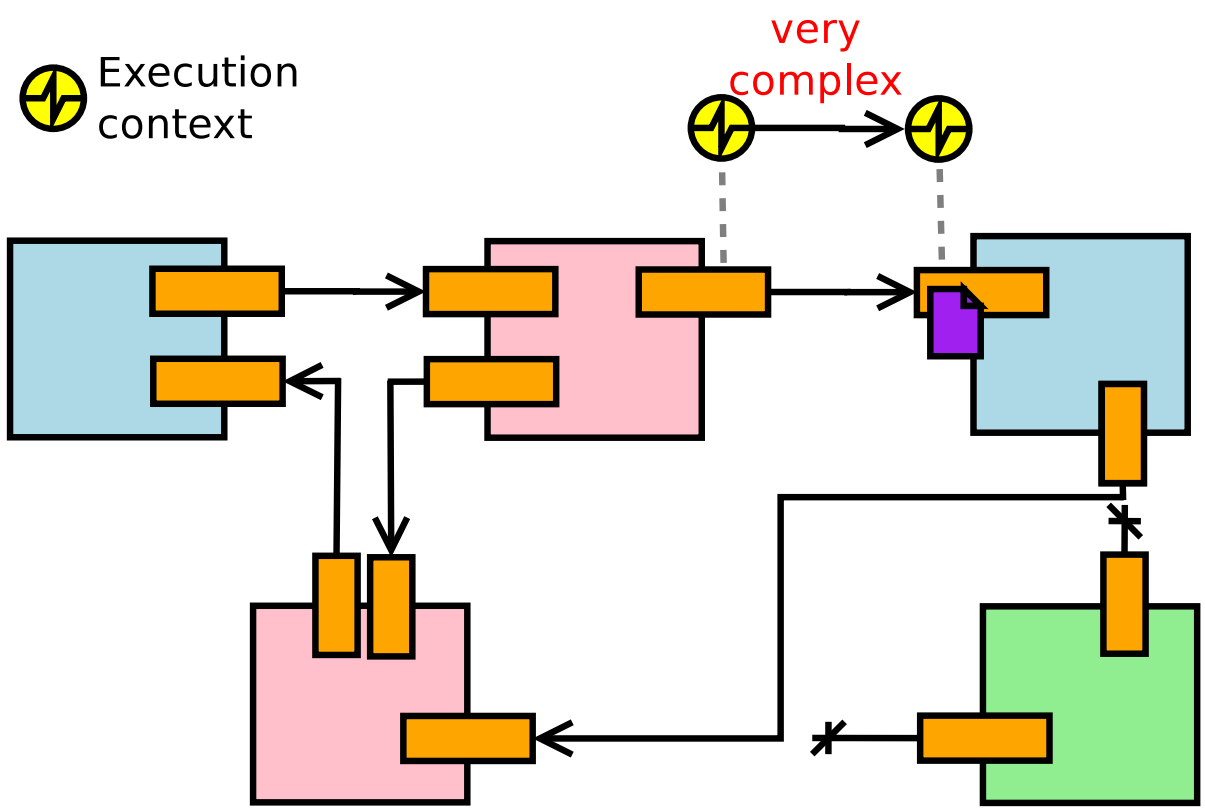
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

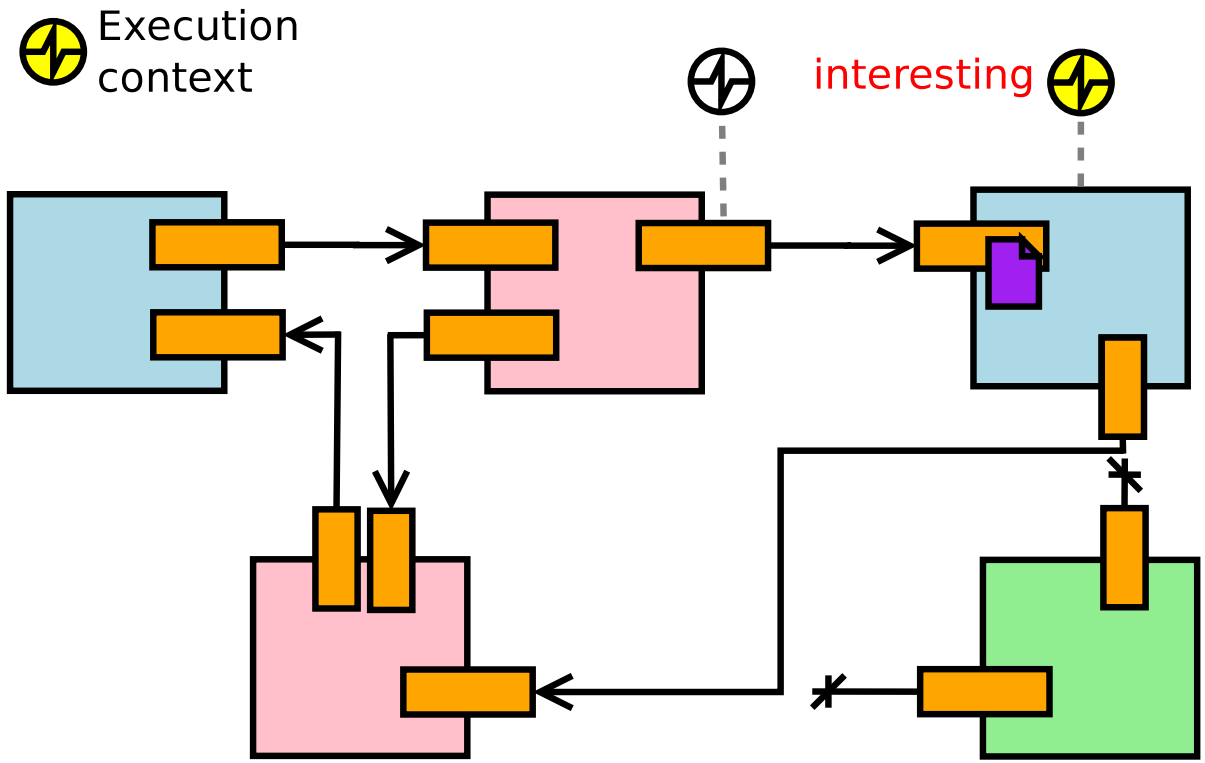
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Components interact with one another

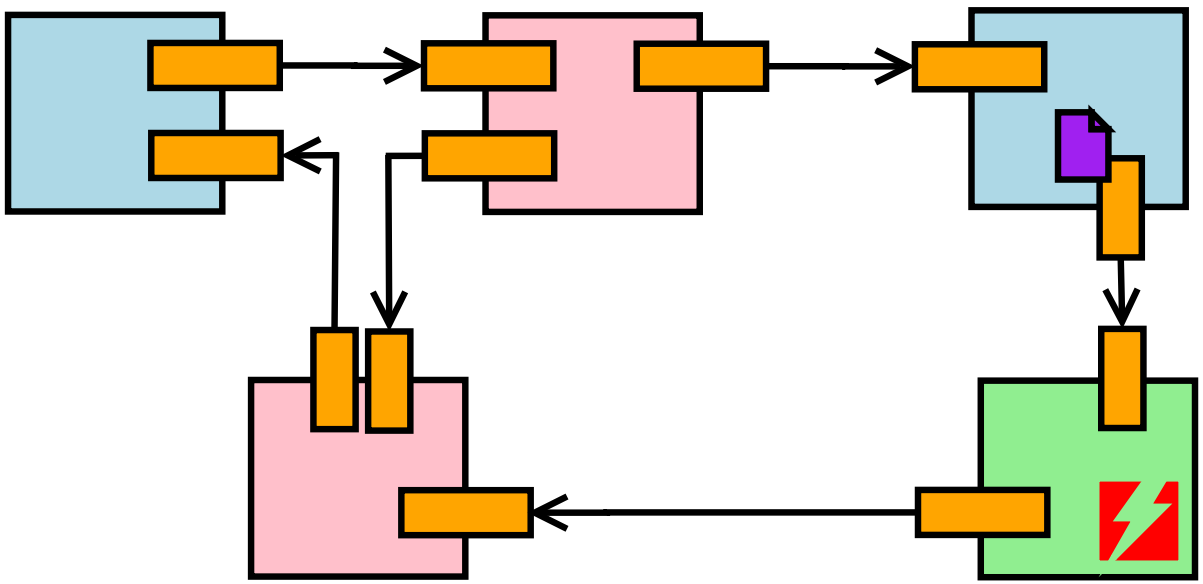
- their execution is driven by interface communications
- complex framework-dependent steps between an interface call and its execution



Component Debugging Challenges

Information flows over the components

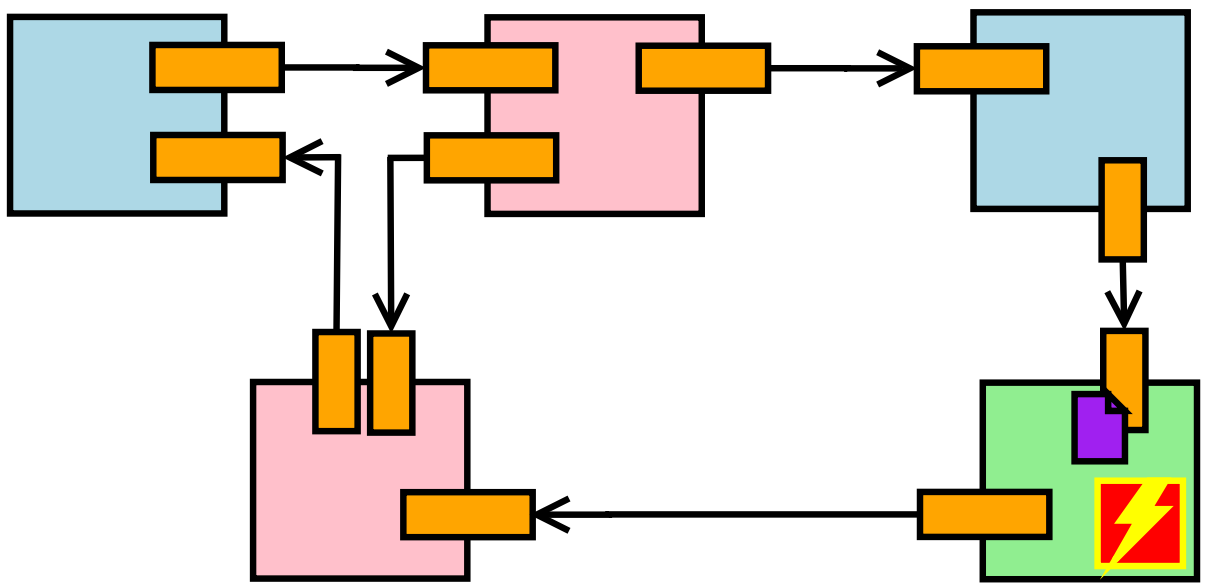
- a corrupted data may be carried over various component before triggering a visible error



Component Debugging Challenges

Information flows over the components

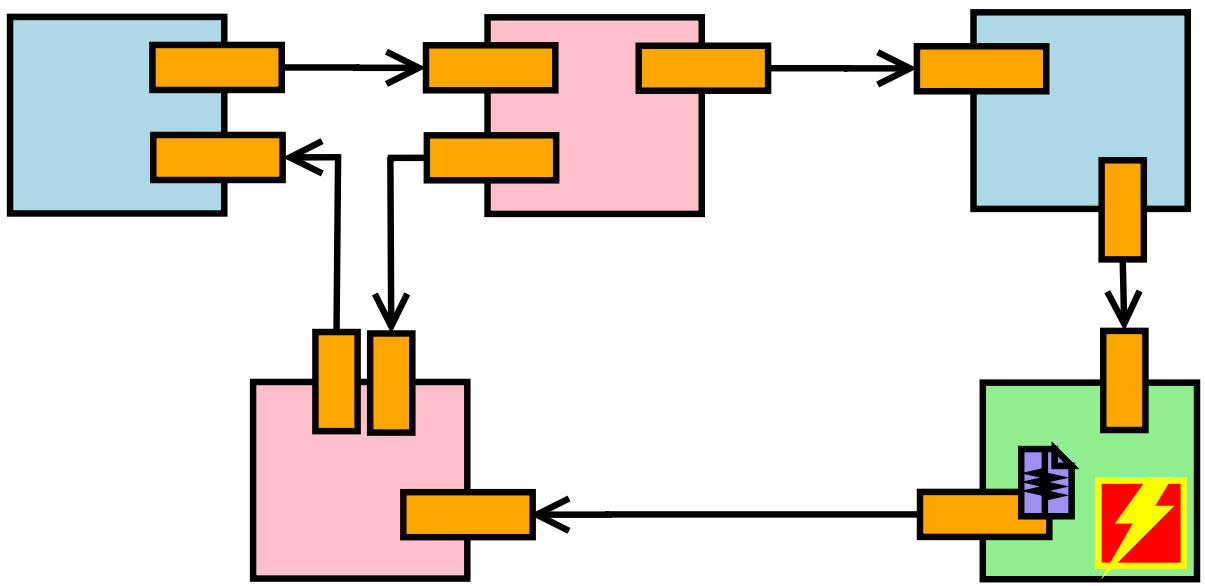
- a corrupted data may be carried over various component before triggering a visible error



Component Debugging Challenges

Information flows over the components

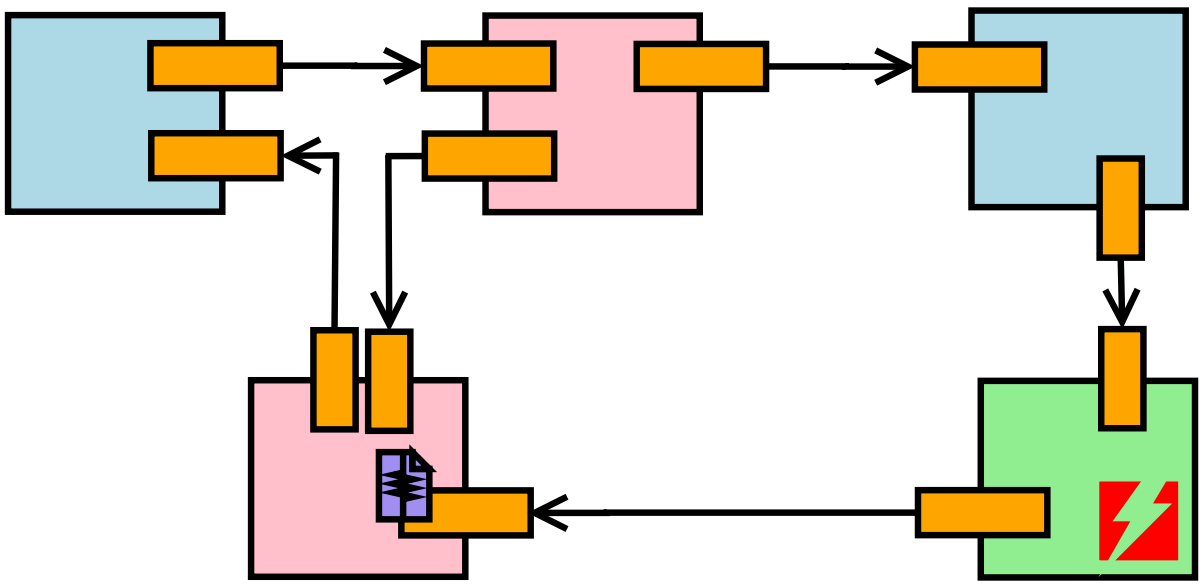
- a corrupted data may be carried over various component before triggering a visible error



