Automatic Exploration of SW Concurrency Bugs through Deterministic Behavior Control

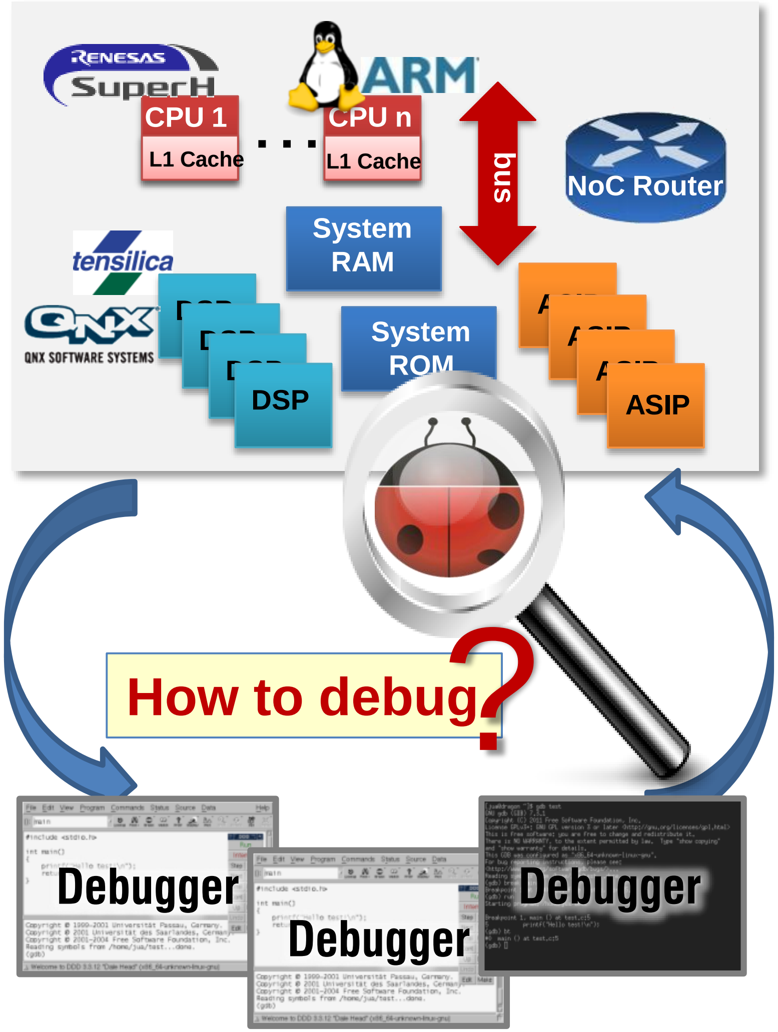
Luis Gabriel Murillo, Rainer Leupers

MAD Workshop

14.11.13, Munich, Germany

Motivation: MPSoC Debug Challenges

- **MPSoCs**
  - Complex communication
  - Shared memory, *KPN* and *SDF* models, message passing…
  - Co-existing OSs, middle-wares...
- **Concurrency → Non-determinism**
- **Many-cores → Many debuggers?**



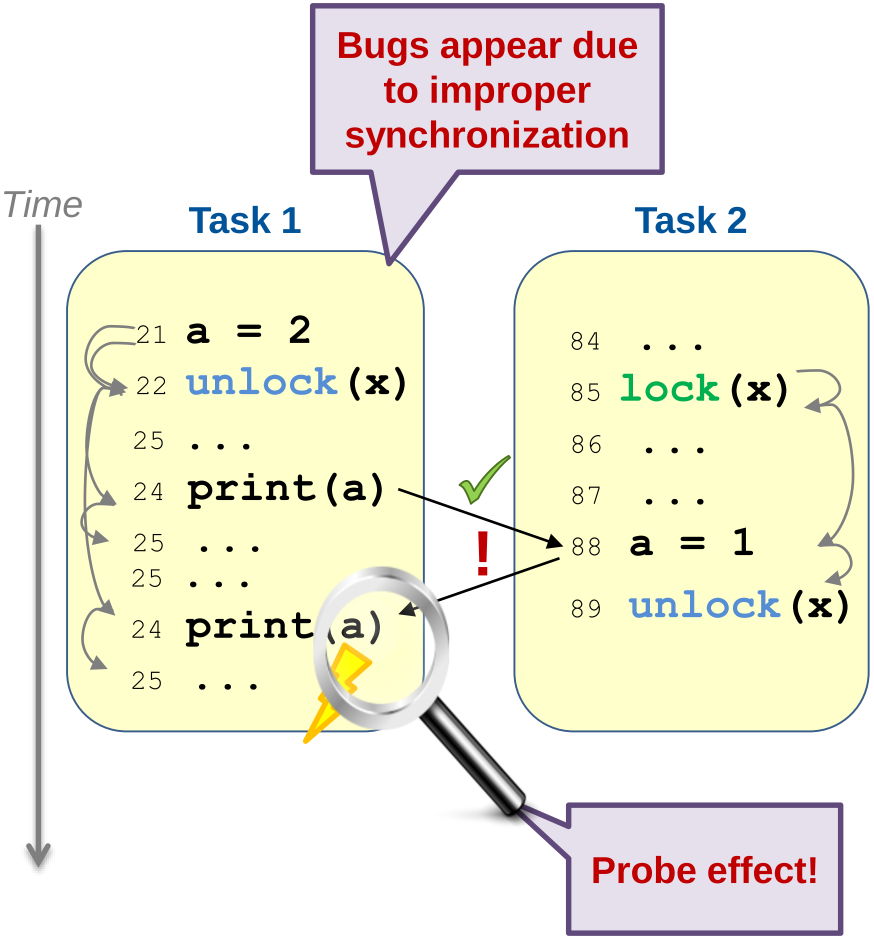
Motivation: Concurrency Bugs

- **MPSoCs are non-deterministic**

- **Concurrency Bugs**
  - Races (order and atomicity violations)
  - Deadlocks, livelocks…

