Science, Sharing,and Repeatability in Memory Forensics

Brendan Dolan-Gavitt
Columbia University

whoami

- @moyix
- Did some early work in memory forensics:
  - VAD, registry in memory, GDI, VMI
- Other software: pdbparse, PANDA
- Currently postdoc researcher at Columbia

Memory Forensics as a Scientific Field

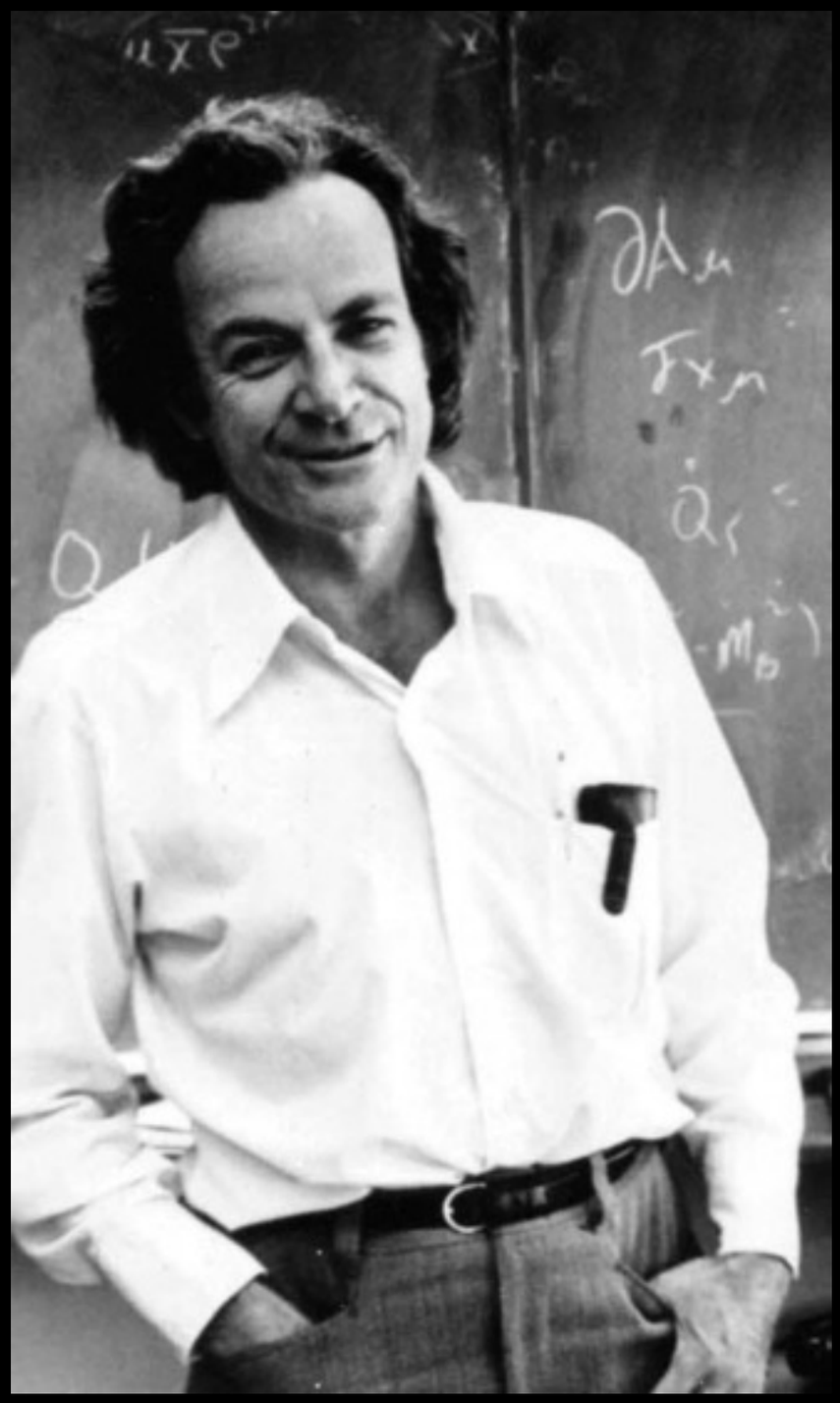
- Still very young
  - First DFRWS memory forensics challenge less than a decade ago!
- The gap between research and practice is not very large

Scientific Method

- In developing memory forensic tools:
  - Form some hypothesis about software artifacts
    - Investigate target – run experiments, disassemble, etc. to confirm/disprove
  - New understanding becomes crystallized into tools, plugins, etc. that practitioners use

Reproducibility

“The first principle is that you must not fool yourself–and you are the easiest person to fool.”



Reproducibility

- Correctness is critical
- Forensics moves fast, however
  - ~1.5 years between OS versions
  - Just describing results (without code) is very slow
- Validation needs to move quickly too

Sharing

- Note that this is the *Open* Memory Forensics Workshop
- Making code available is critical!
  - Redeveloping from scratch takes too long
  - Direct examination of code is better
- Sharing *data* is also necessary (i.e., memory images for testing)

Reproducibility in Memory Forensics

- Standard reference images
  - NIST CFReDS, DFRWS challenge images
- Tool testing (NIST)
- Work on validating acquisition (Vömel & Stüttgen, 2013)

A Missing Piece

- Many investigations involve *dynamic* analyses of executing programs
  - Particularly malware
- What does it mean to reproduce a malware analysis?