Component Debugging Challenges

Information flows over the components

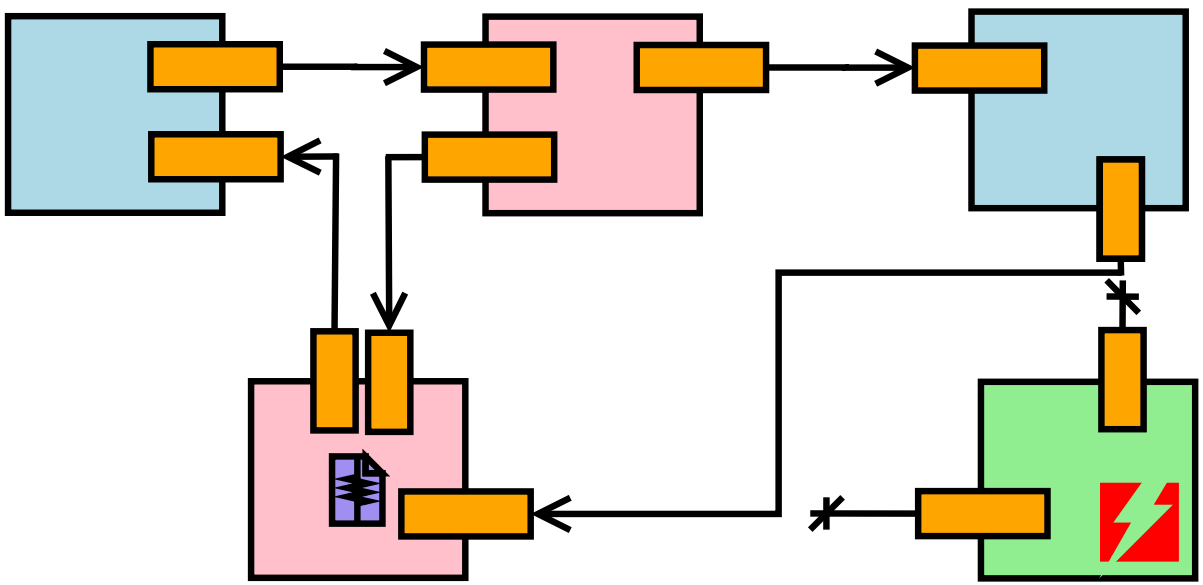
- a corrupted data may be carried over various component before triggering a visible error



Component Debugging Challenges

Information flows over the components

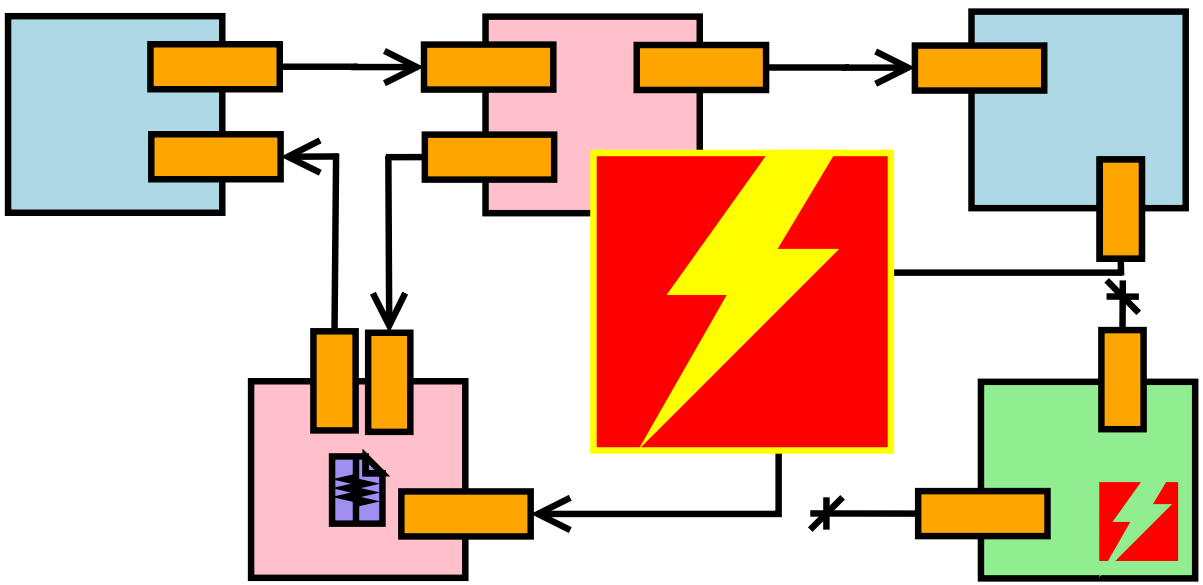
- a corrupted data may be carried over various component before triggering a visible error



Component Debugging Challenges

Information flows over the components

- a corrupted data may be carried over various component before triggering a visible error



Agenda

Component-Aware Interactive Debugging

Objective: Bring the debugger closer to the component model

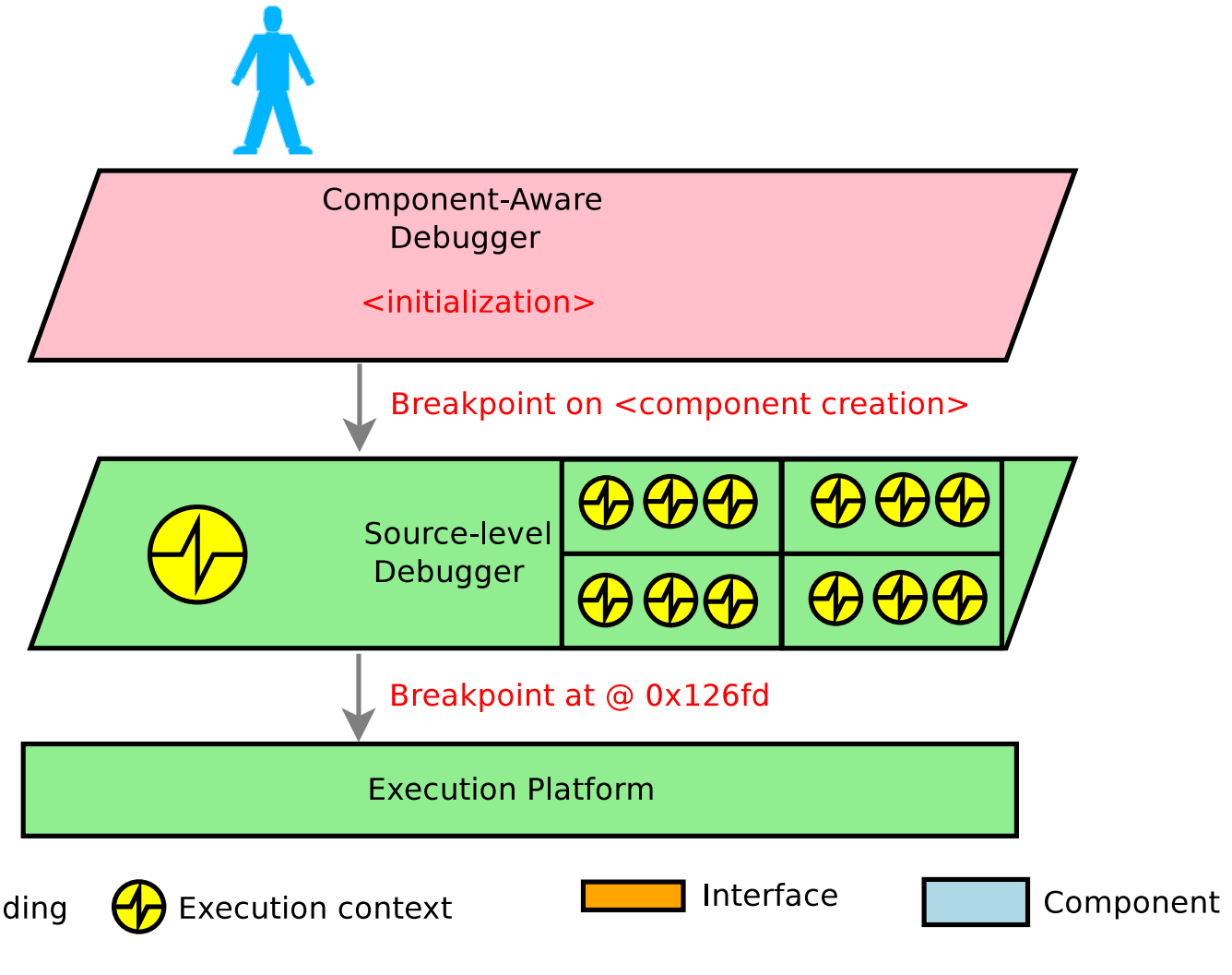
- Show application architecture evolutions
  - component deployment
  - interface binding
  - ...
- Follow the execution flow(s) over the component graph
  - runnable component execution,
  - execution triggered by an interface call
  - ...
- Track inter-component data exchanges
  - payload contents
  - data paths
  - ...

Result: high-level primitives for execution control and state examination

Component-Aware Interactive Debugging

Implementation

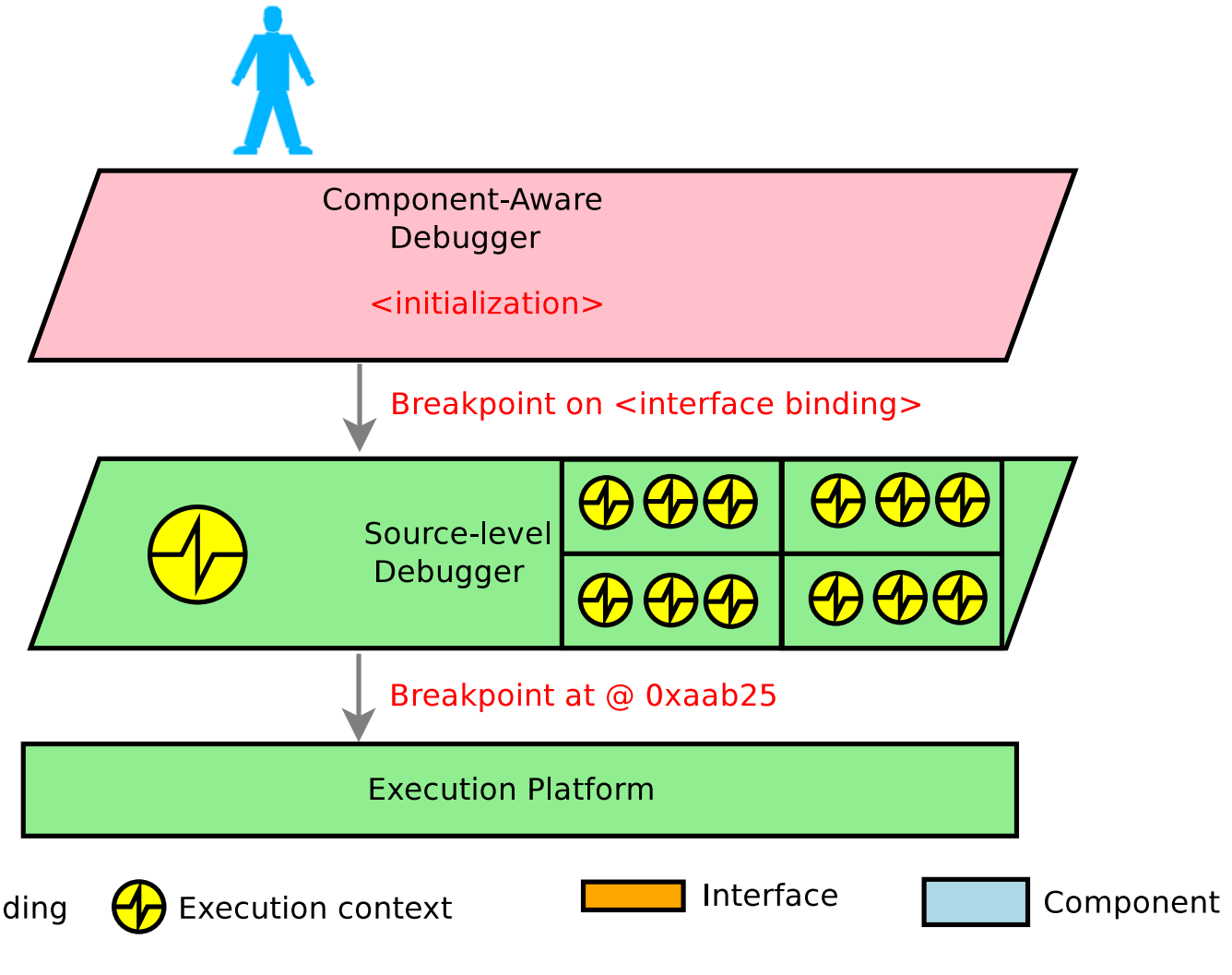
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