- **Difficult to:**
  - Find
  - Understand
  - Reproduce

→Remain unnoticed



Agenda

MPSoC Debug Challenges

→ Methodology Overview

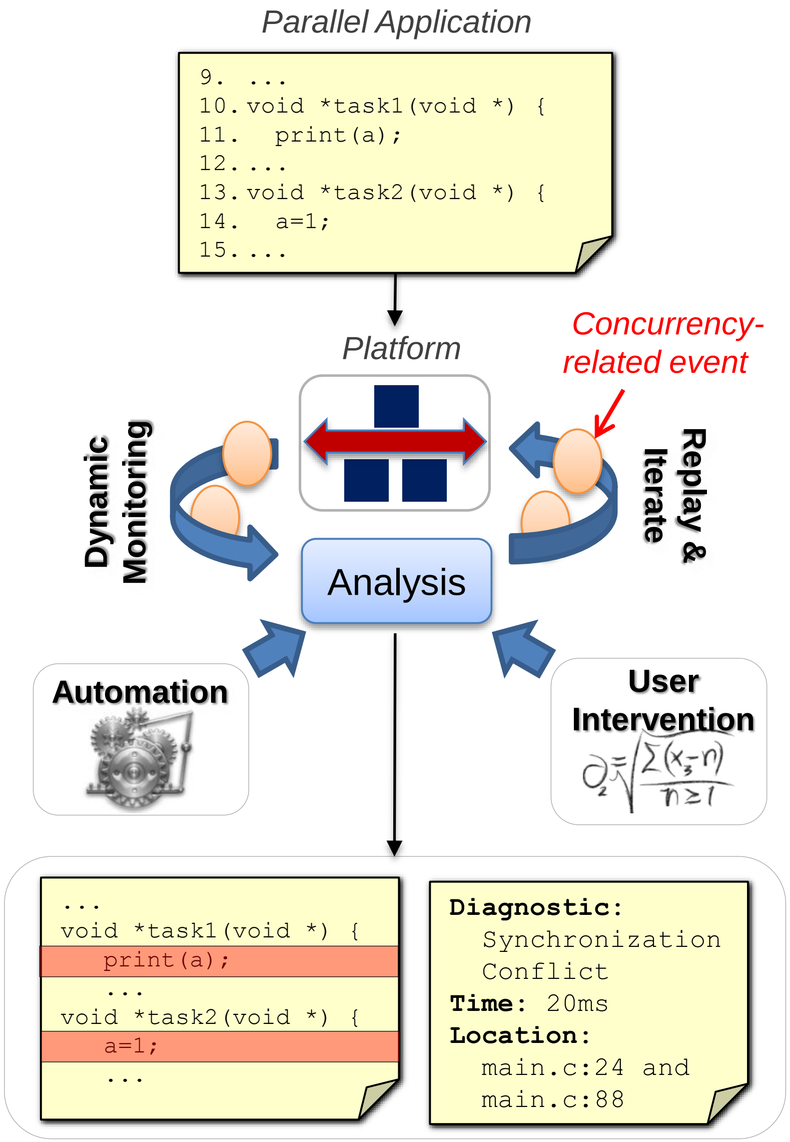
Event-based Debugging

Determinism Analysis & Behavior Control

Results and Conclusions

MPSoC Debug Toolflow

- **Goals:**
  - Help in finding concurrency bugs
  - Unique methodology / debugger for different platforms
  - Tool for SW programmer
- **Key aspects:**
  - Abstraction
  - Automation
  - Retargetability
  - Scalability



```
...
void *task1(void *) {
  print(a);
...
void *task2(void *) {
  a=1;
...
```

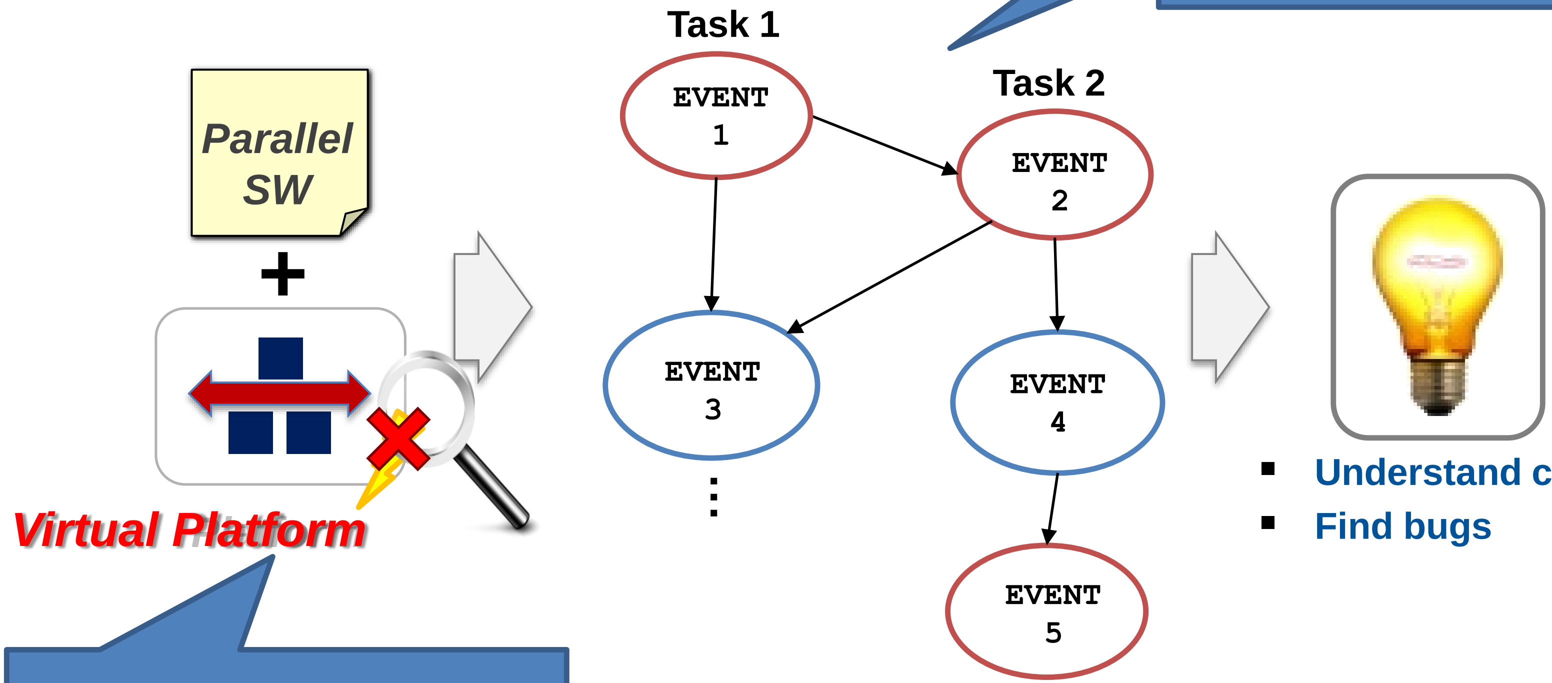
```
...
void *task1(void *) {
   print(a);
   ...
void *task2(void *) {
   a=1;
   ...
```

```
Diagnostic:
  Synchronization
  Conflict
Time: 20ms
Location:
  main.c:24 and
  main.c:88
```

