

# Deterministic Replay and Reverse Debugging for QEMU

P. Dovgalyuk

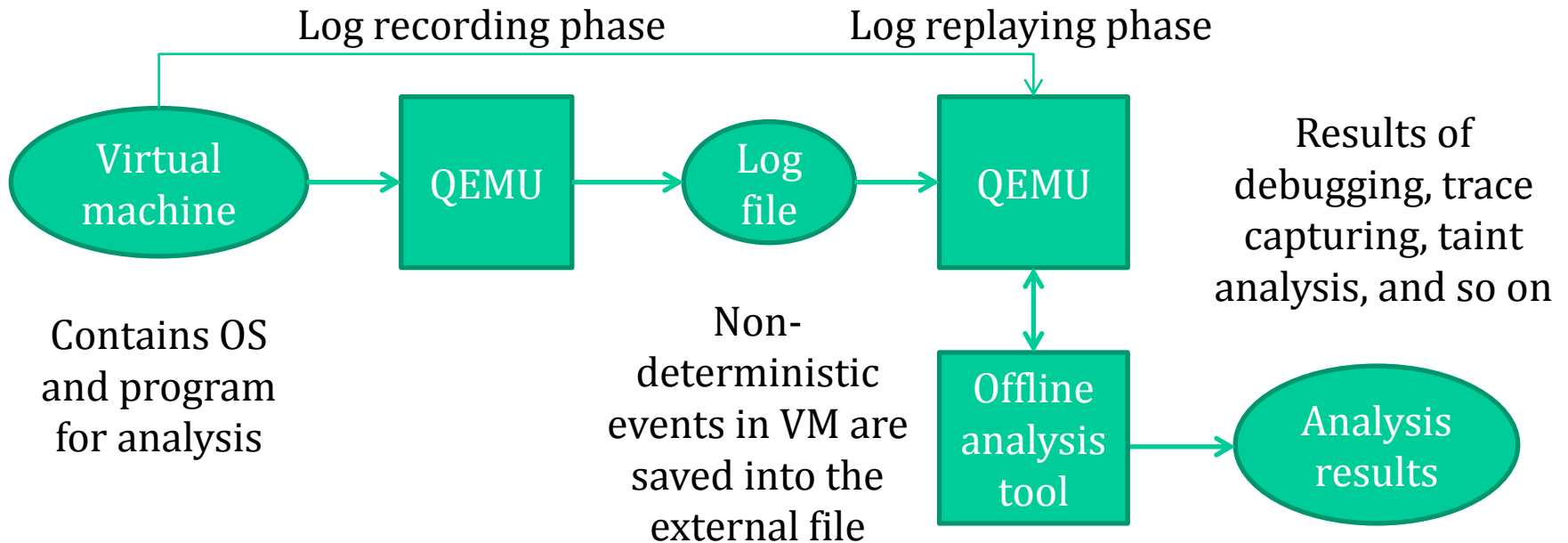
Novgorod State University

Institute for System Programming  
of the Russian Academy of Sciences

# Our projects

- Working on QEMU projects since 2010 (version 0.13)
- Software analysis for x86
- Deterministic replay
- Reverse debugging
- Several upgrades (0.13→0.15→1.0→1.5→2.1)
- Deterministic replay was presented at CSMR 2012