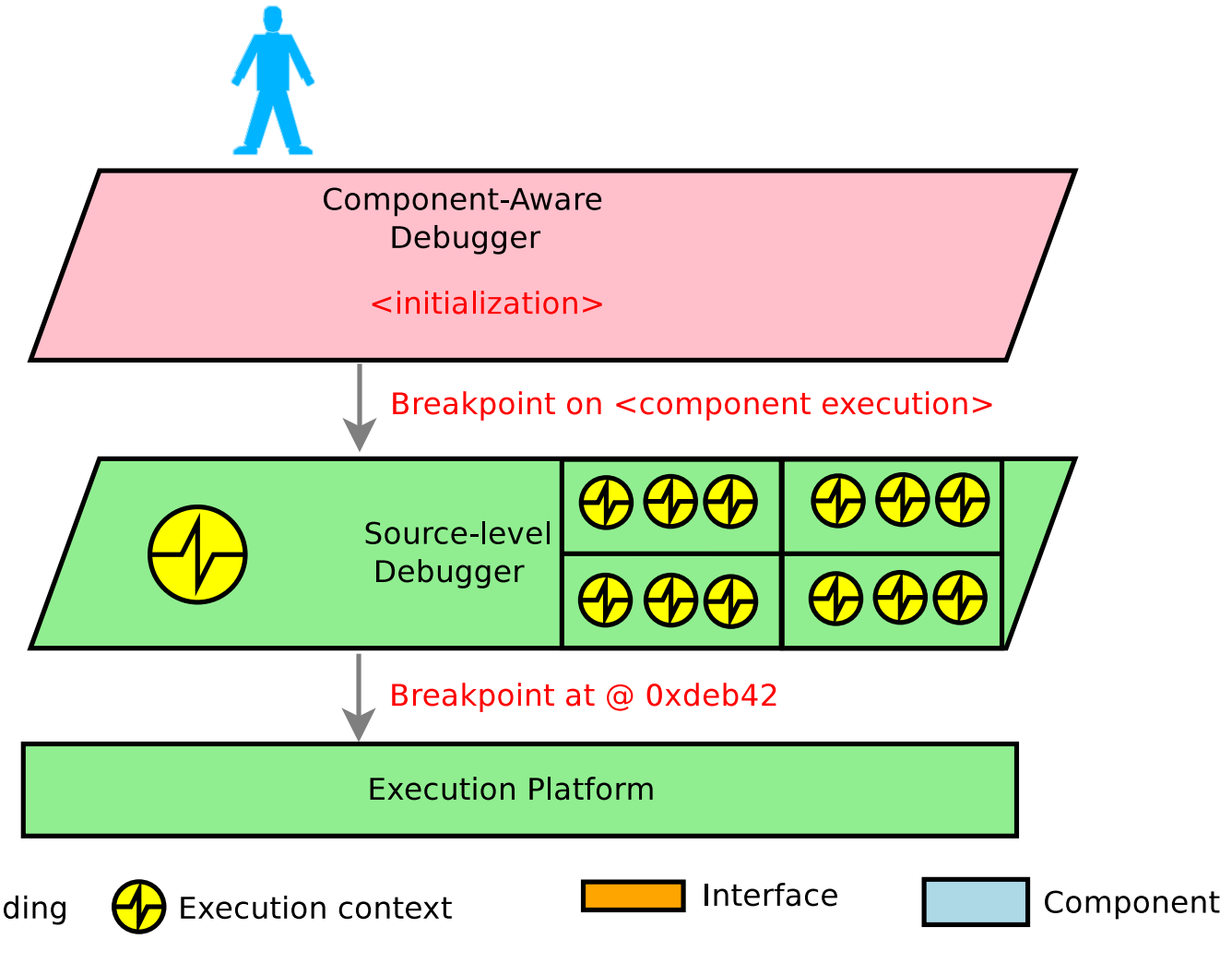
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

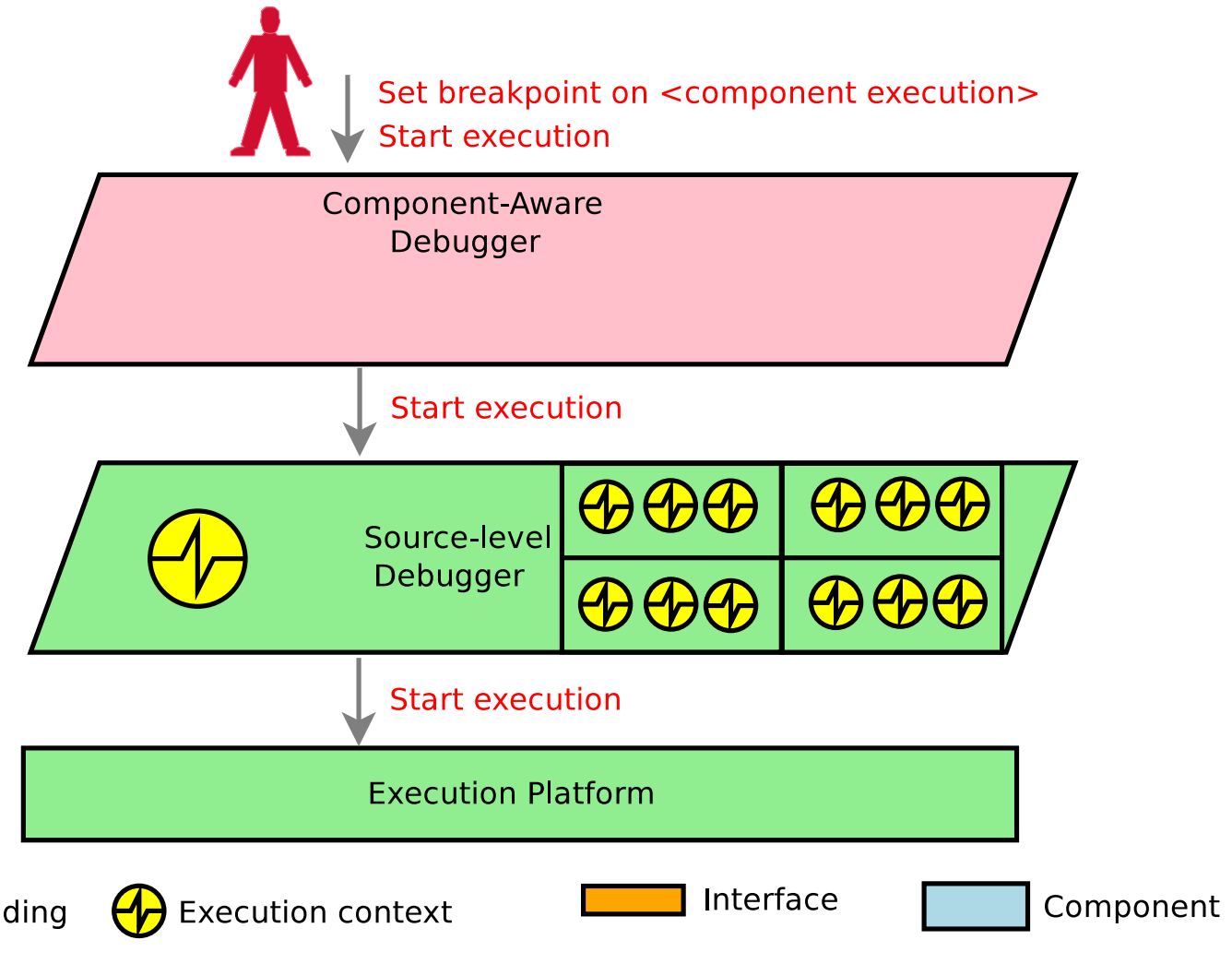
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

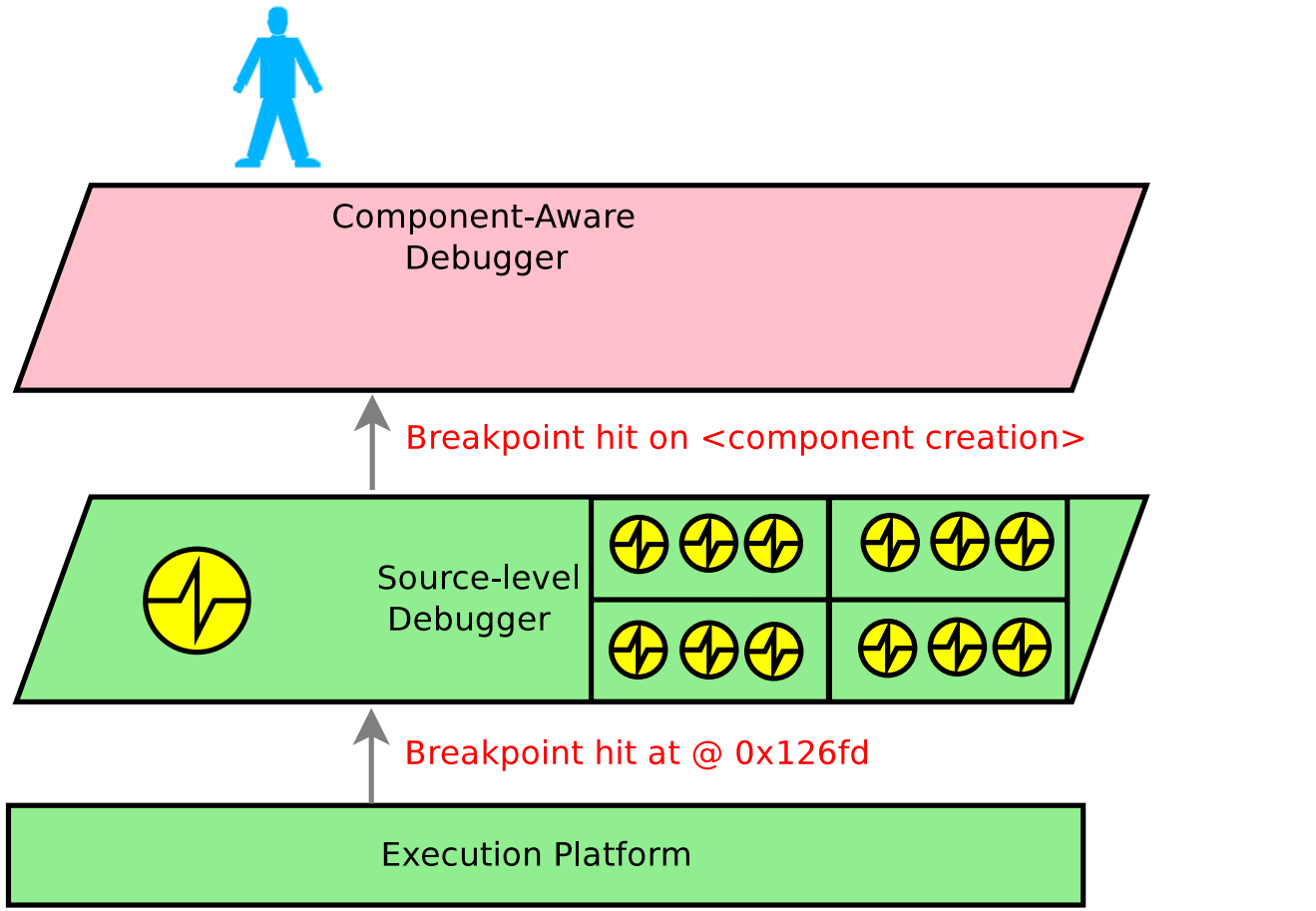
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

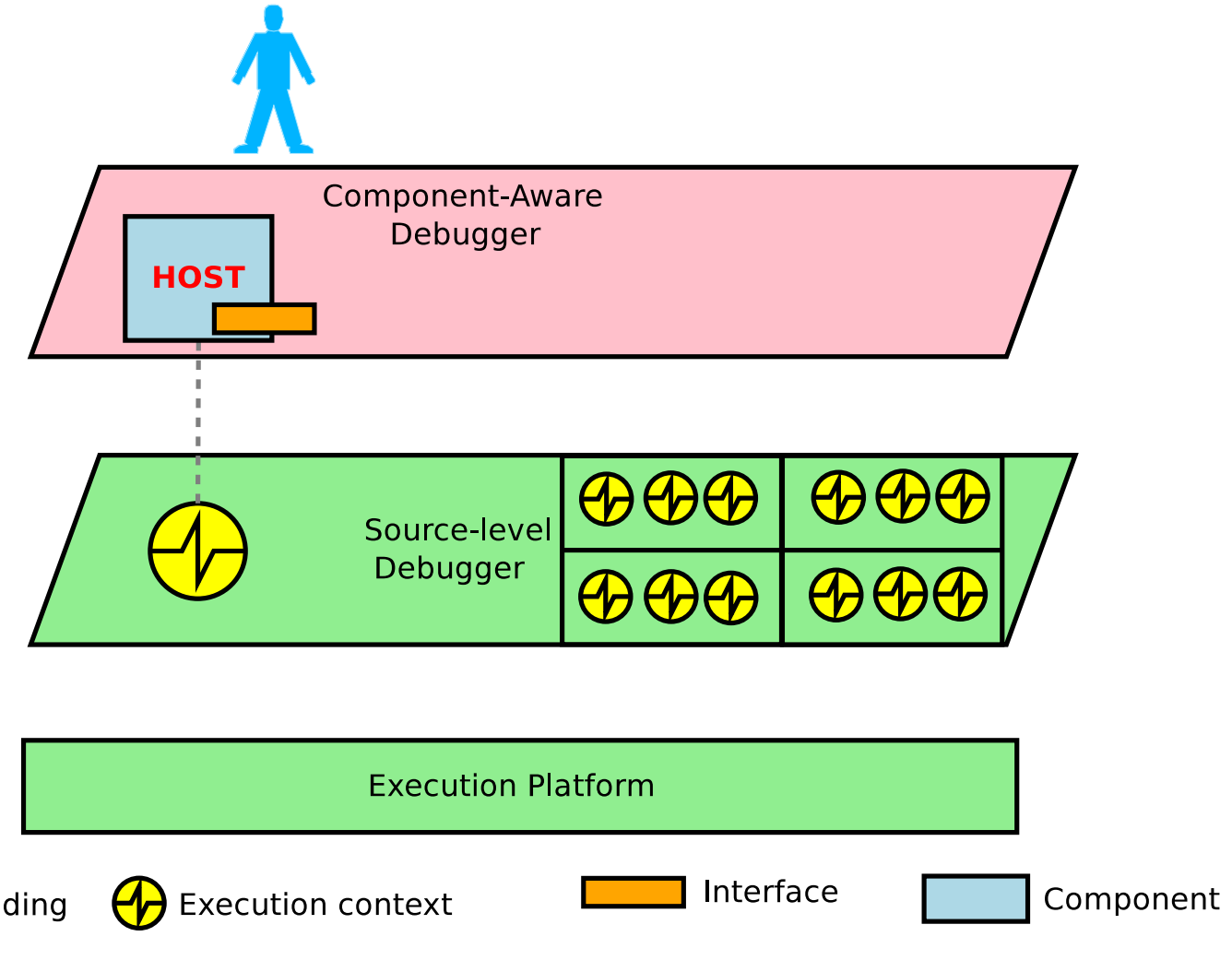
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