Challenges

- Dynamic analyses depend on a runtime environment
  - Network servers may go down
  - Behavior dependent on software & library versions
  - May trigger on certain dates/times

Record / Replay 

- We want to instead share a *specific execution* of a program

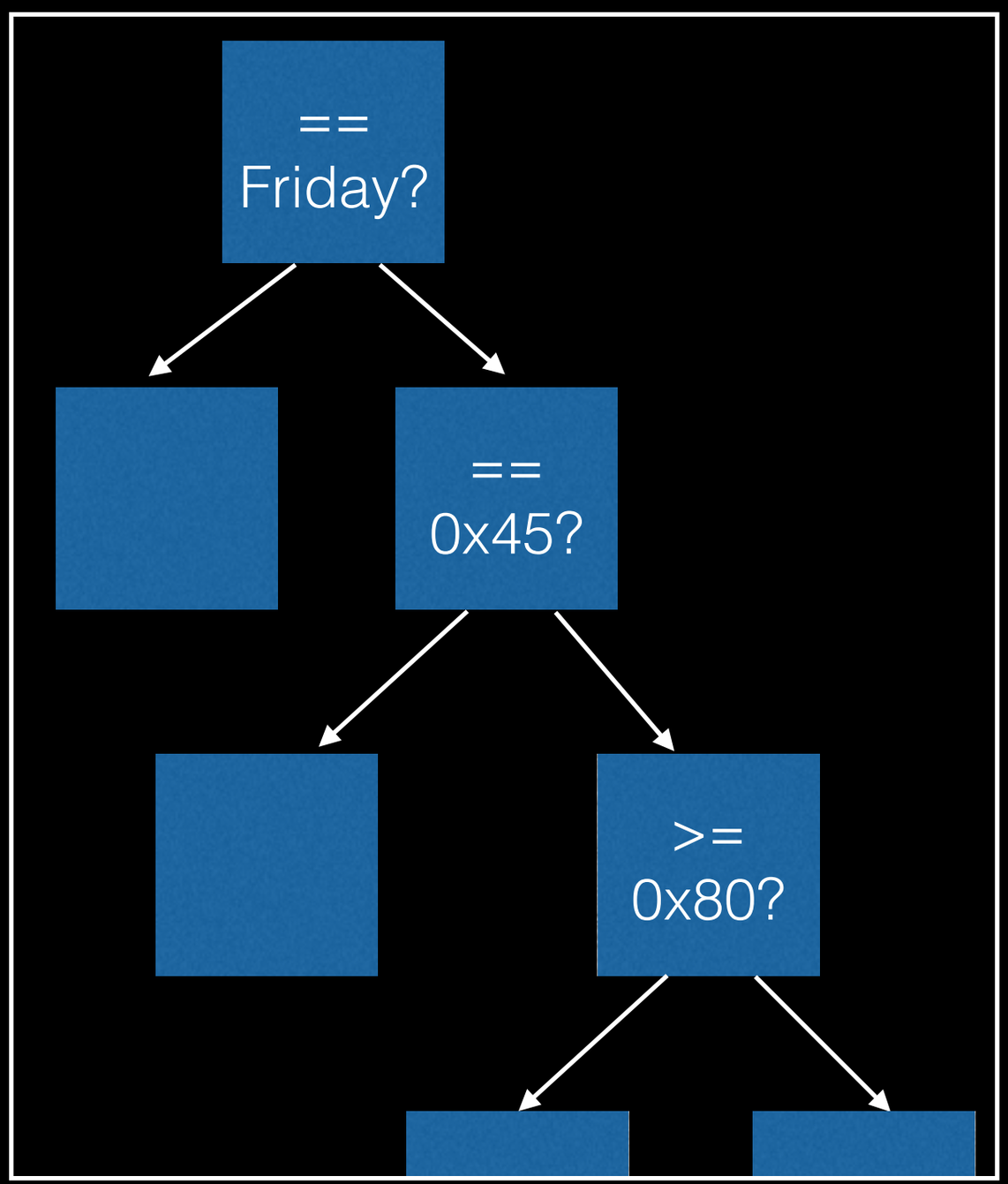
- Observation: if we *record* all the nondeterministic inputs to the system, we can then *replay* the exact execution later

- Technique has been around ~20 years, used mainly for debugging (i.e. reverse execution)