# How it works



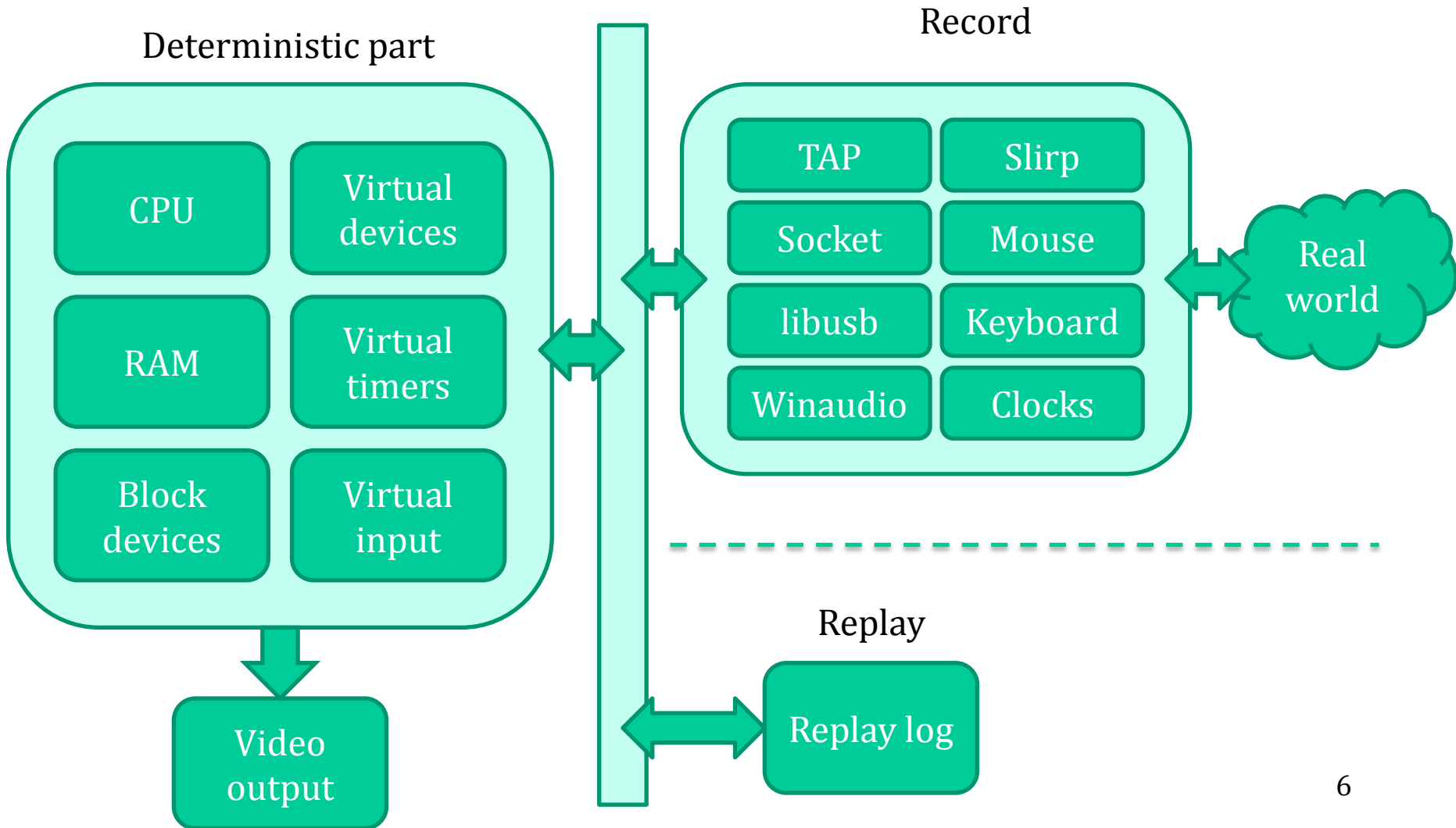
# Deterministic replay applications

- No intrusion and overhead
  - Profiling
  - Taint analysis
  - Offline dynamic analysis
  - Analysis of the real-time applications
- Deterministic
  - Replay and reverse debugging
  - Debugging in complex environment
  - Finding Heisenbugs
  - Debugging of new virtual platforms and virtual devices for QEMU

# Deterministic replay and reverse debugging

- Supports x86, x64, ARM
  - User interface is the same for all platforms
- Works on Windows and Linux hosts
- Whole system debugging
  - Allows debugging system-level code
- Non-intrusive analysis
  - Debugger does not affect on target program
- Offline log analysis
  - Suitable for analysis of network and other real-time applications due to the low recording overhead
- Log file is independent of a host-machine
  - Bug reproducing scenario may be recorded on one machine and replayed on another