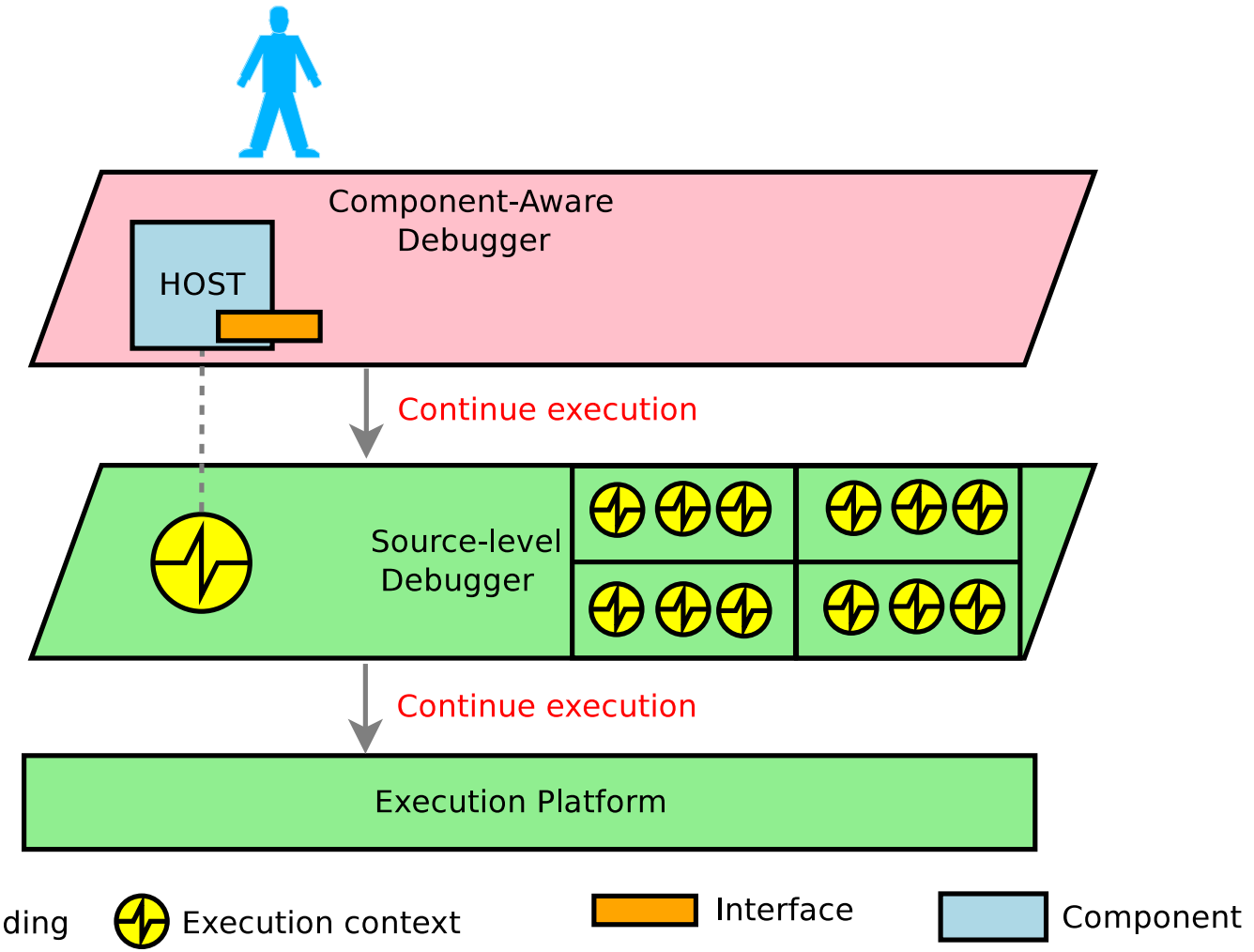
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

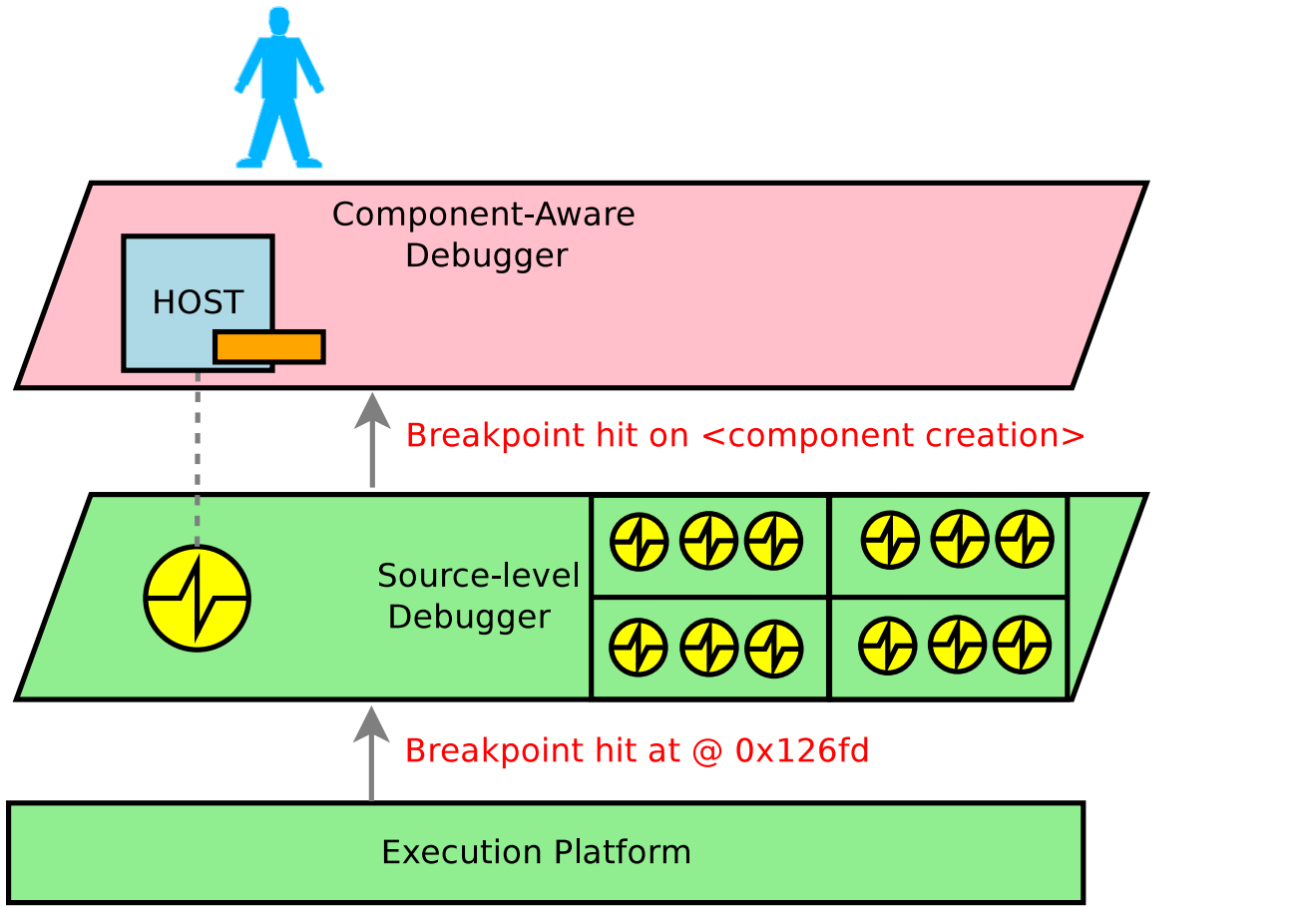
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

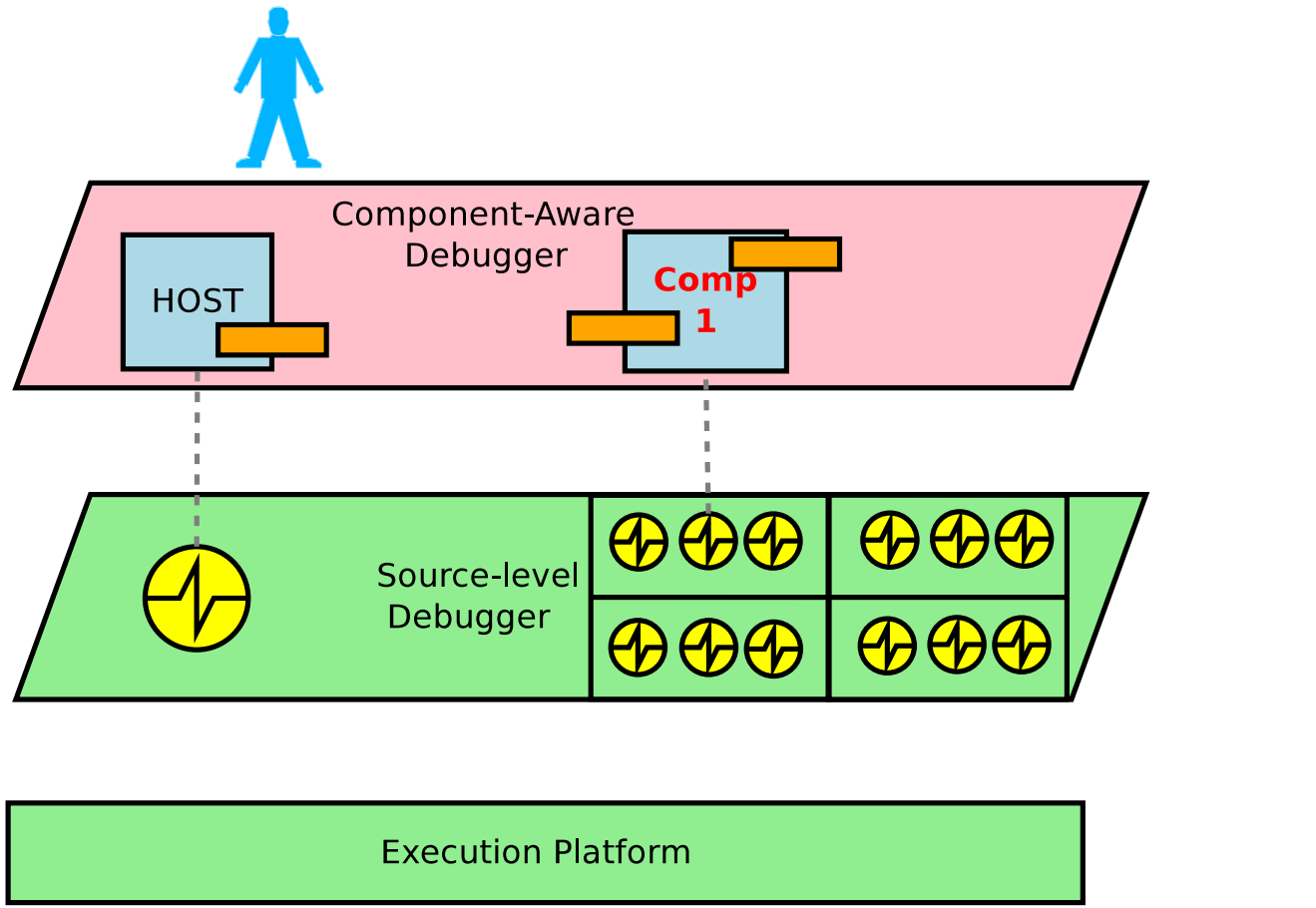
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

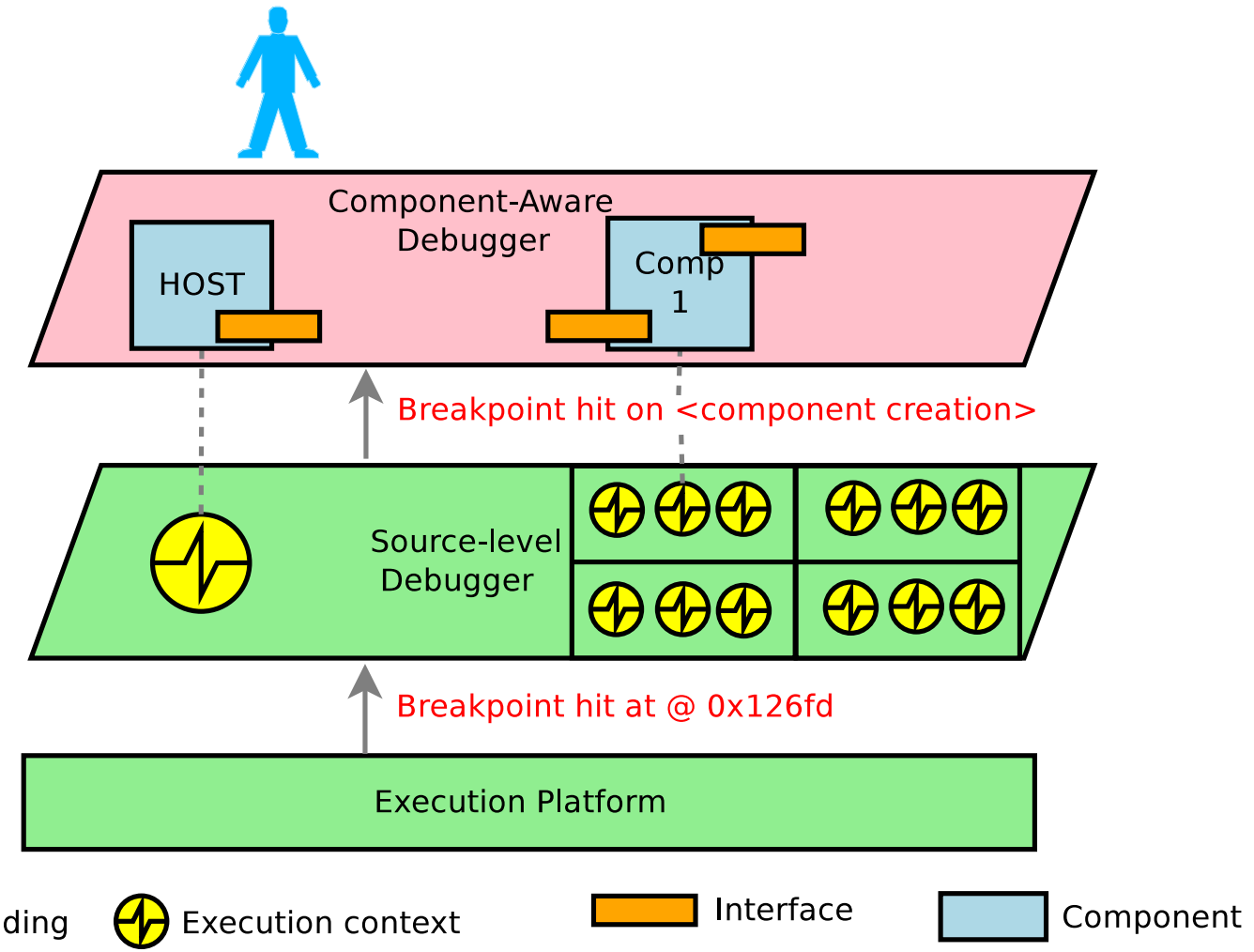
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