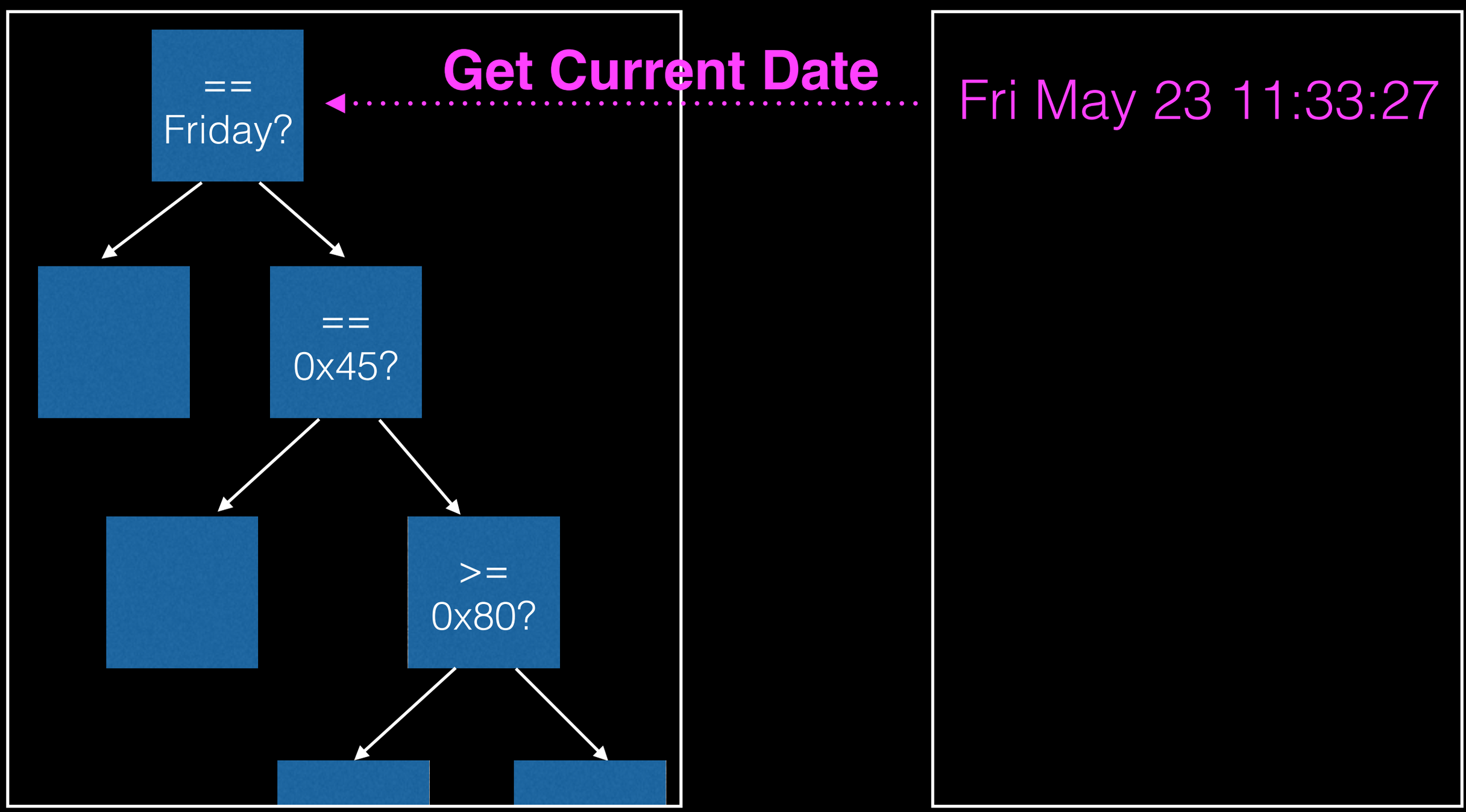
Record / Replay

CPU

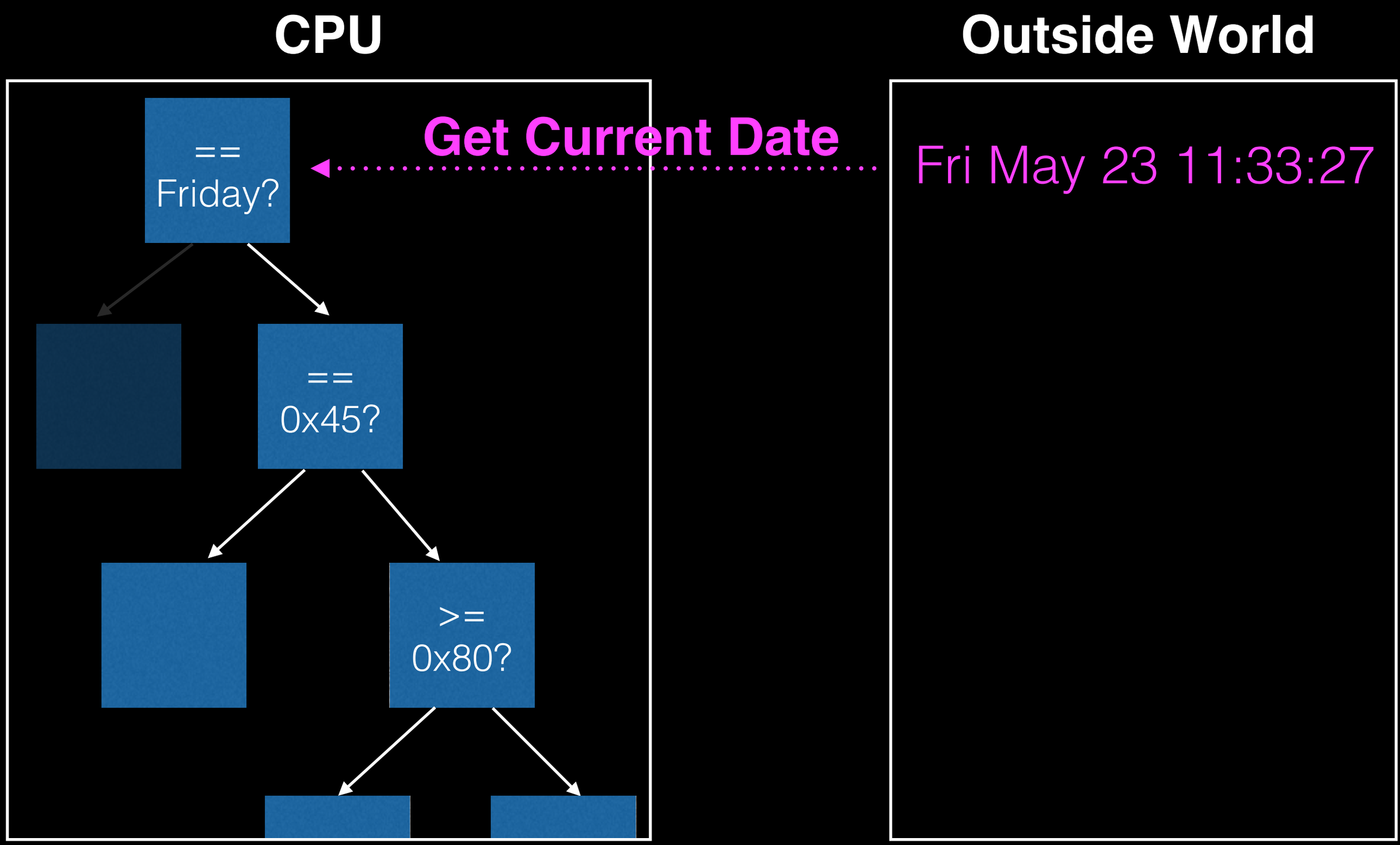
Outside World



Record / Replay