# Replaying the simulator



# Deterministic replay implementation

- High-level non-deterministic events are written into the log in record mode
  - mouse, keyboard, hardware clock, network packets, USB packets
- Non-deterministic events read from the log in replay mode are used instead of real inputs
- Disk I/O is deterministic, because of using unchanged disk image
- Thread pool tasks execution sequence is serialized to make it deterministic

# QEMU changes

- Added log recording subsystem
- Target-specific dynamic translators were changed to add instructions counting
- Target-independent components were changed to record incoming events



# Instructions counting: icount issues

- Different counting for REP instructions in single step and normal modes
  - icount is not incremented for the last (ecx=0) iteration in normal mode
- Incorrect when using breakpoints through gdb
  - icount is incremented for non-executed instruction, which located at the breakpoint address
- Seem to be x86-specific

# Instructions counting

```
check tcg_exit_req
```

```
000f2e0e: push  %ebx
```

```
++icount
```

```
000f2e0f: sub   $0x2c,%esp
```

```
++icount
```

```
000f2e12: movl  $0xf64bc,0x4(%esp)
```

```
++icount
```

```
000f2e1a: movl  $0xf4d50,(%esp)
```

```
++icount
```

```
000f2e21: call  0xf1ca0
```

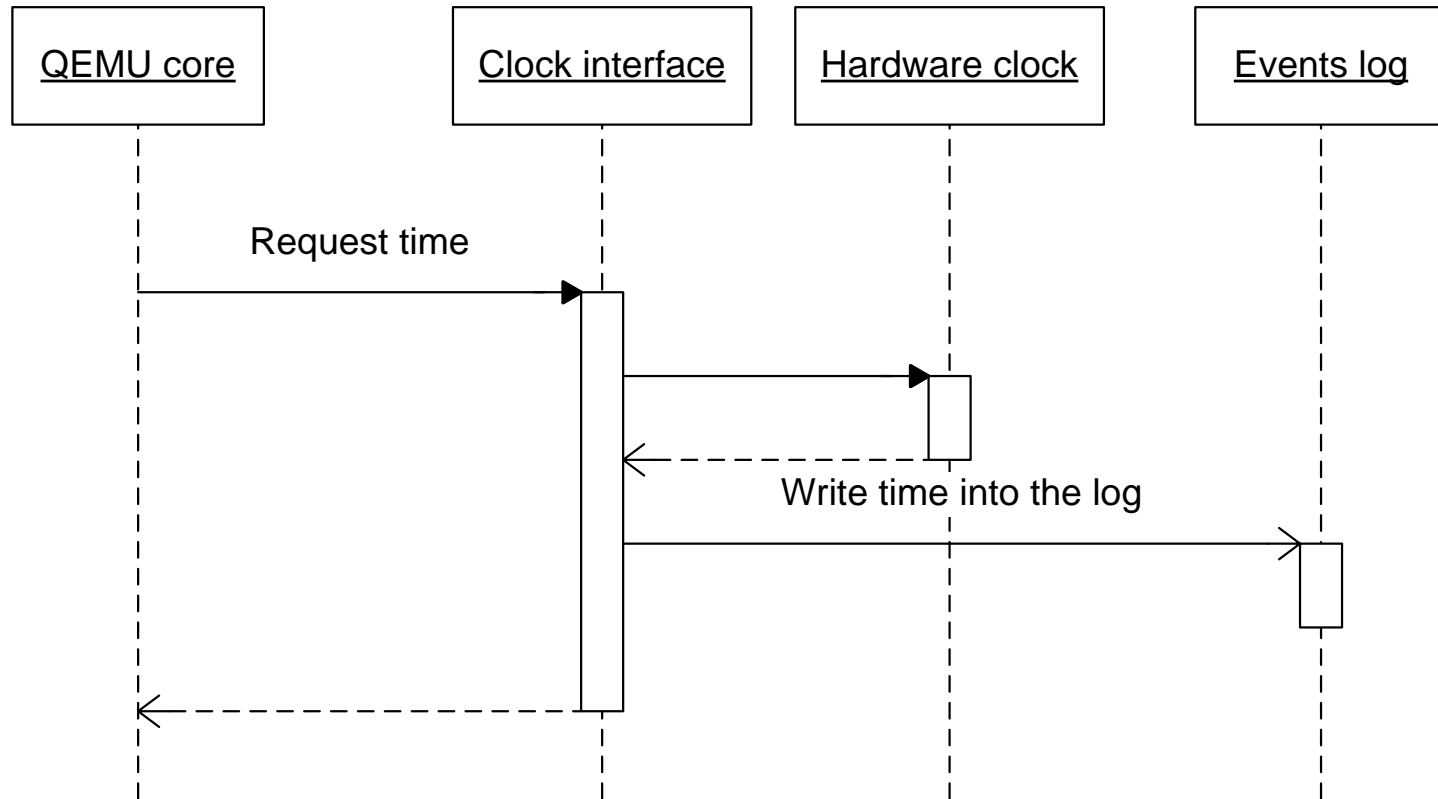
```
++icount
```

I'm going to create my own icount with black jack and hookers.