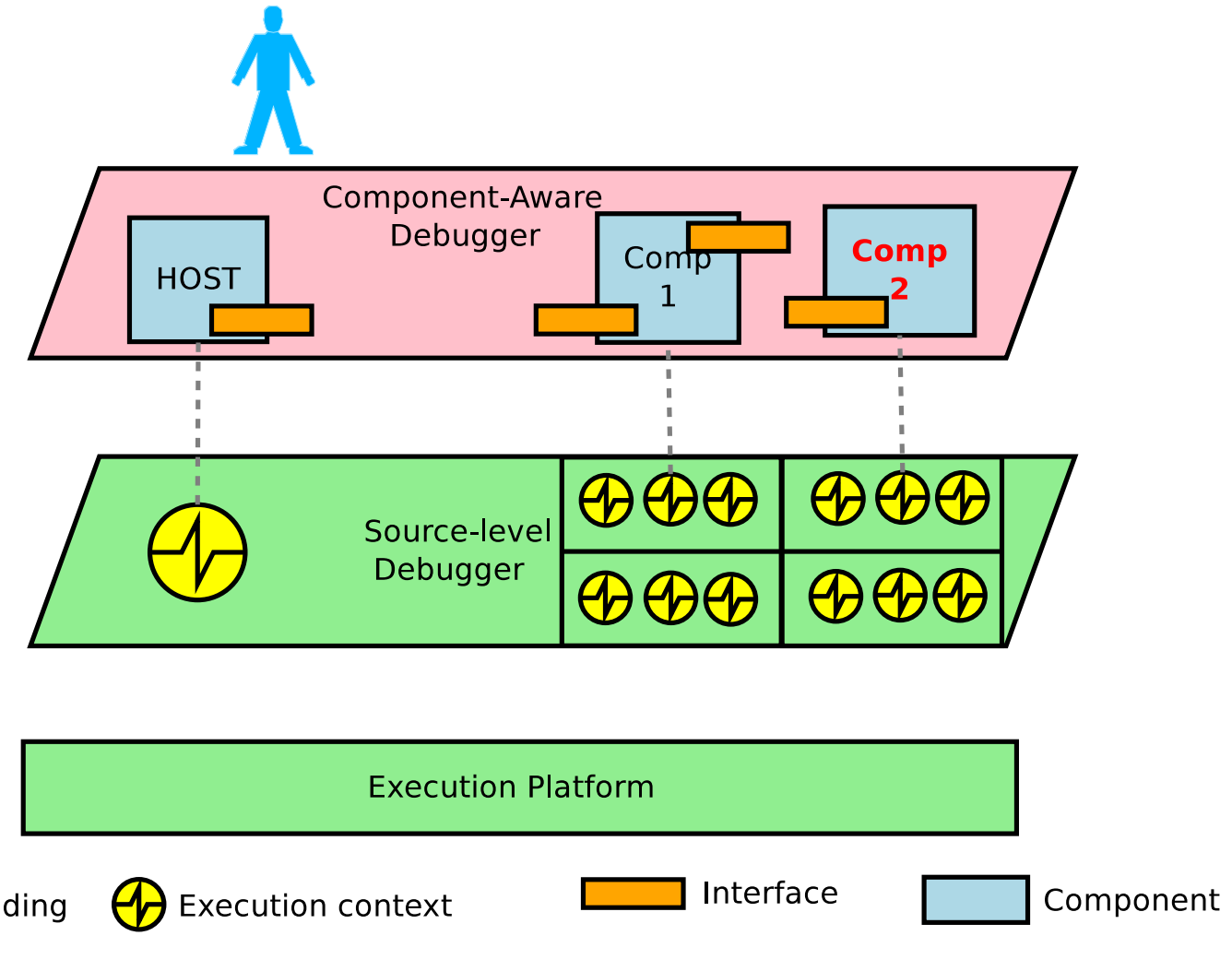
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

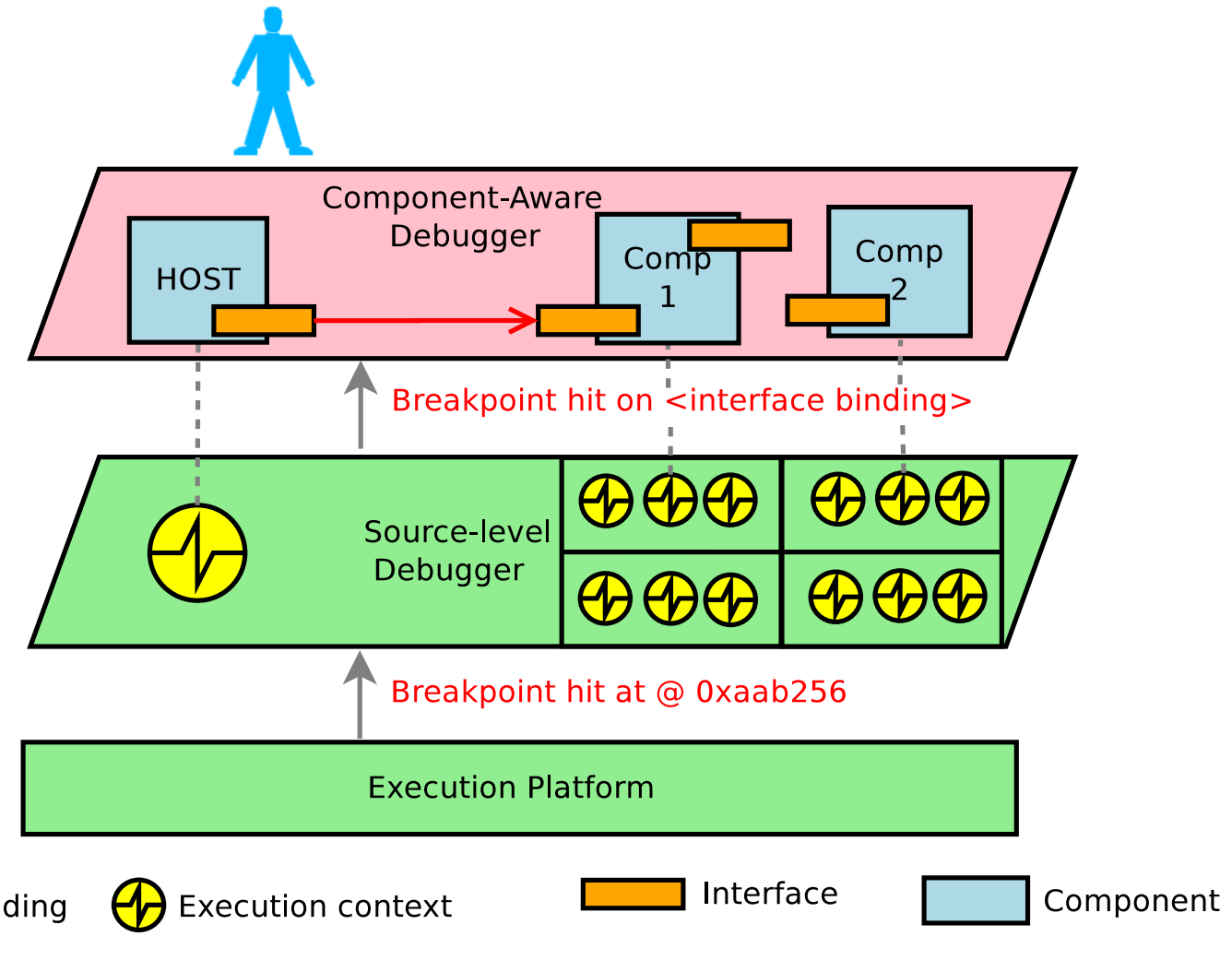
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

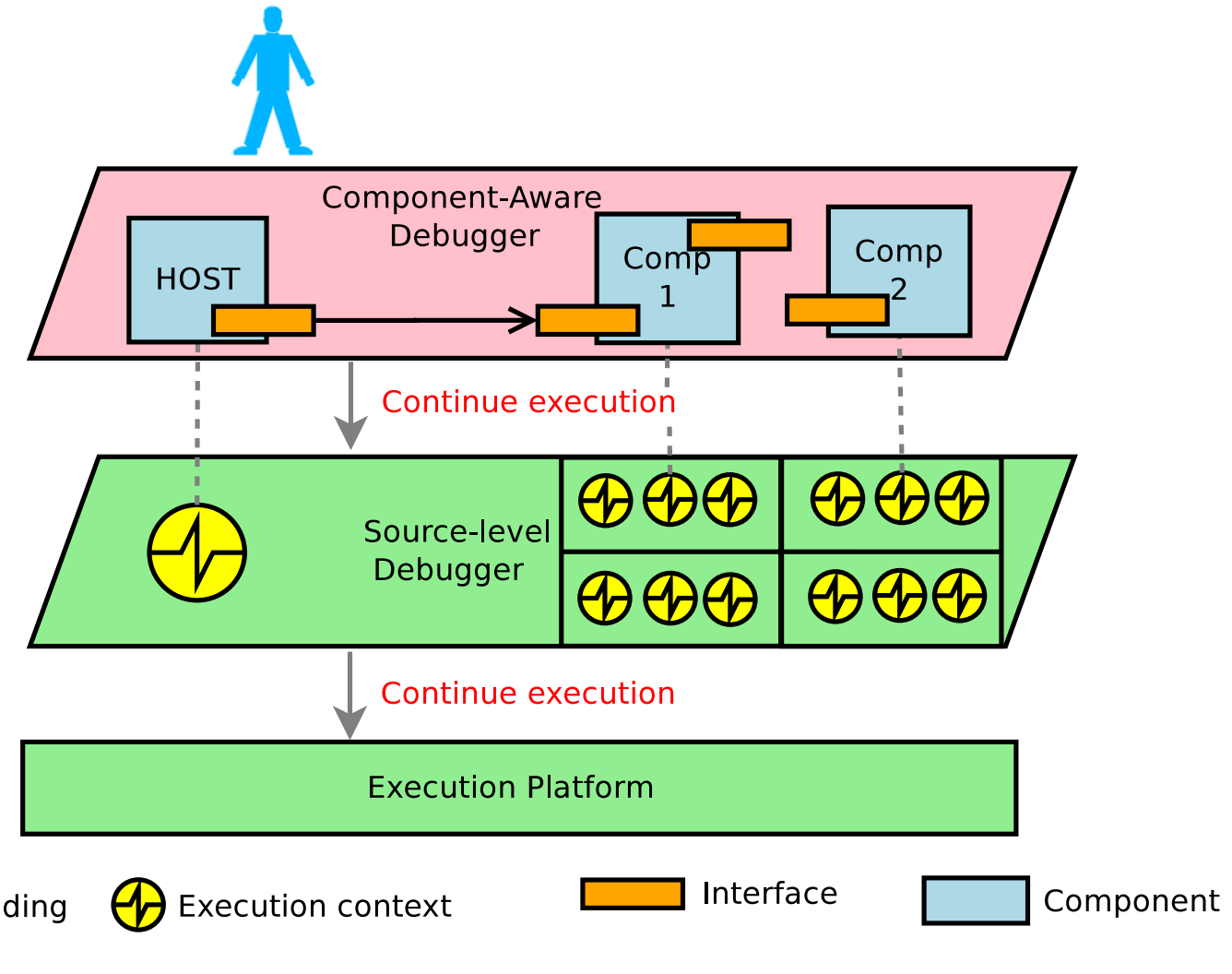
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

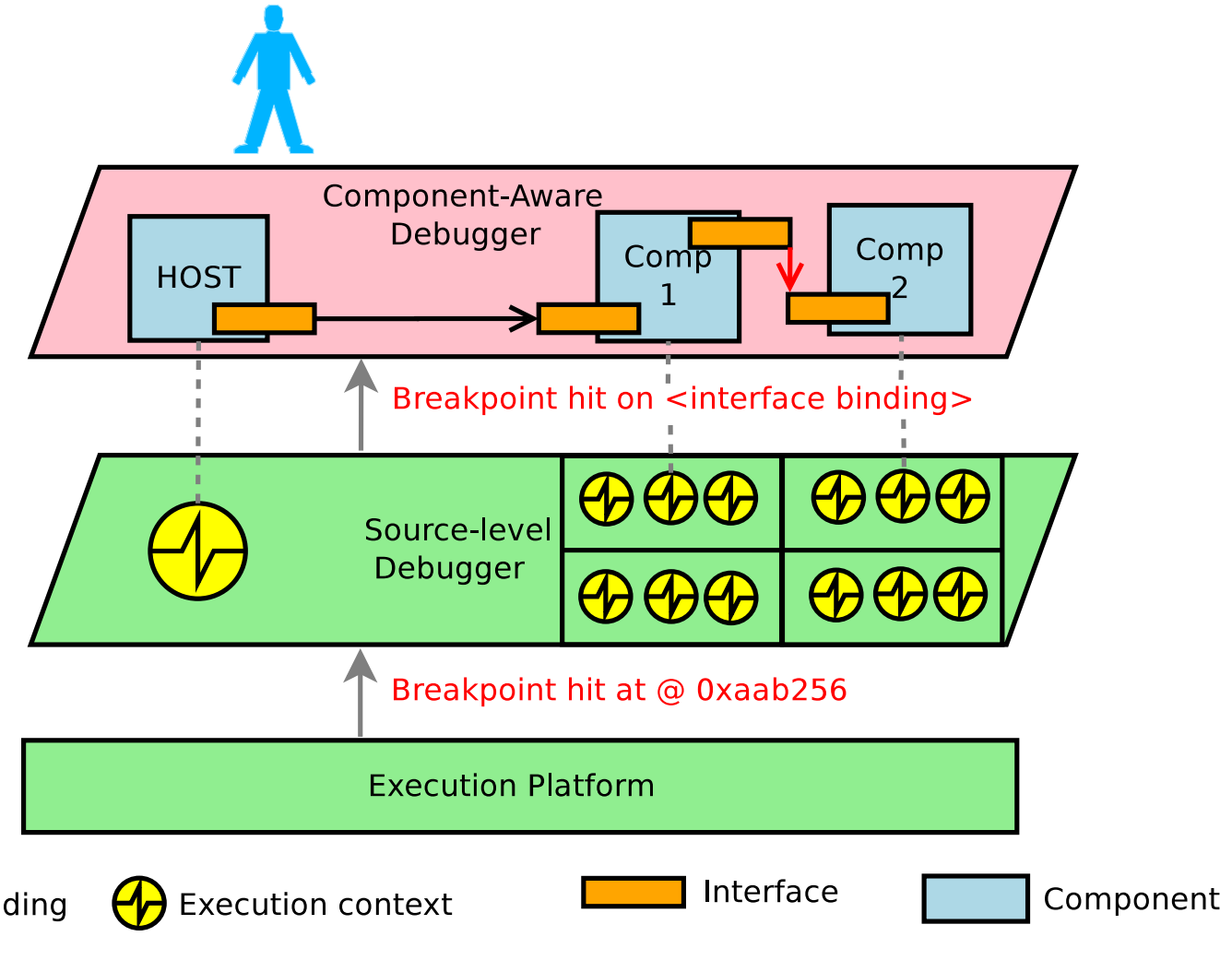
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