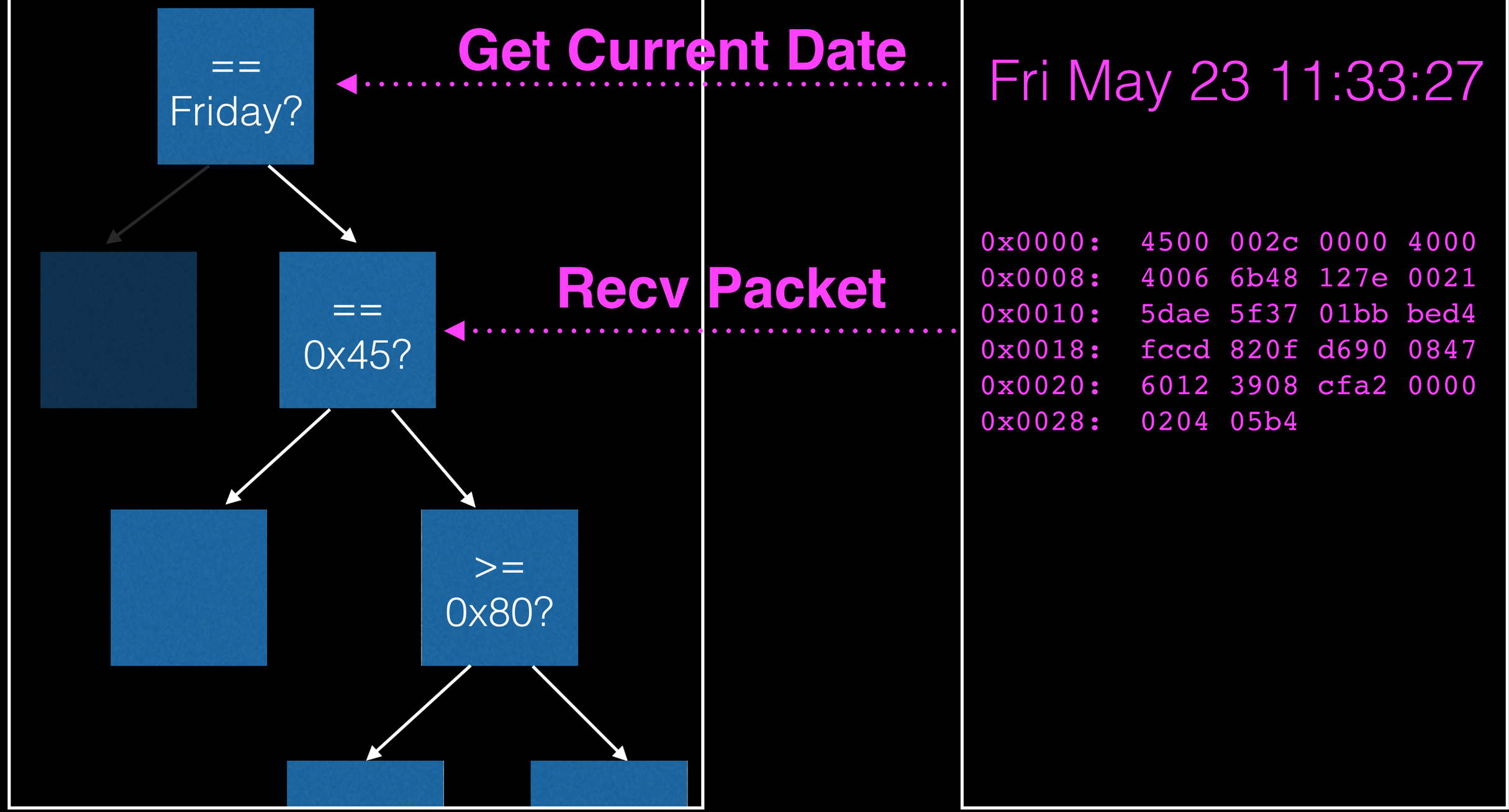
Record / Replay



Record / Replay

CPU

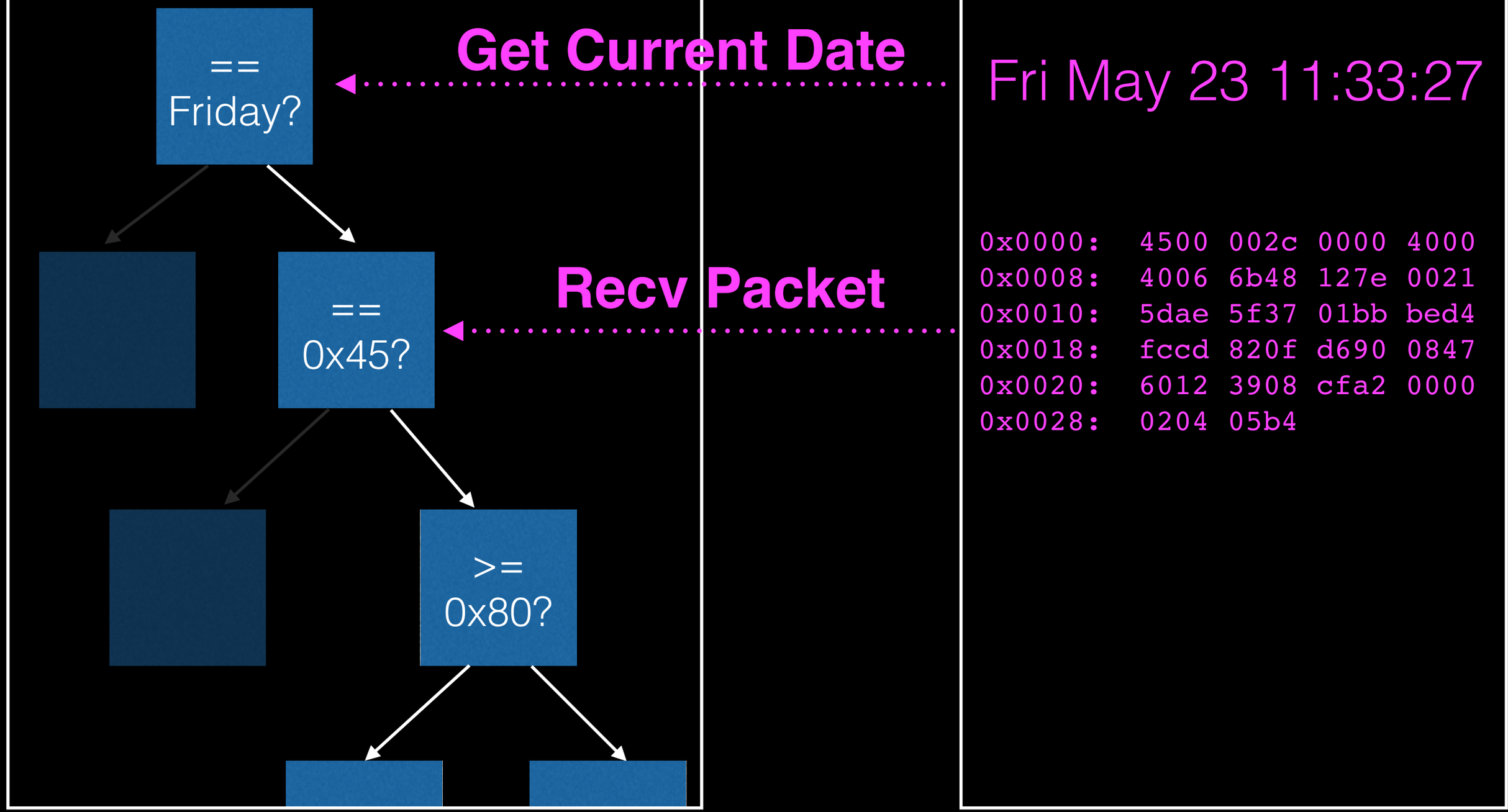
Outside World



Record / Replay