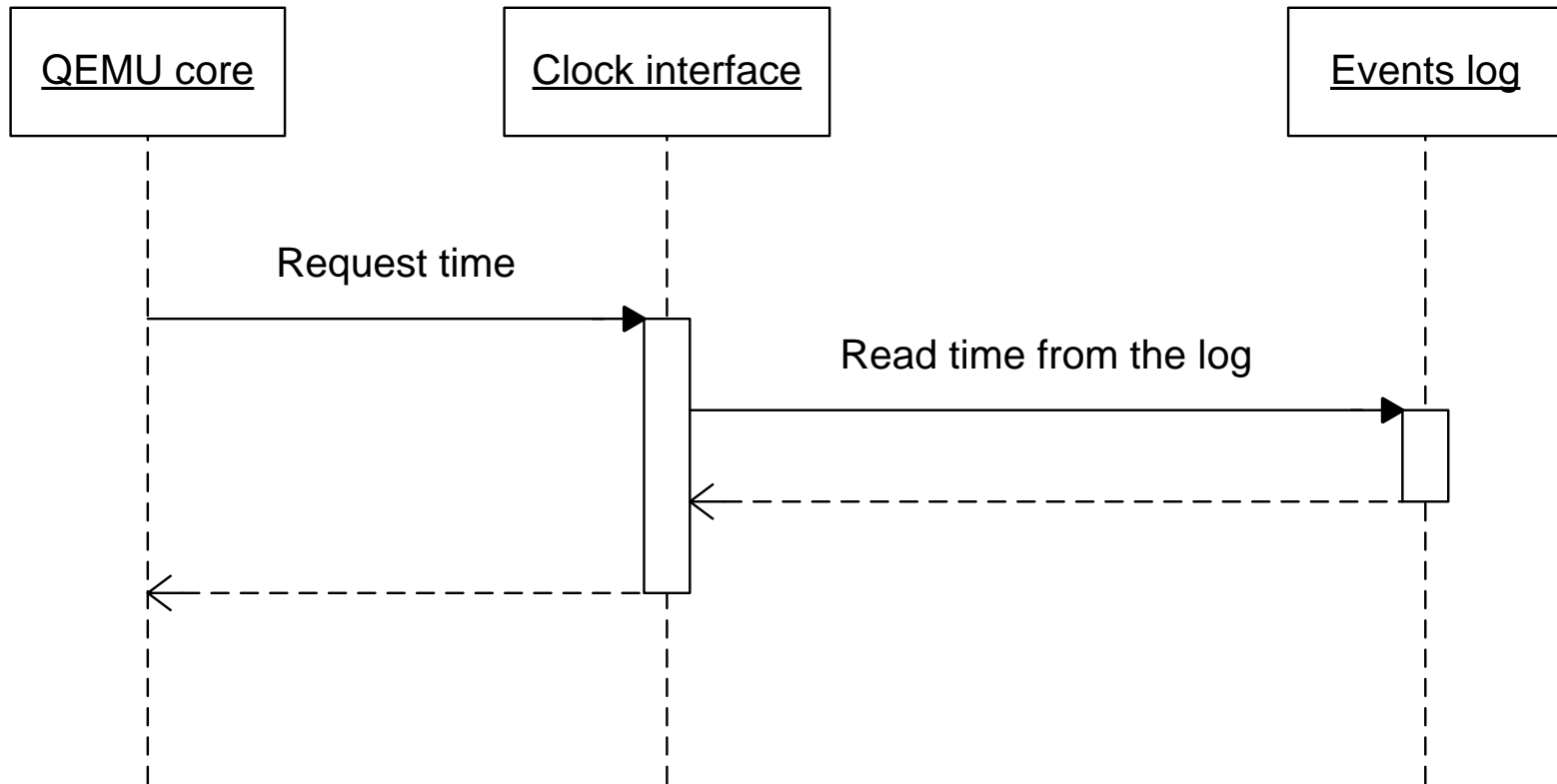
# Hardware clock and timers

- Saving time into the events log



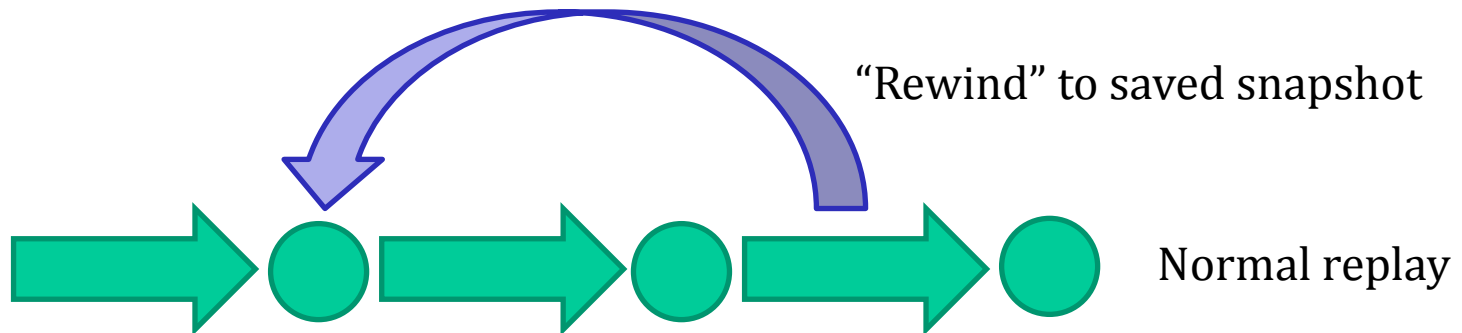
# Hardware clock and timers

- Reading time from the events log



# Checkpointing

- Making periodic snapshots for convenient replay debugging
- Allow navigation in the execution scenario
- Required for reverse execution




# User input and passthru devices

- Keyboard
- Mouse
- Audio card
- Network adapters
- Serial interface
- USB devices