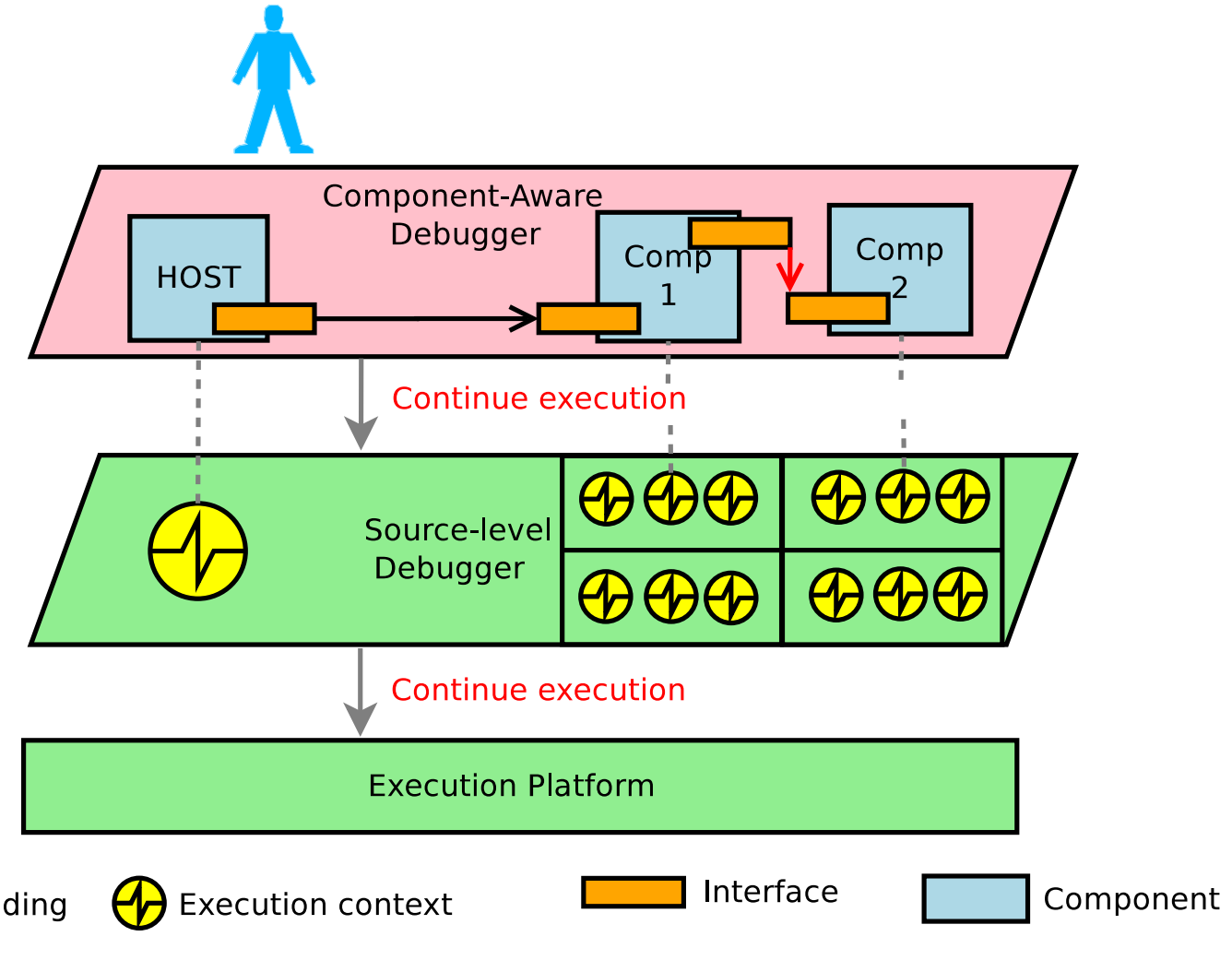
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

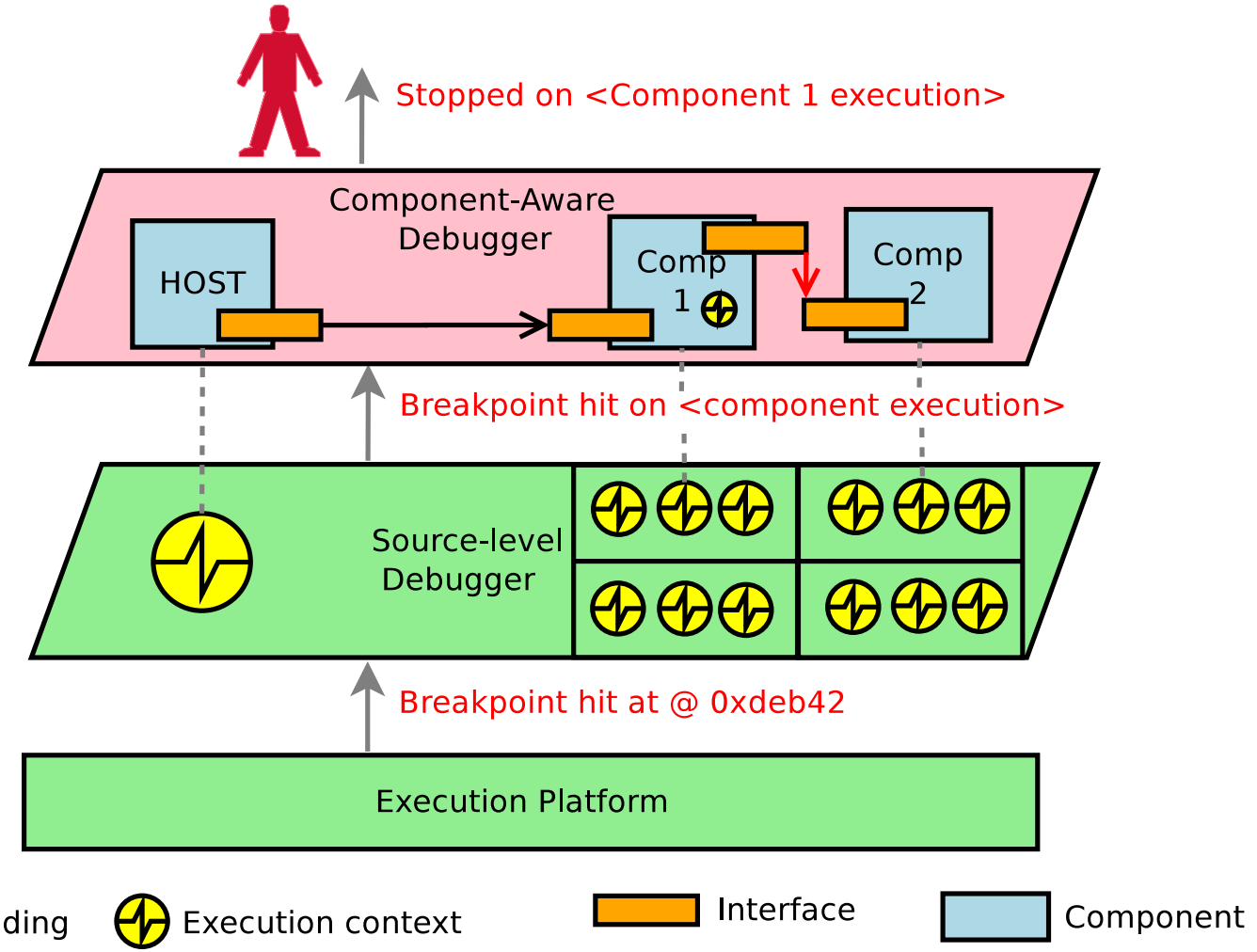
⇒ Detect and interpret key events in the component framework