Event-based Debugging

- **Abstracting away program flow:**
  - Focus on programmer level actions / concurrency related events

All synchronization, task management, message passing, shared memory…



- **Understand concurrency**
- **Find bugs**

- Non-intrusive inspection
- System-wide view
- Unmodified SW execution

Related Work

	AVIO **(Lu et al. ’06)**	**Chess** **(Microsoft ’08)**	**Portend** **(EPFL ’12)**	**This work**
Target system	x86	Windows	LLVM	Virtual Platform
Target application	C(++)	.NET	Pthread	SW + HW
Non-intrusive	✗ Instrumentation	✗ Wrapper	✗ Symbolic execution	✓
Deterministic replay	✗	✓	✓	✓
Deterministic program exploration	✗	✓	✓	✓
Extensibility	✗	✗	✗	✓

Agenda

MPSoC Debug Challenges

Methodology Overview

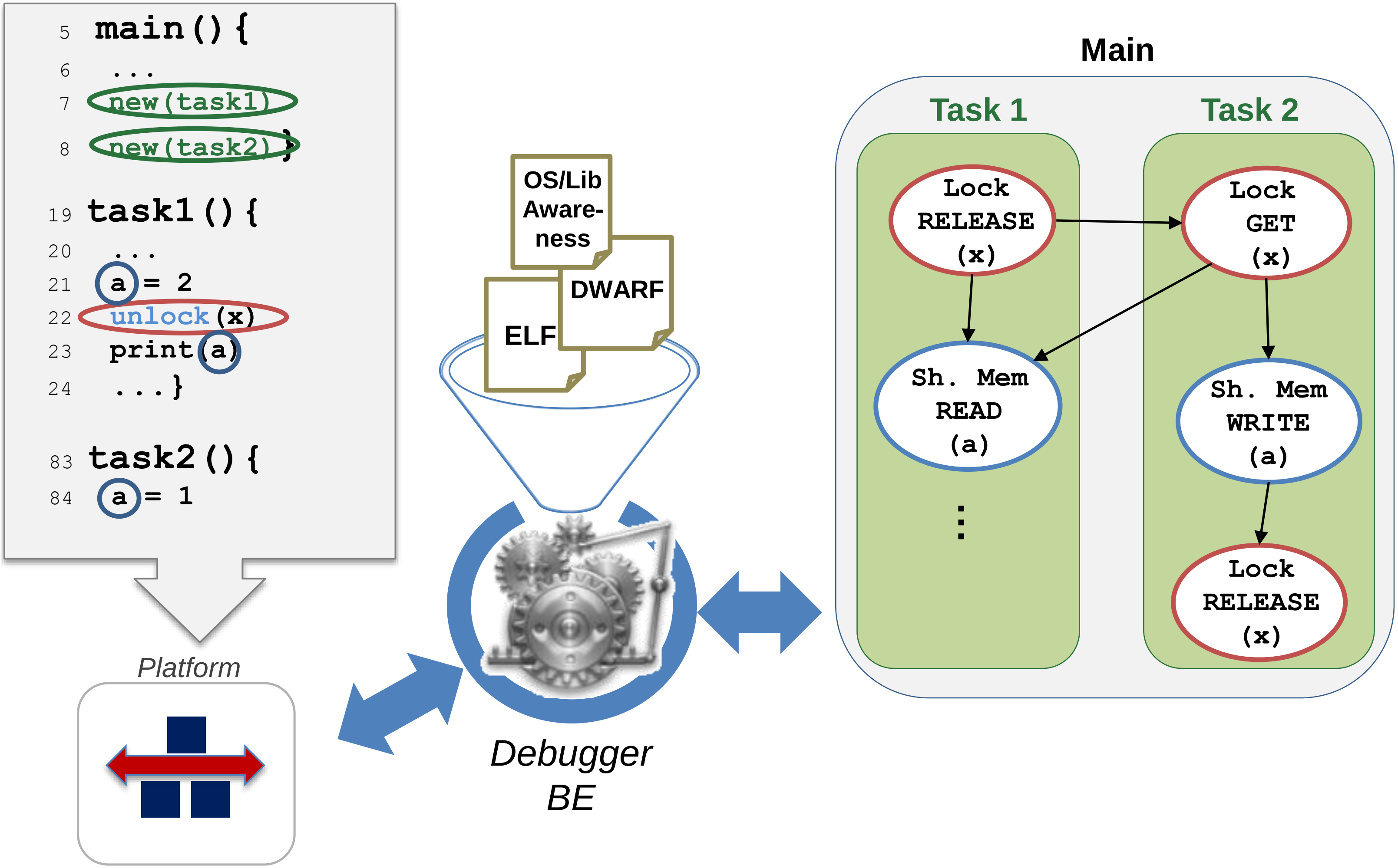
➔ Event-based Debugging

Determinism Analysis & Behavior Control

Results and Conclusions

Abstracting Concurrent Software

- **Debugger framework for Dynamic Monitoring**