# Reverse debugging

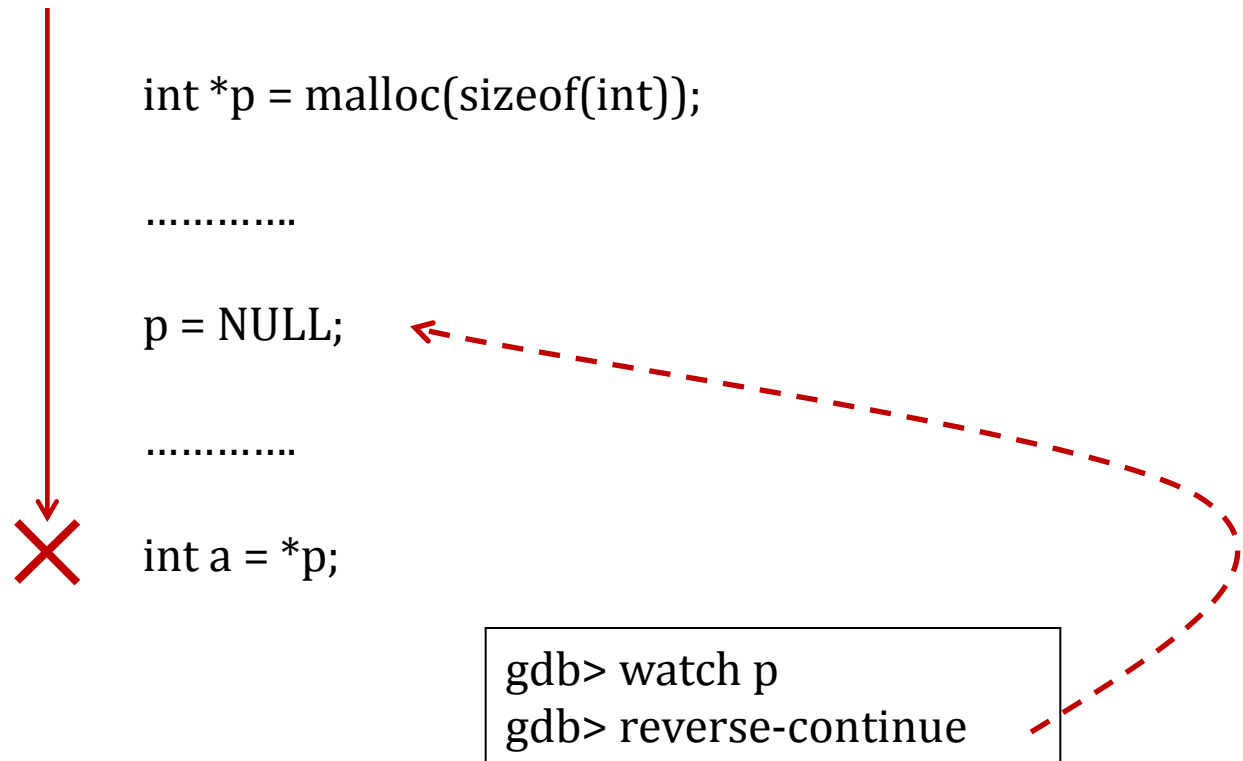
- Using checkpoints for faster rewind to the desired moment of execution
- GDB reverse debugging commands
  - reverse-continue
  - reverse-step
  - reverse-stepi
  - reverse-next
  - reverse-nexti
  - reverse-finish