CPU

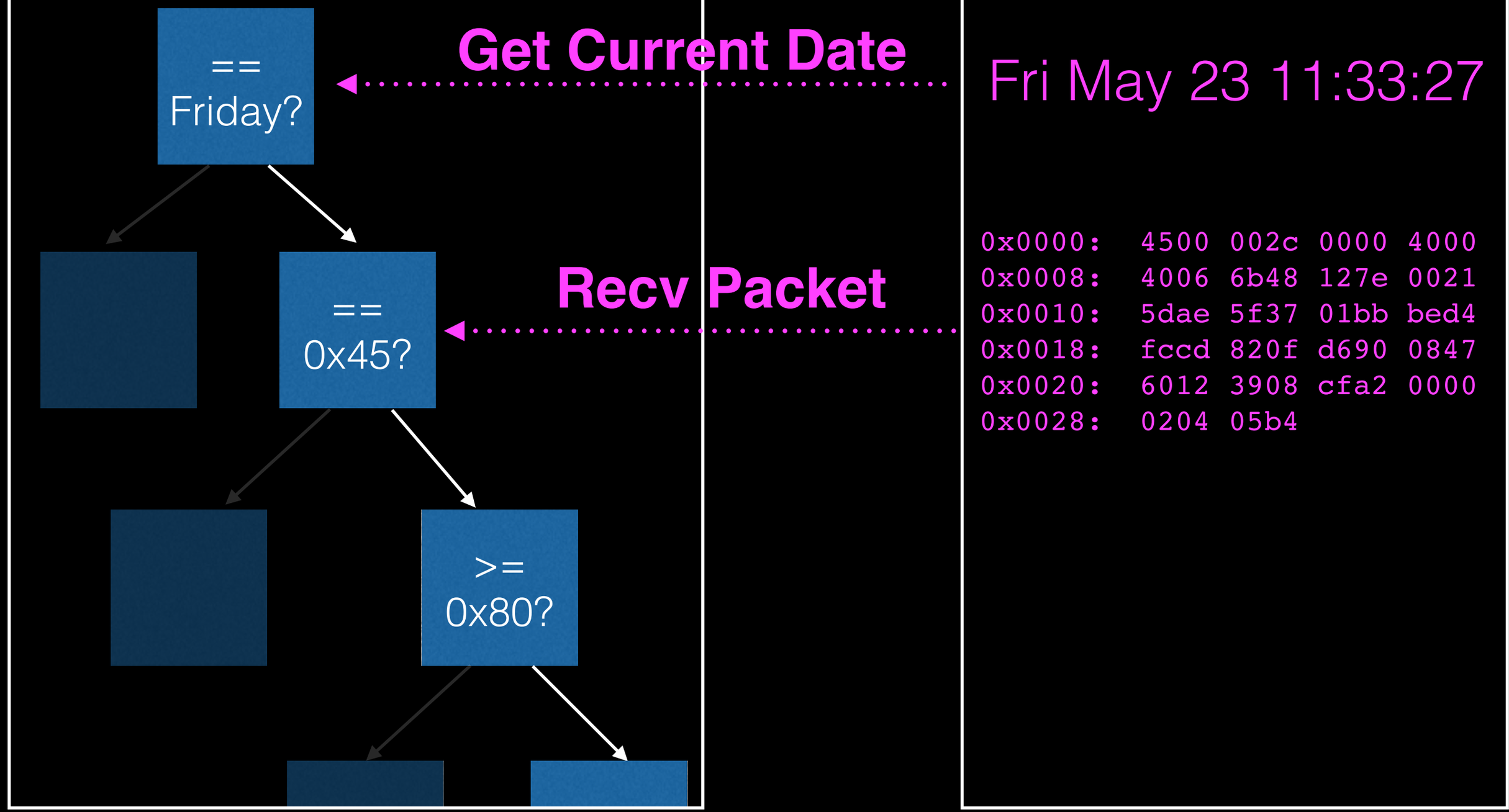
Outside World



Record / Replay

CPU

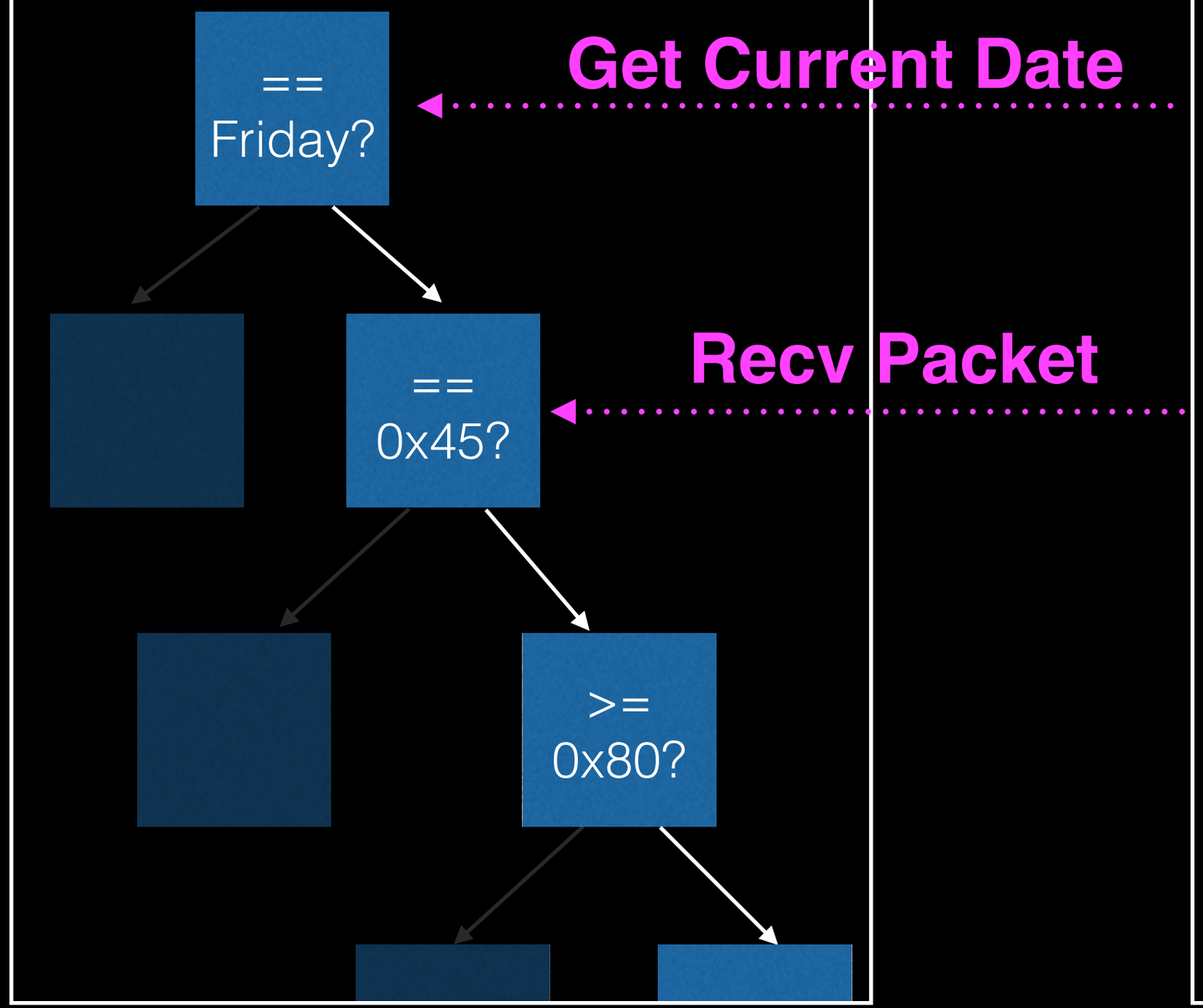
Outside World