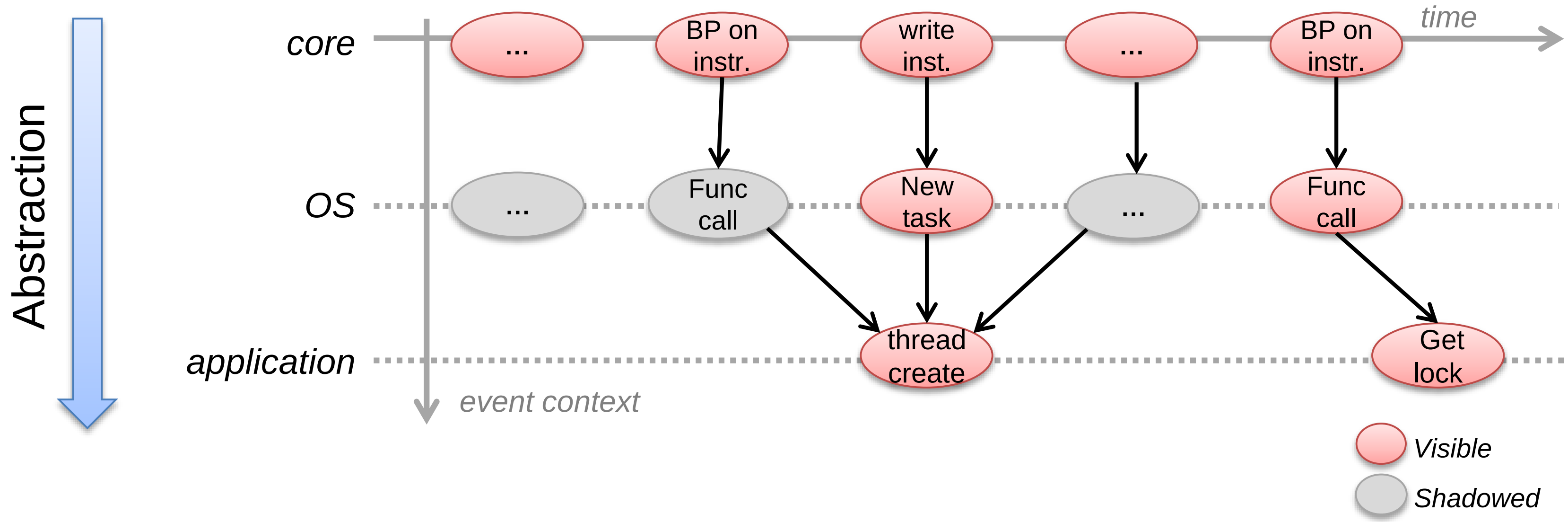
```
main(){
 ...
 new(task1)
 new(task2)}

task1(){
 ...
 a = 2
 unlock(x)
 print(a)
 ...}

task2(){
 a = 1
```

Event Composition

- **Problem: High-level atomic events for analysis but fully trackable to origins**
- **Solution:**
  - Bi-dimensional composition: *time*, *context*
  - Propagation of semantic information



Event-based Debugging: Advantages

- **Reveals the order of programming-level events**
  - “Understanding” the application

- **Identification of relevant source code location / task / core**
  - Dynamic monitoring with source debugger

- **No source code instrumentation, no changes to target SW, non-intrusive monitoring…**

- **Trace captures one single execution**
  - One single “task interleaving” ☺
  - Other possible interleavings? ☹

Agenda

MPSoC Debug Challenges

Event-based Debugging

Bug-pattern Assertions

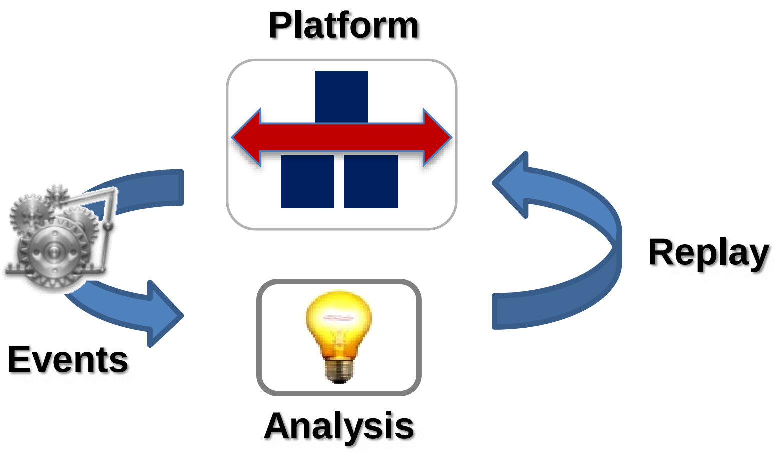
➔ Determinism Analysis & Behavior Control