# Reverse debugging

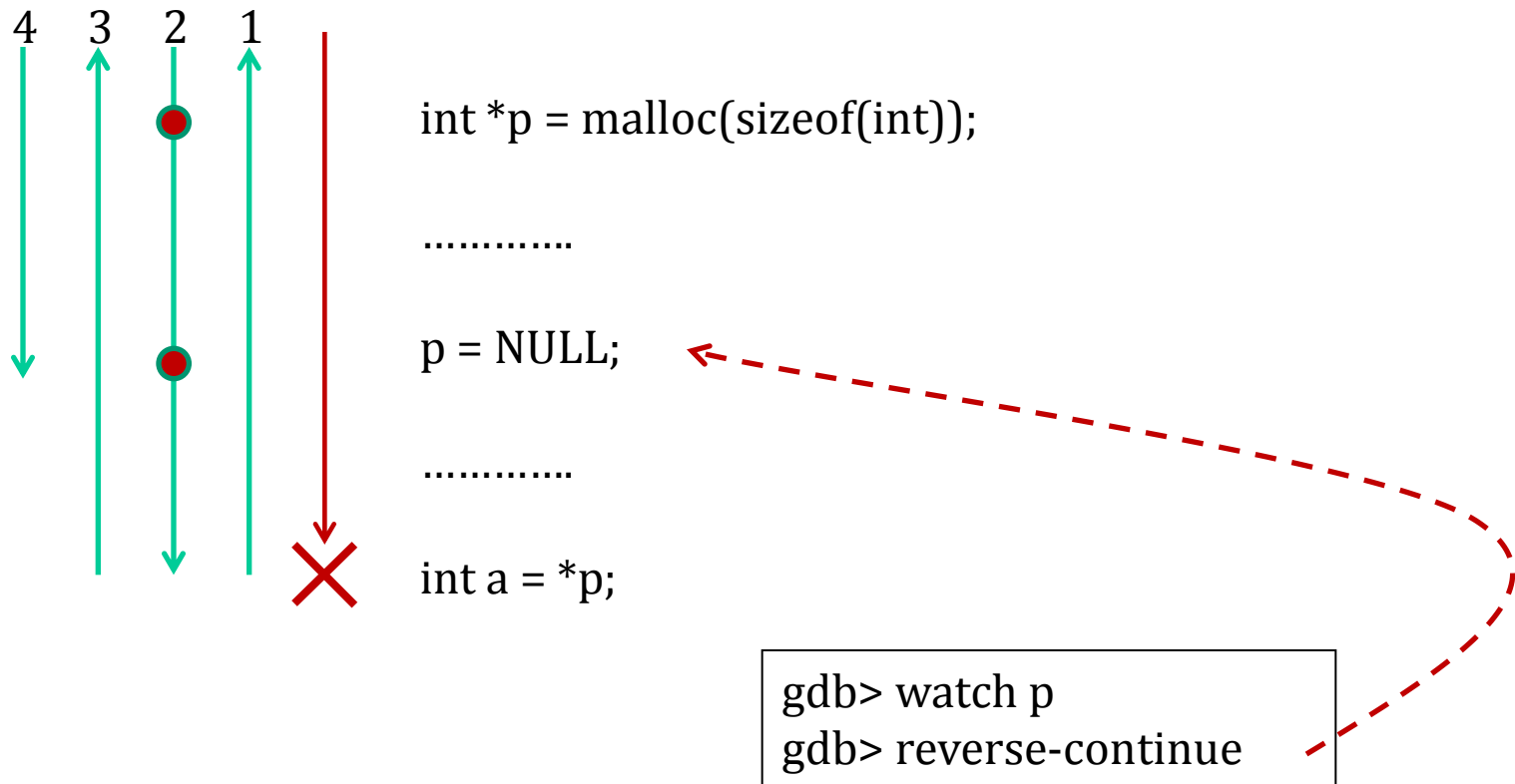
```
int *p = malloc(sizeof(int));  
.....  
p = NULL;  
.....  
 int a = *p;
```



# Reverse debugging



# Reverse debugging



# Evaluation

- Environment
  - Core i7 with 8G of RAM
  - Windows 7 x64
- Virtual machine
  - i386 with 128M of RAM
- Three testing scenarios
  - Windows XP loading
  - Download of 8M file
  - Compressing 8M file with gzip

# Evaluation

	Windows XP	File download	Gzip
Instructions count	5 billions	1.4 billions	3 billions
Normal execution time	69 sec	41 sec	12 sec
Log size	15 M	2.1 M*	1.1 M
Log size per 1000 instructions	3 bytes	1.6 bytes	0.4 bytes
Recording overhead	1.5x	1.1x	3.2x
Replaying overhead	4.5x	2.9x	12.8x

\* Without network traffic

# Results and status

- Found several bugs in QEMU core
- Prepared reverse execution patches
  - about 6000 LOC
  - split into two parts – replay core and reverse debugging