Component-Aware Interactive Debugging

Implementation

⇒ Detect and interpret key events in the component framework



Agenda

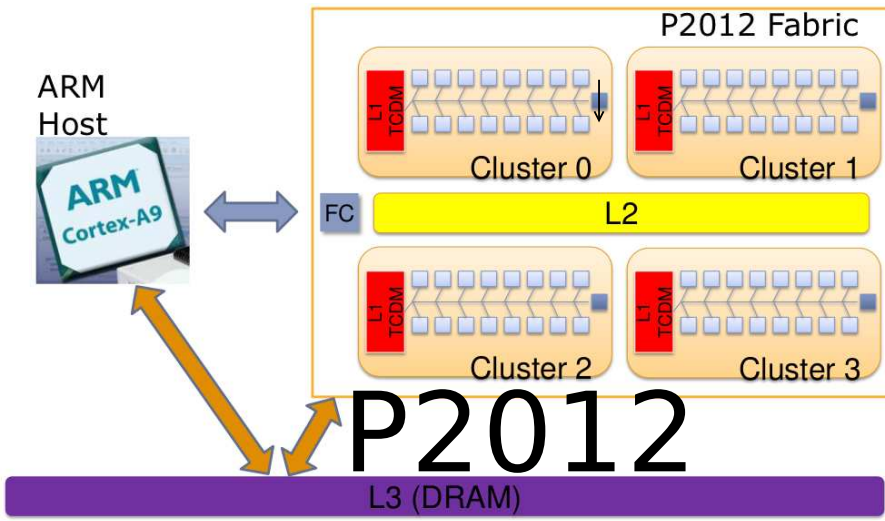
Feature Demonstration

Proof-of-concept environment

Platform 2012

ST MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Feature Demonstration

Proof-of-concept environment

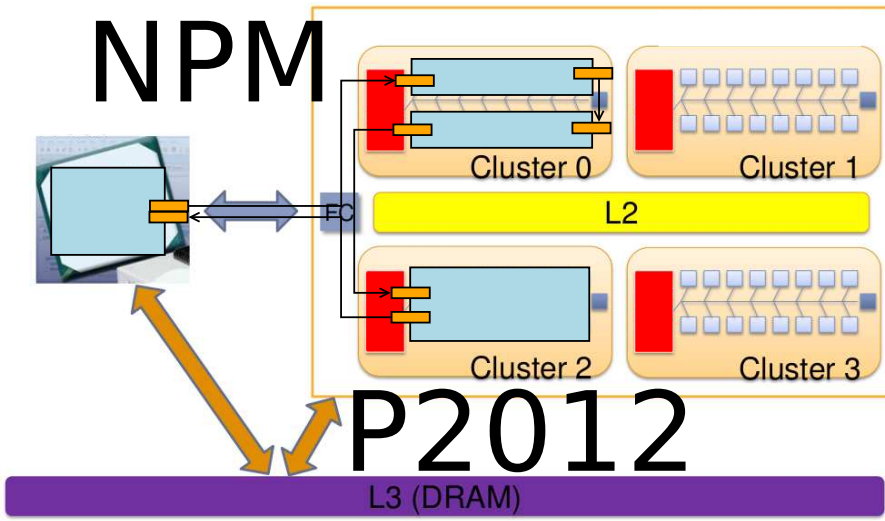
Native Programming Model

- P2012 component framework
- Provides communication components and interface

Platform 2012

ST MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Feature Demonstration

Proof-of-concept environment

The Gnu Debugger

- Adapted to low level debugging
- Large user community

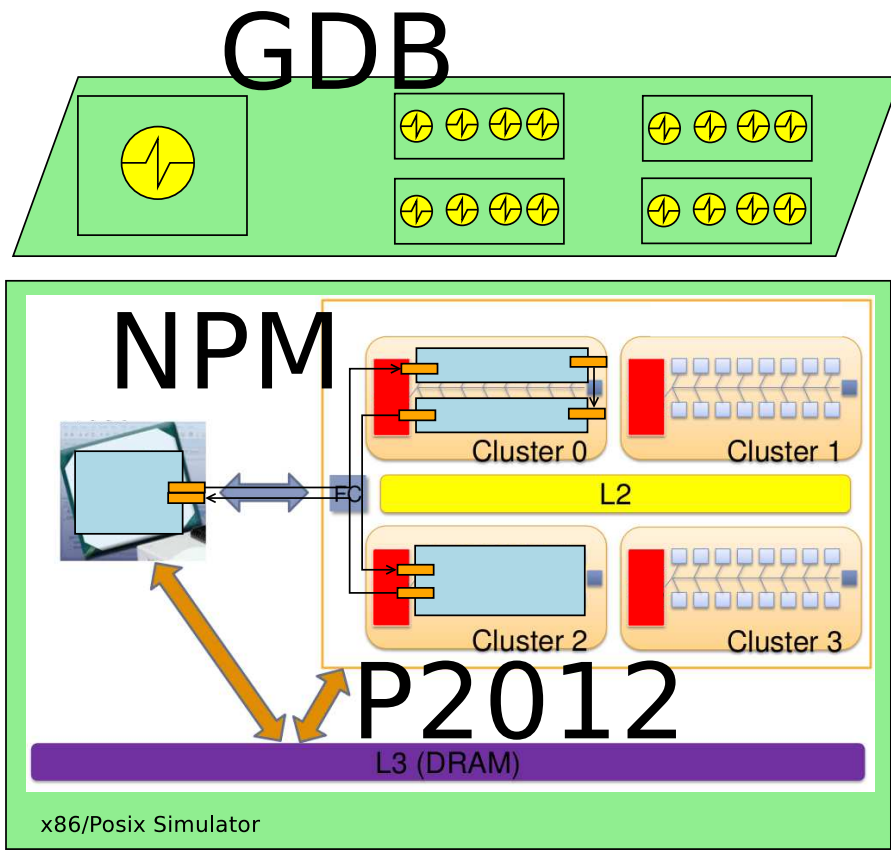
Native Programming Model

- P2012 component framework
- Provides communication components and interface

Platform 2012

ST MPSoC research platform

- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Feature Demonstration

Proof-of-concept environment

The Gnu Debugger

- Adapted to low level debugging
- Large user community

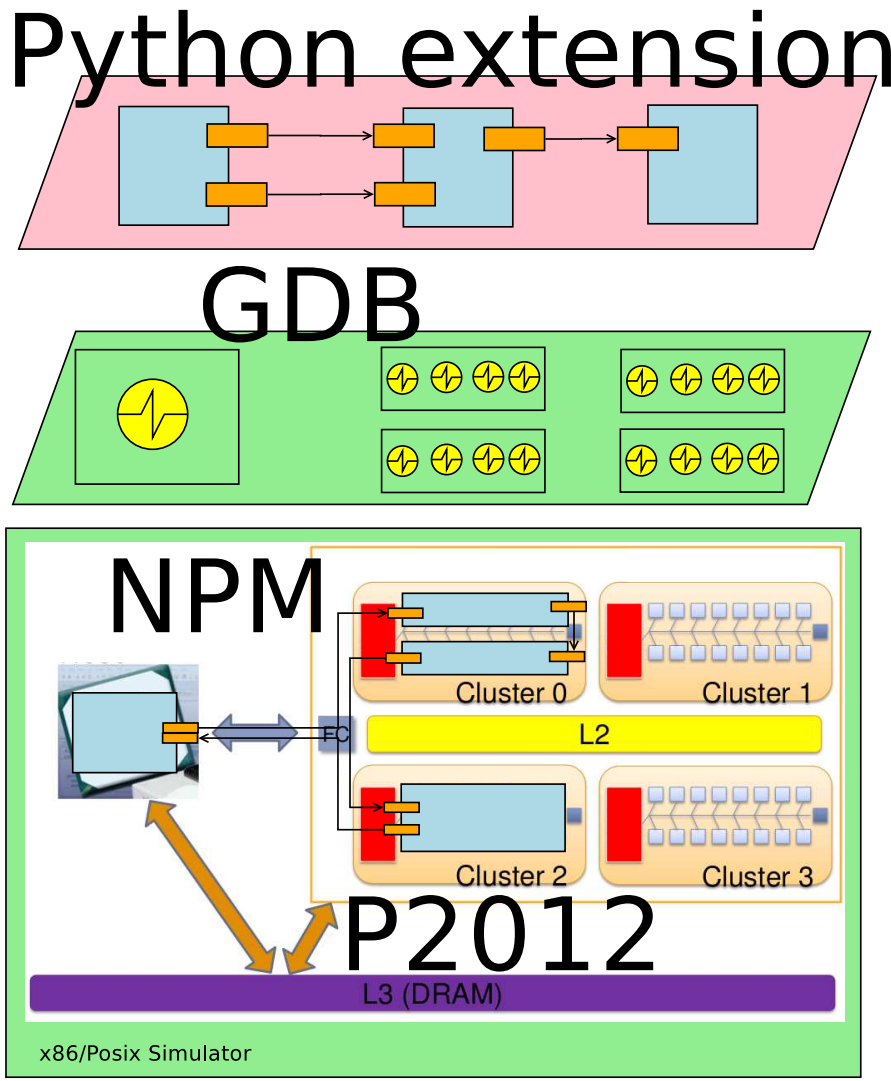
Native Programming Model

- P2012 component framework
- Provides communication components and interface

Platform 2012

ST MPSoC research platform

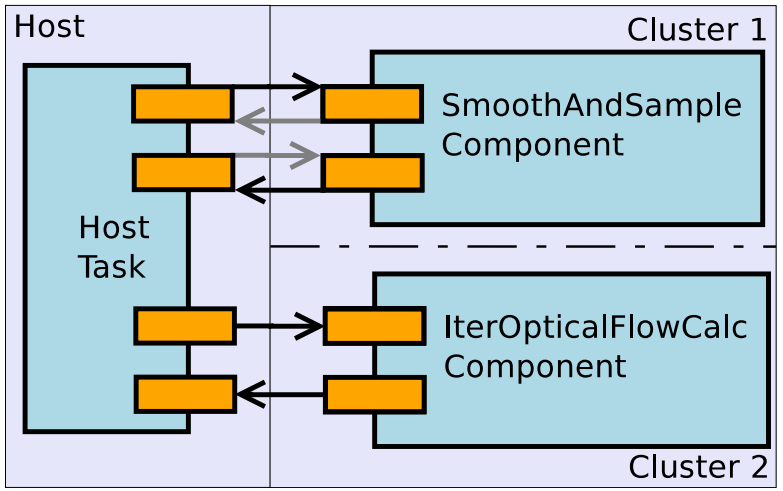
- Heterogeneous
- 4x16 CPU OS-less comp. fabric



Feature Demonstration

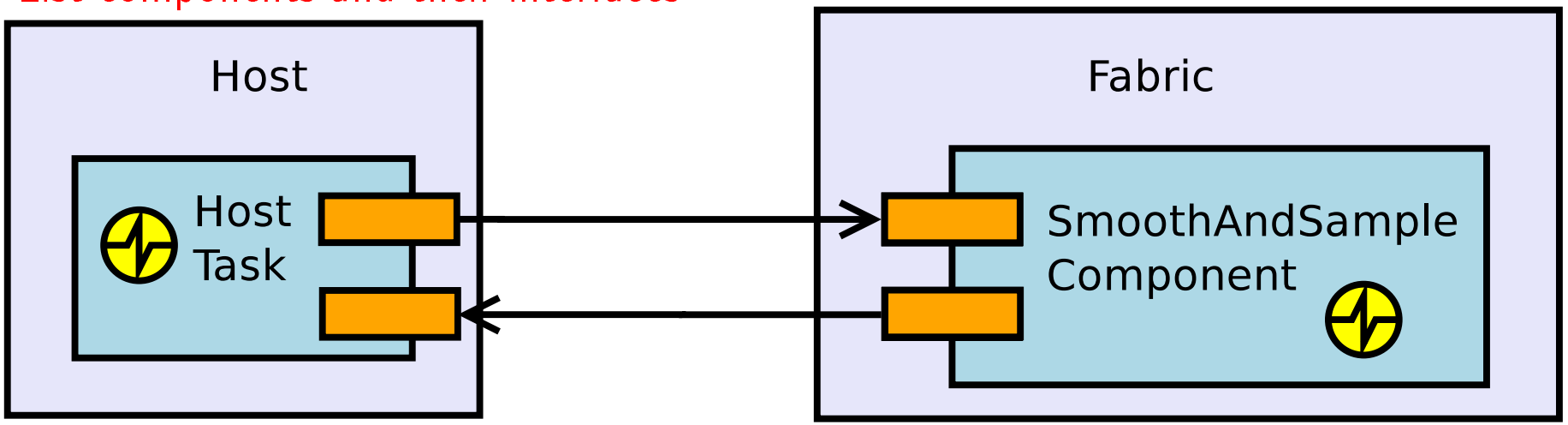
Case study: Debugging a Pyramidal Feature Tracker

- part of an augmented reality application
- analyzes video frames to track interesting features motion



Case study: Debugging a Pyramidal Feature Tracker

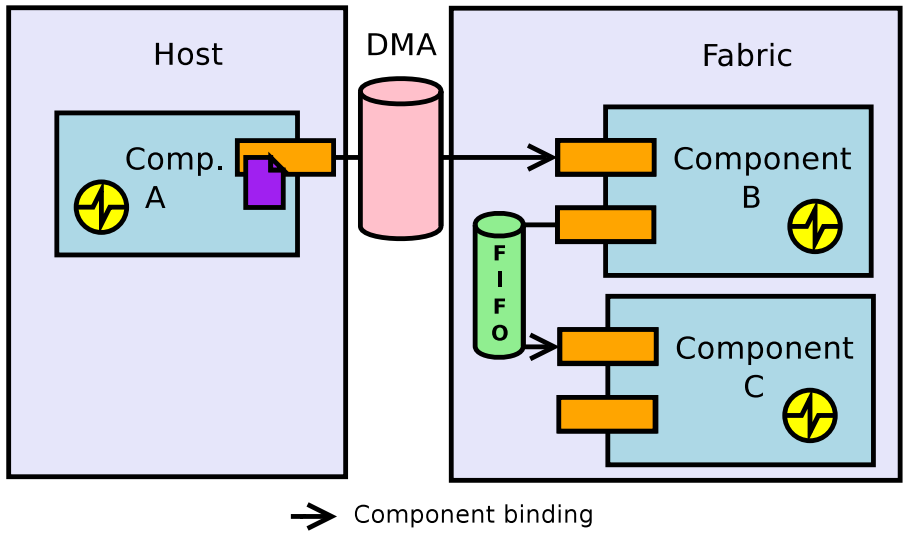
List components and their interfaces




```
(gdb) info component +connections
 #1 Host[31272]
      DMAPush/0x...   <DMA> srcPullBuffer Component...   #2
      DMAPull/0x...   <DMA> dstPushBuffer Component...   #2
      ...
* #2 Component[SmoothAndSampleProcessor.so]
      srcPullBuffer <DMA> DMAPush/0x...   Host[31272]
      dstPullBuffer <DMA> DMAPull/0x...   Host[31272]
      ...
```

Case study: Debugging a Pyramidal Feature Tracker

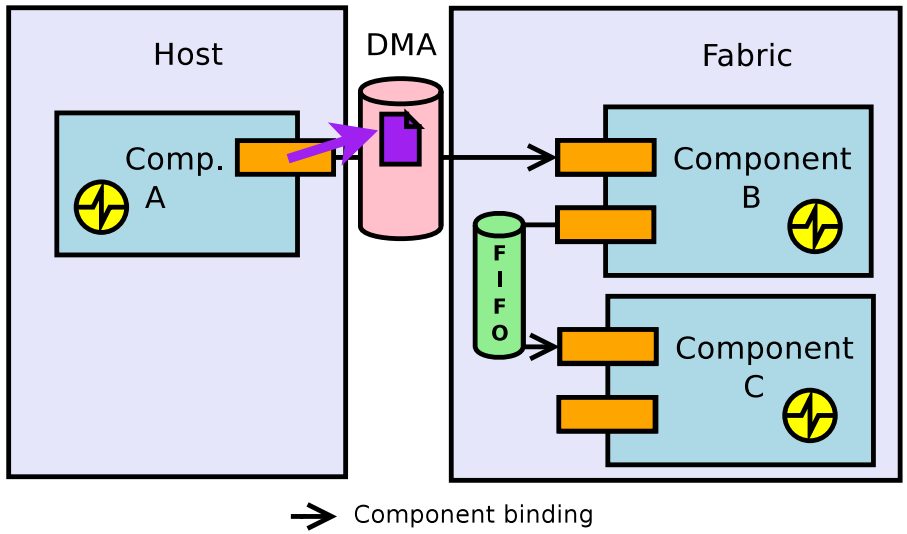
Information about messages




```
Message 1:
Component A # Message created
```

Case study: Debugging a Pyramidal Feature Tracker

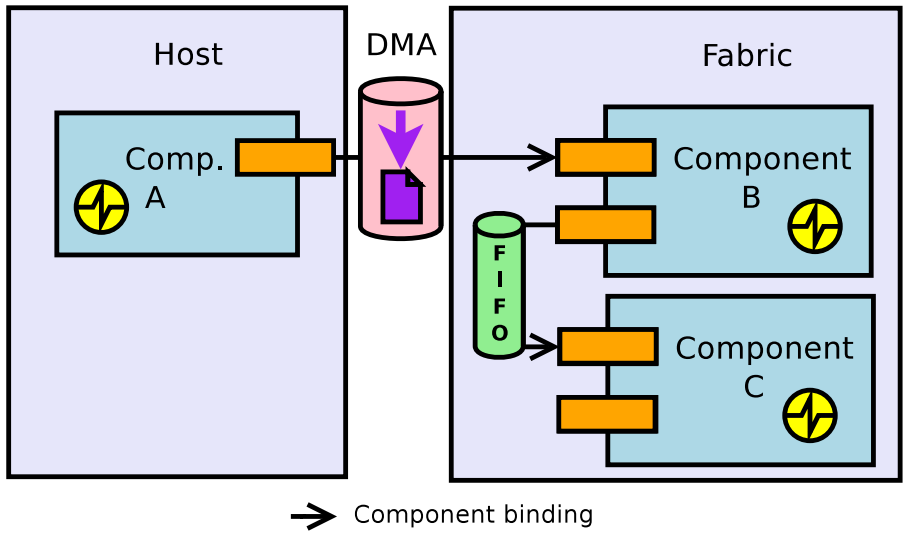
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
```

Case study: Debugging a Pyramidal Feature Tracker

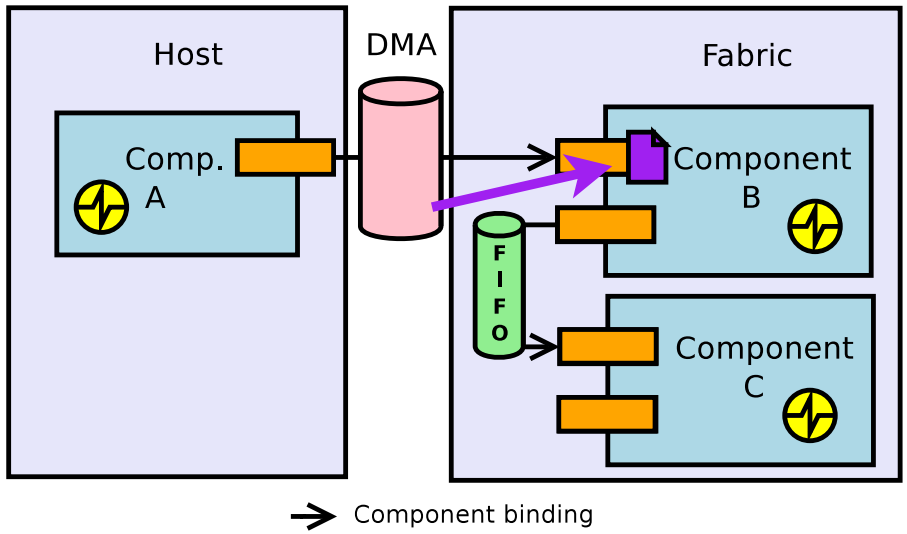
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
```

Case study: Debugging a Pyramidal Feature Tracker

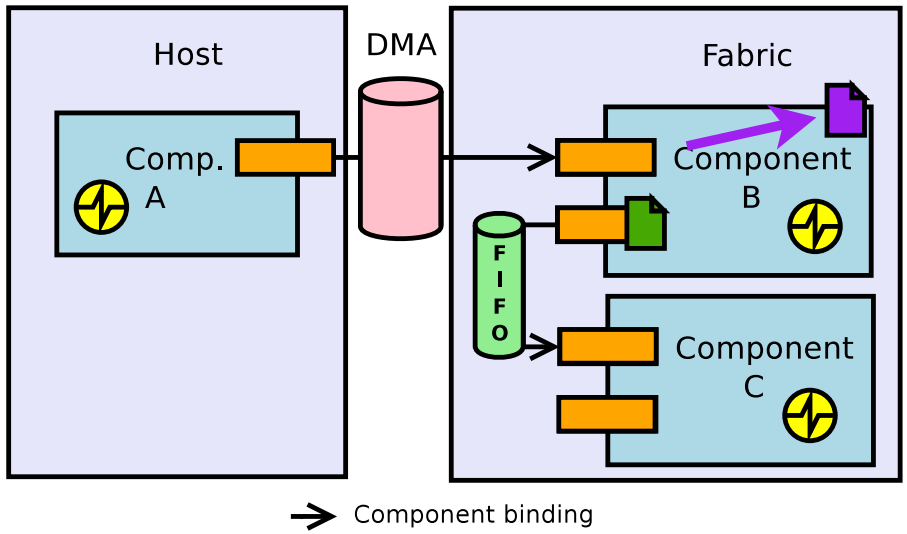
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
```

Case study: Debugging a Pyramidal Feature Tracker

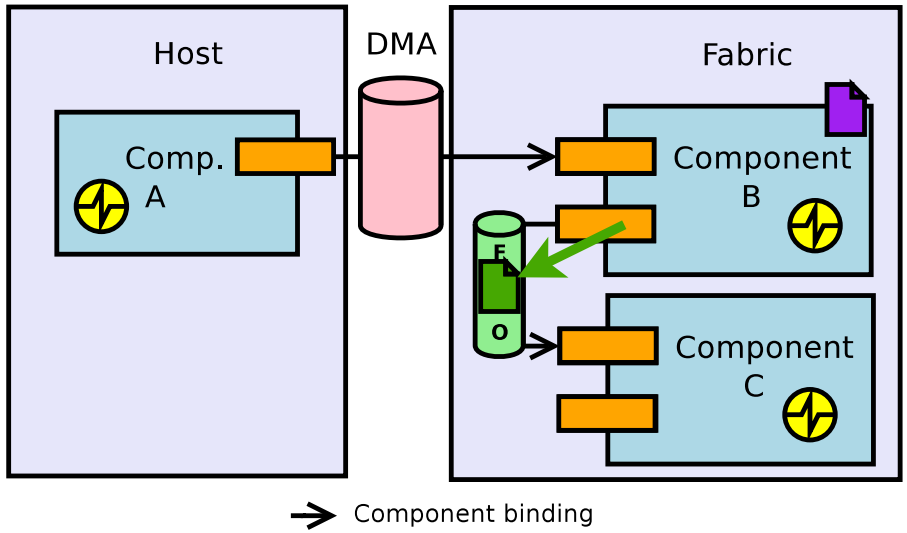
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
Message 2:
Component B # Message created
```