Results and Conclusions

Determinism Analysis

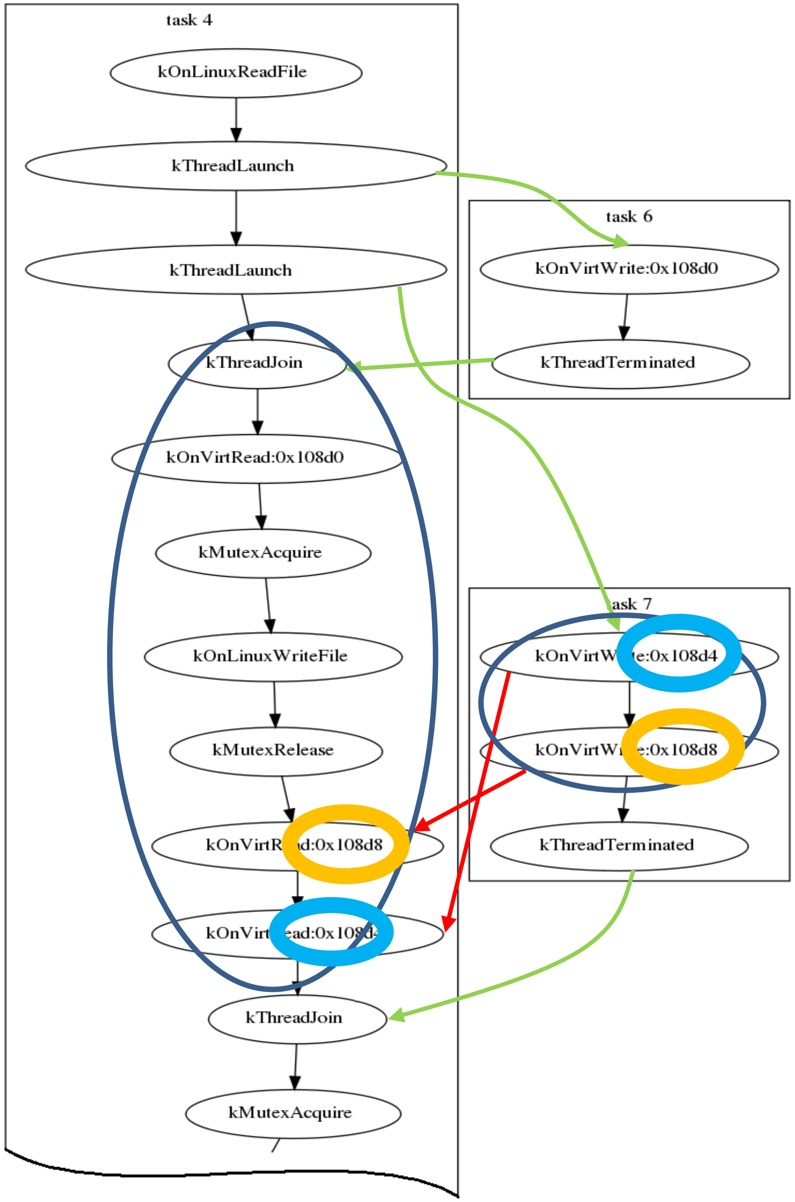
- **Problem: “One single execution is not enough to spot concurrency bugs“**

- **Solution: concurrency analysis and controlled replay**
  - Investigate suspicious interleavings
  - Identification of non-determinism ‘with notable effect‘
  - Provoke bugs which are hidden! ☺