# Results: applied patches

Date	Hash	Description
2014-09-26	5a6e8ba	kvmvapic: fix migration when VM paused and when not running Windows
2014-09-18	5bde140	target-i386: update fp status fix
2014-09-11	462efe9	gdbstub: init mon_chr through qemu_chr_alloc
2014-09-11	a28fe7e	pckbd: adding new fields to vmstate
2014-09-11	0b10215	mc146818rtc: add missed field to vmstate
2014-09-11	2c9ecde	piix: do not set irq while loading vmstate
2014-09-11	7385b27	serial: fixing vmstate for save/restore
2014-09-11	461a275	parallel: adding vmstate for save/restore
2014-09-11	c0b92f3	fdc: adding vmstate for save/restore
2014-09-11	4603ea0	cpu: init vmstate for ticks and clock offset
2014-09-11	a6dead4	apic_common: vapic_paddr synchronization fix
2014-09-05	6c3bff0	exec: Save CPUState::exception_index field
2013-04-20	089305a	i386 ROR r8/r16 instruction fix
2012-06-15	b75a028	Prevent disk data loss when closing qemu
2011-02-25	c7eb1f0	Fixing network over sockets implementation for win32



# How can you use it? (2/3)

- Reverse debugging through GDB interface
  - user-mode programs
  - drivers
  - kernel

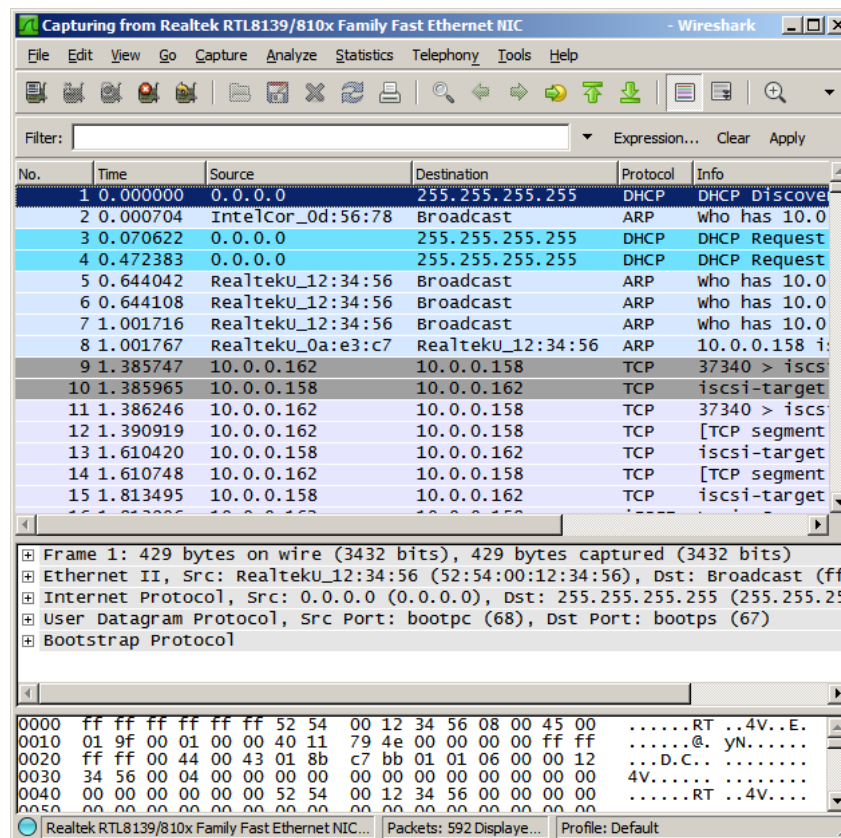


```
initrd-tools: 0.1.73
mount: fs type devfs not supported by kernel
umount: devfs: not mounted
mount: fs type devfs not supported by kernel
umount: devfs: not mounted
pivot_root: No such file or directory
/sbin/init: 426: cannot open dev/console: No such file
Kernel panic: Attempted to kill init!
```



# How can you use it? (3/3)

- Capturing execution data for offline analysis
  - instructions sequence
  - memory accesses
  - network traffic



# Future work

- Up-streaming the rest of the bug-fixes
- Up-streaming the reverse execution core patches
- Up-streaming reverse debugging patches
- Improving performance
  - Improve instructions counting to increase replaying speed
  - Reduce log saving overhead to improve usability
  - Save additional data for faster transition between states while replaying and debugging
- Creating framework for testing of the deterministic replay