Record / Replay

CPU

Outside World



Get Current Date

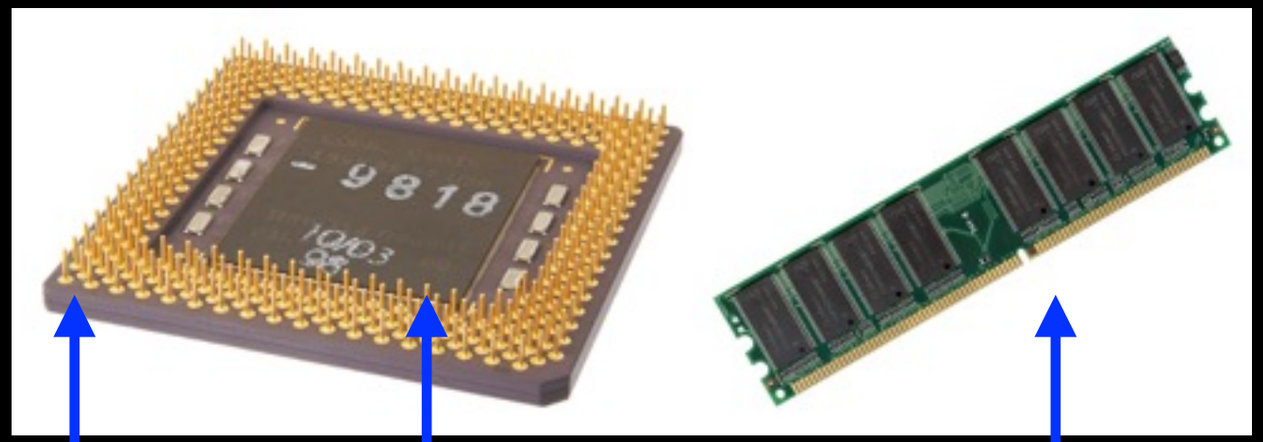
Recv Packet

Fri May 23 11:33:27

```
0x0000:  4500 002c 0000 4000
0x0008:  4006 6b48 127e 0021
0x0010:  5dae 5f37 01bb bed4
0x0018:  fccd 820f d690 0847
0x0020:  6012 3908 cfa2 0000
0x0028:  0204 05b4
```