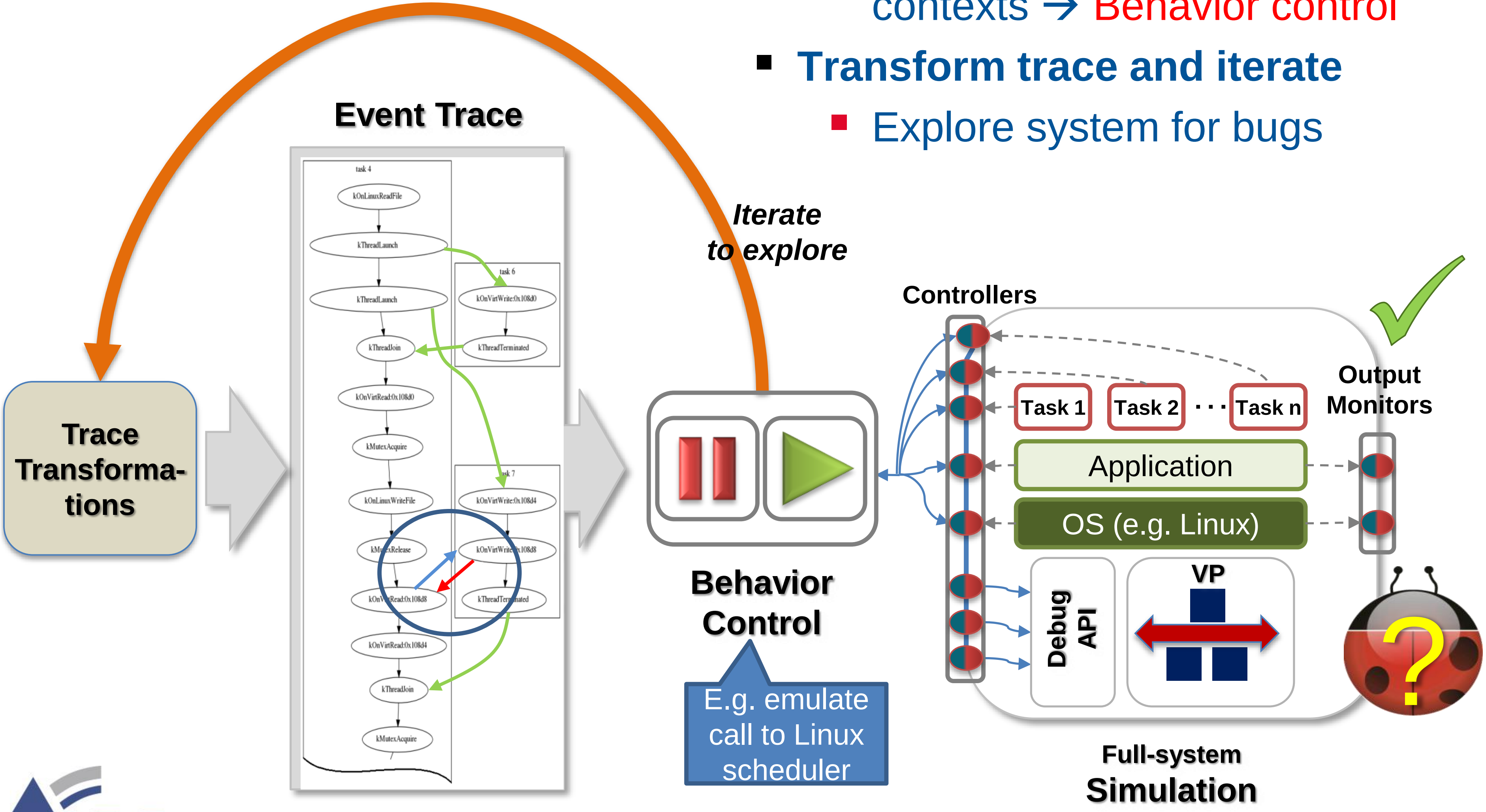
Analyzing the Event Trace

- **Concurrency analysis and conflict extraction:**
  1. Identify synchronization
     - Mark “always happen” event orders (“happens before” analysis)
  2. Identify “always concurrent” events
  3. Identify event dependencies
     - On shared resources (“Visit/Modify”)
  4. Identify conflicts
     - Dependencies not in sync
  5. For exact replay or bug provoke:
     - Enforce order of conflicting events
     - Minimal set of event pairs



Replay and Trace Transformations

- **Event-based replay**
  - Suspend/resume event contexts → Behavior control
- **Transform trace and iterate**
  - Explore system for bugs



Event Trace

Trace Transforma-tions

Iterate to explore

Behavior Control

E.g. emulate call to Linux scheduler

Controllers

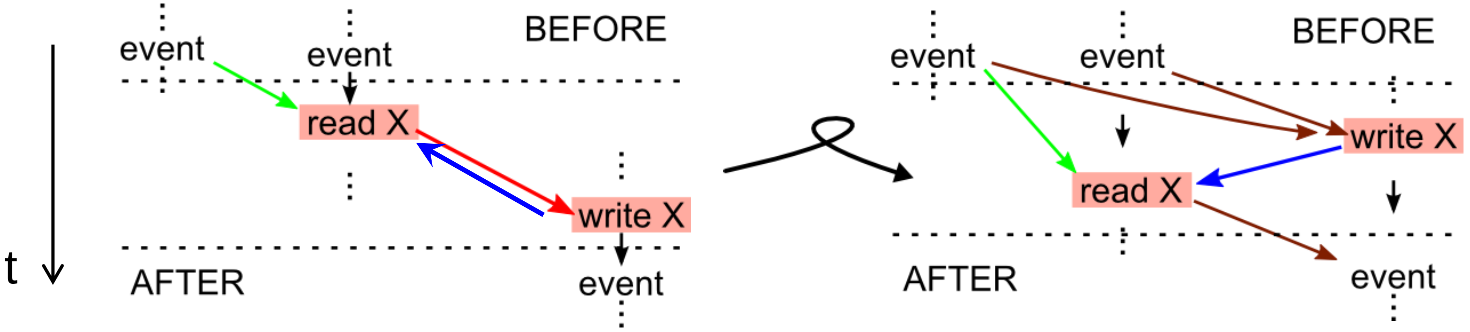
Output Monitors

Full-system Simulation

Constraint Swapping

- **Swapping a conflicting event order**
  - Locally invert a constraint
  - → Single swap is safe and likely to change behaviour

- **Swapping a constraint**
  1. **Swap** event pair order
  2. Add **repair constraints** for locality



→Random Constraint Swapping

Agenda

MPSoC Debug Challenges

Event-based Debugging

Bug-pattern Assertions

Determinism Analysis

➔ Results and Conclusions

Target Systems and Results

- **EURETILE (www.euretile.eu)**
  - European reference tiled architecture experiment
  - Many-tiled system for embedded and HPC
- **Multi-core Synopsys Virtual Platforms**
  - ARM Versatile Express with 4 Cortex A9
    - SMP Linux 3.4.7, pthreads, SPLASH-2

Results ARM Versatile Express		
Event-based Framework		
	Retargetable BE	**High-level Monitors**
Adaptation Effort	~1 man-month	~2 man-days
Monitoring and Analysis		
	Synthetic	**SPLASH-2**
Total events (no SM)	~500	600 – 123k
Total events	~2500	3000 – 1.9M
Overhead	~3x	~3x (WC:60x)
Replay Constraints	~50	500 - 3200