Case study: Debugging a Pyramidal Feature Tracker

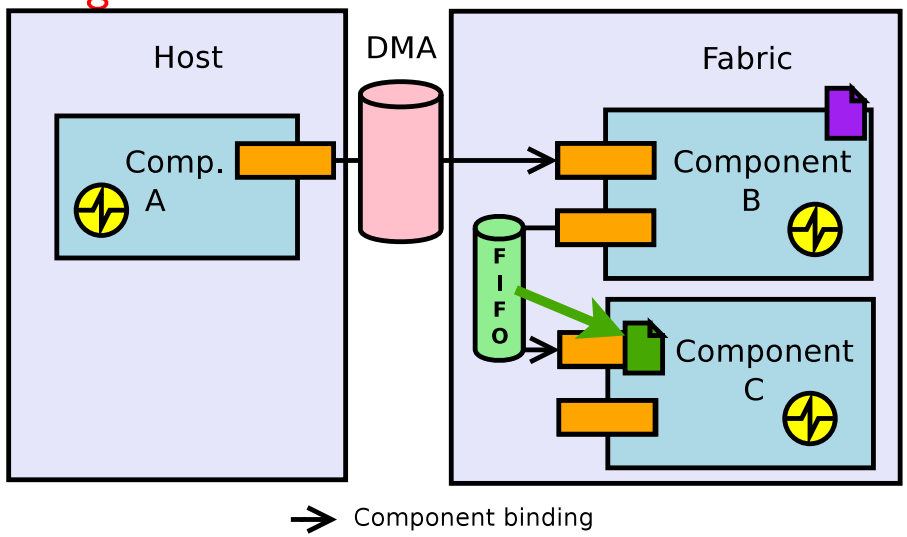
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
Message 2:
Component B # Message created
Component B::Interface B.2 # Message sent
```

Case study: Debugging a Pyramidal Feature Tracker

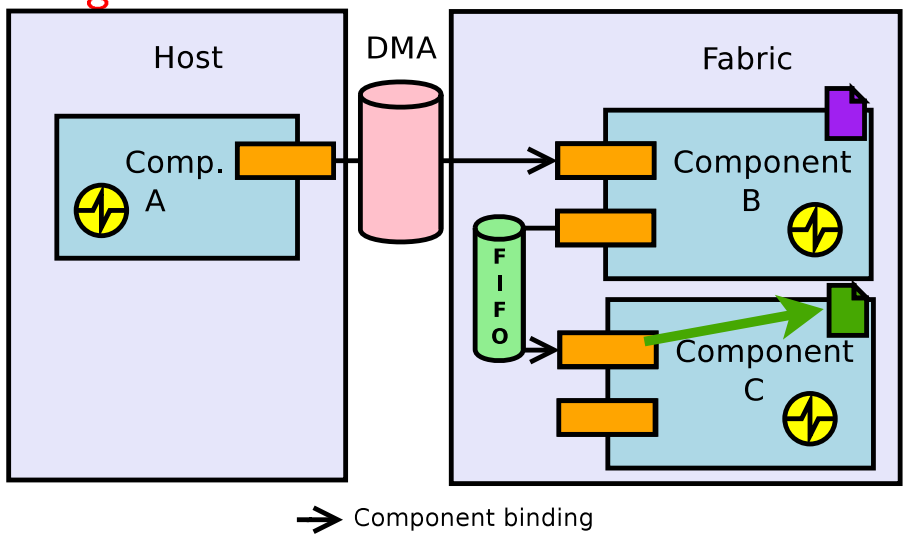
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
Message 2:
Component B # Message created
Component B::Interface B.2 # Message sent
Component C::Interface C.1 # Message received
```

Case study: Debugging a Pyramidal Feature Tracker

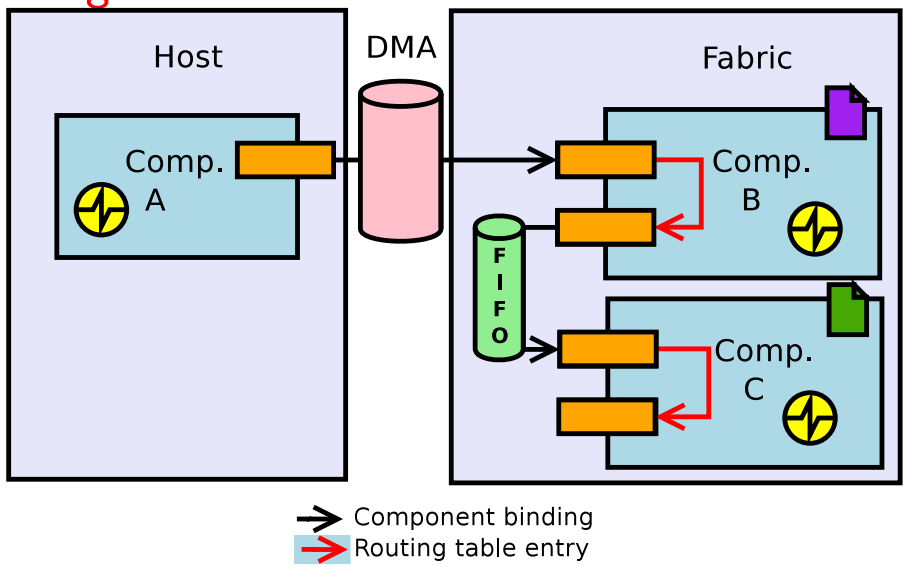
Information about messages




```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
Message 2:
Component B # Message created
Component B::Interface B.2 # Message sent
Component C::Interface C.1 # Message received
```

Case study: Debugging a Pyramidal Feature Tracker

Information about messages

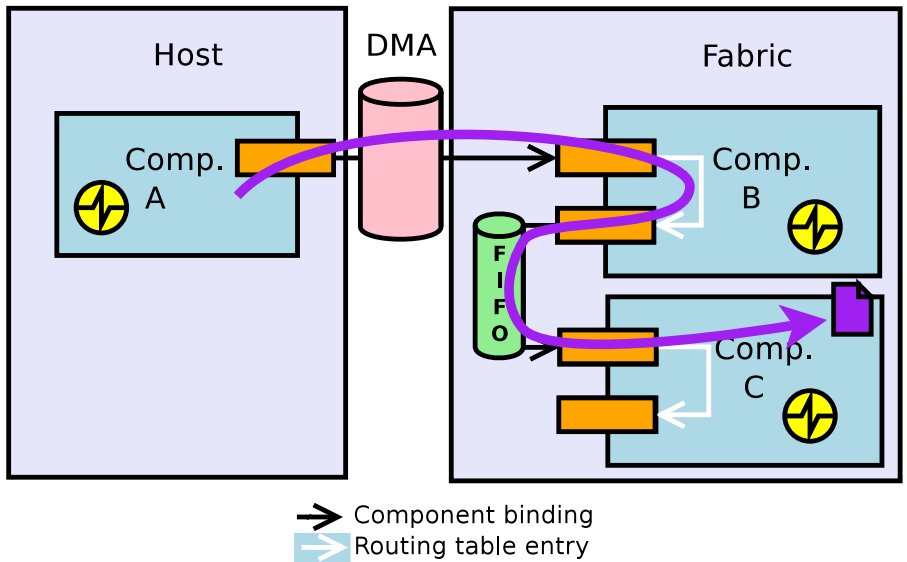


- messages can be logically aggregated with user-defined routing tables:

```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
~~Message 2:~~
~~Component B # Message created~~
Component B::Interface B.2 # Message sent
Component C::Interface C.1 # Message received
```

Case study: Debugging a Pyramidal Feature Tracker

Information about messages

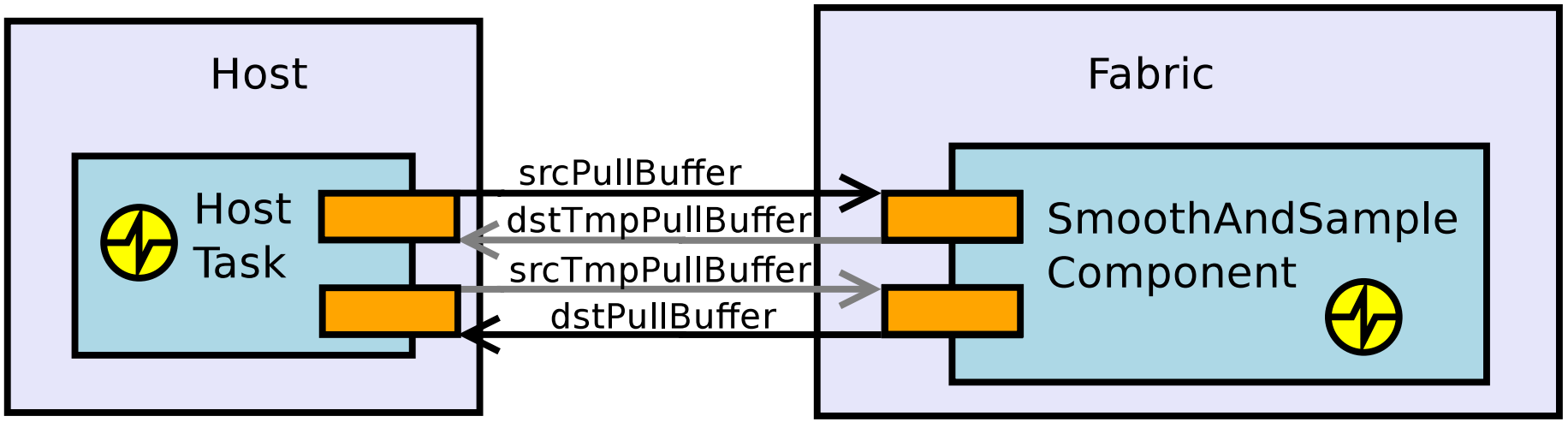


- messages can be logically aggregated with user-defined routing tables:

```
Message 1:
Component A # Message created
Component A::Interface A.1 # Message sent
Component B::Interface B.1 # Message received
Component B::Interface B.2 # Message sent
Component C::Interface C.1 # Message received
```

Case study: Debugging a Pyramidal Feature Tracker

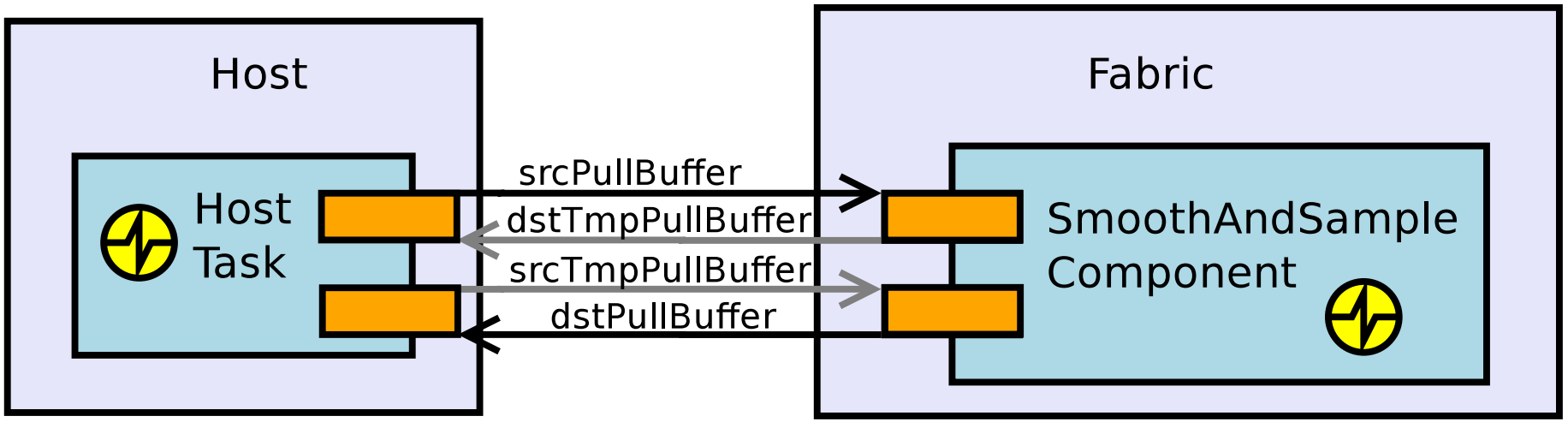
Information about interface activity




```
(gdb) info components +counts
#2 CommComponent[SmoothAndSampleProcessor.so]
    srcPullBuffer #35 msgs
    dstTmpPushBuffer #36 msgs
    srcTmpPullBuffer #35 msgs
    dstPushBuffer #34 msgs
```

Case study: Debugging a Pyramidal Feature Tracker

Information about interface activity




```
(gdb) info components +counts
#2 CommComponent[SmoothAndSampleProcessor.so]
    srcPullBuffer #35 msgs
    dstTmpPushBuffer #36 msgs
    srcTmpPullBuffer #35 msgs
    dstPushBuffer #34 msgs
```

- allowed us to find a bug in the application (msg sent to the wrong interface)

Case study: Debugging a Pyramidal Feature Tracker

Information about interface activity

Excerpt from a 300 lines-of-code file

```
/* Compute last lines if necessary */
if (tmp_size > 0) {
    ...
    /* Transmit the last lines computed */
    CALL(srcTmpPullBuffer, release)(...);
    CALL(dstTmpPushBuffer, push)(...);
}
```

Agenda

Conclusion

- Debugging dynamic component application is challenging
- No high level information about components

- **Our contribution:** bring debuggers closer to the component model
  - better understanding application behavior
  - keep focused on bug tracking

Conclusion

- Debugging dynamic component application is challenging
- No high level information about components

- **Our contribution:** bring debuggers closer to the component model
  - better understanding application behavior
  - keep focused on bug tracking

- Proof-of-concept: GDB/Python, widely used in embedded development

- Going further programming-model aware debugging
  - OpenCL,
  - Dataflow execution model,
  - ...