E.g., Analysis of SPLASH2 OCEAN Application

→ Event trace and analysis results

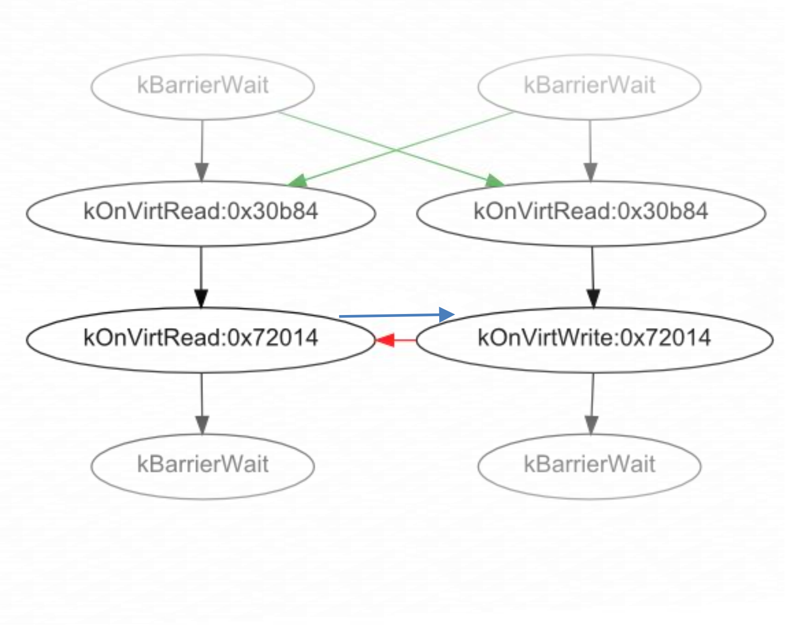
	Filtered conflicts			
	Total	Sync 🟩	Mutex 🟦	Conflict 🟥
Count	284	260	23	1
rel.		91.5 %	8.1 %	0.4 %

→ Unsynchronized dependency in OCEAN event trace

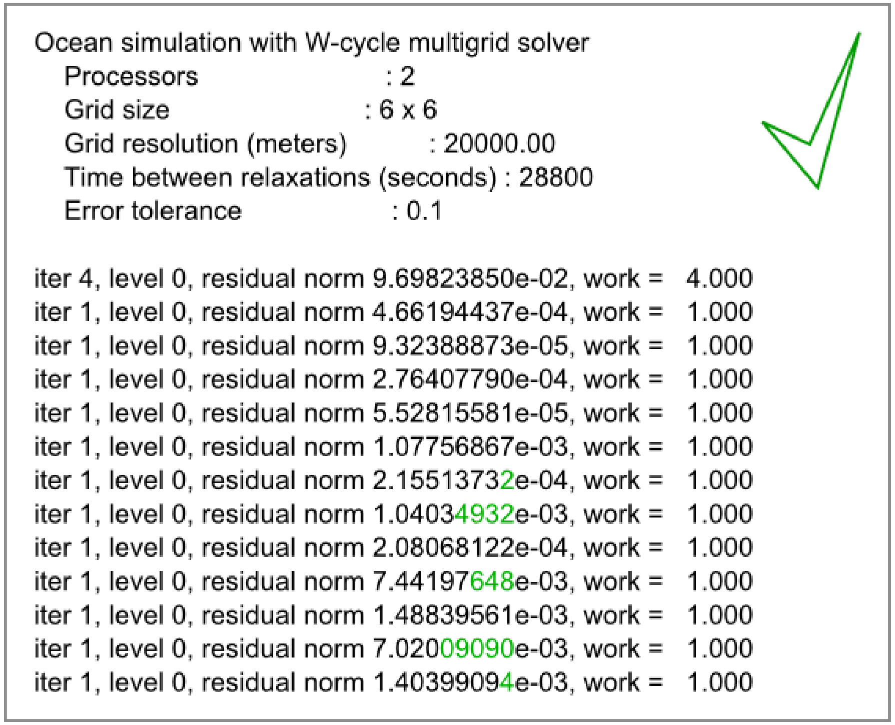
- Variable at `0x72014: global->psibi`

```
516: /*LOCK(locks->psibilock)*/
517:   global->psibi = global->psibi + psibipriv;
218: /*UNLOCK(locks->psibilock)*/
```

```
item0: previous modify (6) at 1405
(6,kNone).kOnVirtWrite(0) @00072014
@000199dc: slave1.C:517
===
item1: current visit (4) at 19913
(4,kNone).kOnVirtRead(0) @00072014
@000199bc: slave1.C:517
```

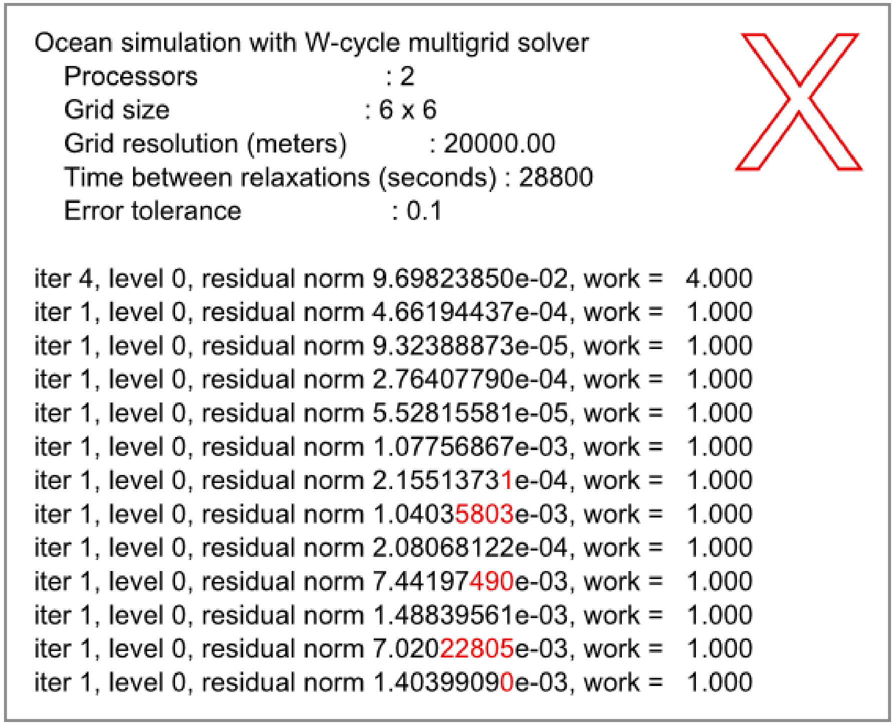


E.g., Result of Exploring Bugs in OCEAN


✓

```
Ocean simulation with W-cycle multigrid solver
   Processors                      : 2
   Grid size                     : 6 x 6
   Grid resolution (meters)          : 20000.00
   Time between relaxations (seconds) : 28800
   Error tolerance                   : 0.1

iter 4, level 0, residual norm 9.69823850e-02, work =   4.000
iter 1, level 0, residual norm 4.66194437e-04, work =   1.000
iter 1, level 0, residual norm 9.32388873e-05, work =   1.000
iter 1, level 0, residual norm 2.76407790e-04, work =   1.000
iter 1, level 0, residual norm 5.52815581e-05, work =   1.000
iter 1, level 0, residual norm 1.07756867e-03, work =   1.000
iter 1, level 0, residual norm 2.15513732e-04, work =   1.000
iter 1, level 0, residual norm 1.04034932e-03, work =   1.000
iter 1, level 0, residual norm 2.08068122e-04, work =   1.000
iter 1, level 0, residual norm 7.44197648e-03, work =   1.000
iter 1, level 0, residual norm 1.48839561e-03, work =   1.000
iter 1, level 0, residual norm 7.02009090e-03, work =   1.000
iter 1, level 0, residual norm 1.40399094e-03, work =   1.000
```


✗

```
Ocean simulation with W-cycle multigrid solver
   Processors                      : 2
   Grid size                     : 6 x 6
   Grid resolution (meters)          : 20000.00
   Time between relaxations (seconds) : 28800
   Error tolerance                   : 0.1

iter 4, level 0, residual norm 9.69823850e-02, work =   4.000
iter 1, level 0, residual norm 4.66194437e-04, work =   1.000
iter 1, level 0, residual norm 9.32388873e-05, work =   1.000
iter 1, level 0, residual norm 2.76407790e-04, work =   1.000
iter 1, level 0, residual norm 5.52815581e-05, work =   1.000
iter 1, level 0, residual norm 1.07756867e-03, work =   1.000
iter 1, level 0, residual norm 2.15513731e-04, work =   1.000
iter 1, level 0, residual norm 1.04035803e-03, work =   1.000
iter 1, level 0, residual norm 2.08068122e-04, work =   1.000
iter 1, level 0, residual norm 7.44197490e-03, work =   1.000
iter 1, level 0, residual norm 1.48839561e-03, work =   1.000
iter 1, level 0, residual norm 7.02022805e-03, work =   1.000
iter 1, level 0, residual norm 1.40399090e-03, work =   1.000
```

```
src/RandomSwapBugFinder.cc:299 : bug occurs when events happen in this order:
first event: 0xc170f508 (4,kNone).kOnVirtRead(0) @00072014
@000199bc: slave1.C:517
second event: 0xc1702d48 (6,kNone).kOnVirtWrite(0) @00072014
@000199dc: slave1.C:517
```

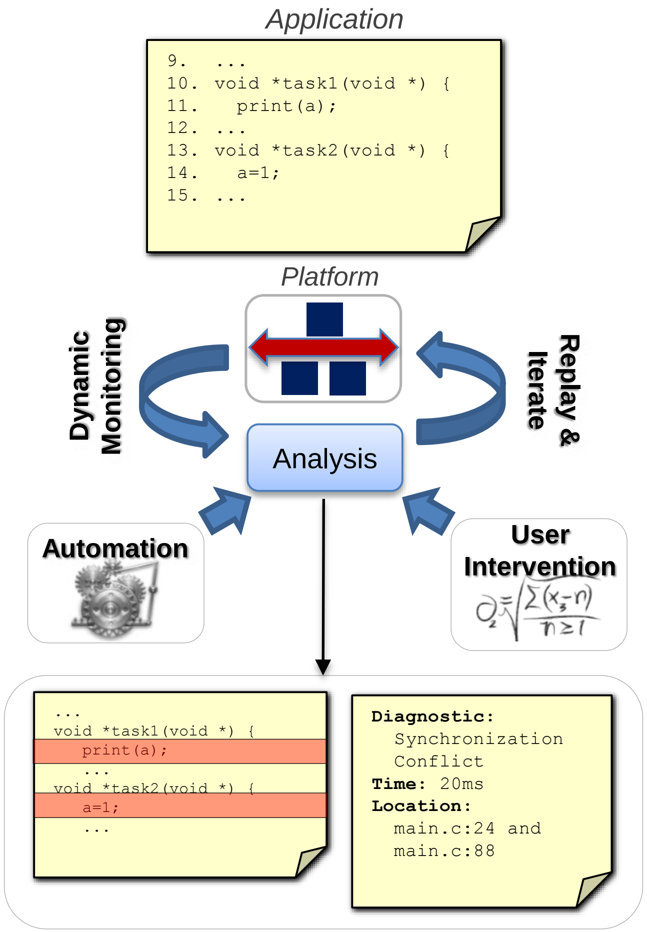
→The bug was found after one iteration.

Conclusions

- **MPSoC debuggers should:**
  - Facilitate intuitive ways to catch and identify system-wide bugs
  - Explore different concurrent interleavings

- **VPs + Concurrency Analysis → Good recipe to deal with concurrency bugs**

- **ICE's event-based debugging:**
  - Retargetability
  - Abstraction
  - Automation
  - Scalability



```
...
void *task1(void *) {
  print(a);
...
void *task2(void *) {
  a=1;
...
```

```
...
void *task1(void *) {
  print(a);
  ...
void *task2(void *) {
  a=1;
  ...
```

```
Diagnostic:
  Synchronization
  Conflict
Time: 20ms
Location:
  main.c:24 and
  main.c:88
```

Thanks!
&
Questions?