Record Log

Record / Replay



rdtsc

interrupt

DMA

Time

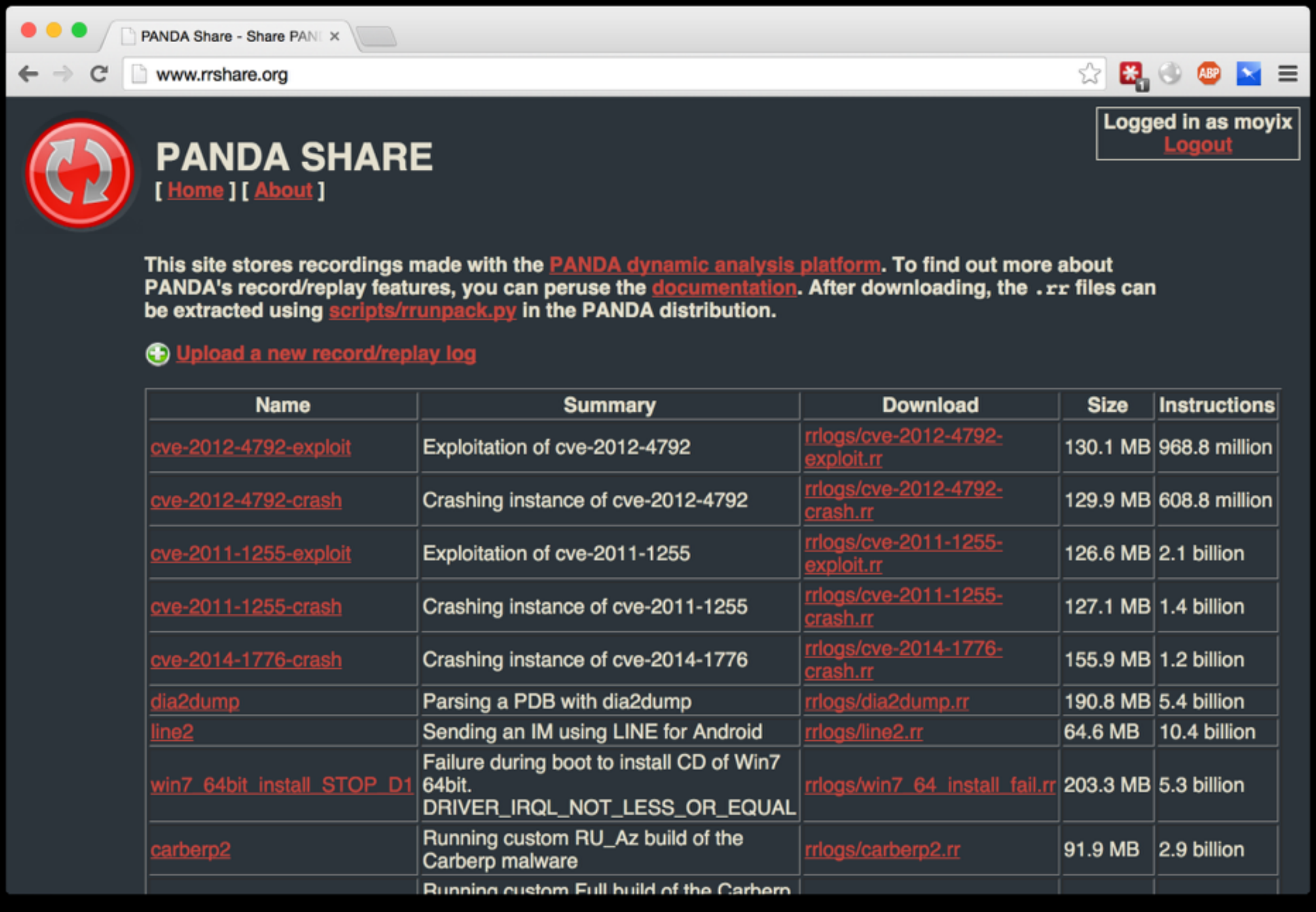
Reproducible Dynamic Analysis with PANDA

- PANDA - Platform for Architecture Neutral Dynamic Analysis
- Supports *shareable* recordings of whole-system execution
- Write *plugins* to analyze replays as they execute

PANDA Model

- Record / replay critical:
  - Heavy analyses don’t disrupt execution
  - Analyses don’t have to worry about memory layout changing between runs

www.rrshare.org



Name	Summary	Download	Size	Instructions
cve-2012-4792-exploit	Exploitation of cve-2012-4792	rrlogs/cve-2012-4792-exploit.rr	130.1 MB	968.8 million
cve-2012-4792-crash	Crashing instance of cve-2012-4792	rrlogs/cve-2012-4792-crash.rr	129.9 MB	608.8 million
cve-2011-1255-exploit	Exploitation of cve-2011-1255	rrlogs/cve-2011-1255-exploit.rr	126.6 MB	2.1 billion
cve-2011-1255-crash	Crashing instance of cve-2011-1255	rrlogs/cve-2011-1255-crash.rr	127.1 MB	1.4 billion
cve-2014-1776-crash	Crashing instance of cve-2014-1776	rrlogs/cve-2014-1776-crash.rr	155.9 MB	1.2 billion
dia2dump	Parsing a PDB with dia2dump	rrlogs/dia2dump.rr	190.8 MB	5.4 billion
line2	Sending an IM using LINE for Android	rrlogs/line2.rr	64.6 MB	10.4 billion
win7_64bit_install_STOP_D1	Failure during boot to install CD of Win7 64bit. DRIVER_IRQL_NOT_LESS_OR_EQUAL	rrlogs/win7_64_install_fail.rr	203.3 MB	5.3 billion
carberp2	Running custom RU_Az build of the Carberp malware	rrlogs/carberp2.rr	91.9 MB	2.9 billion

Log Size

Replay	Instructions	Log Size	Instr/Byte
freebsdboot	9.3 billion	533 MB	17
spotify	12 billion	229 MB	52
haikuurl	8.6 billion	119 MB	72
carberp1	9.1 billion	43 MB	212
win7iessl	8.6 billion	9.4 MB	915
Starcraft	60 million	1.8 MB	33

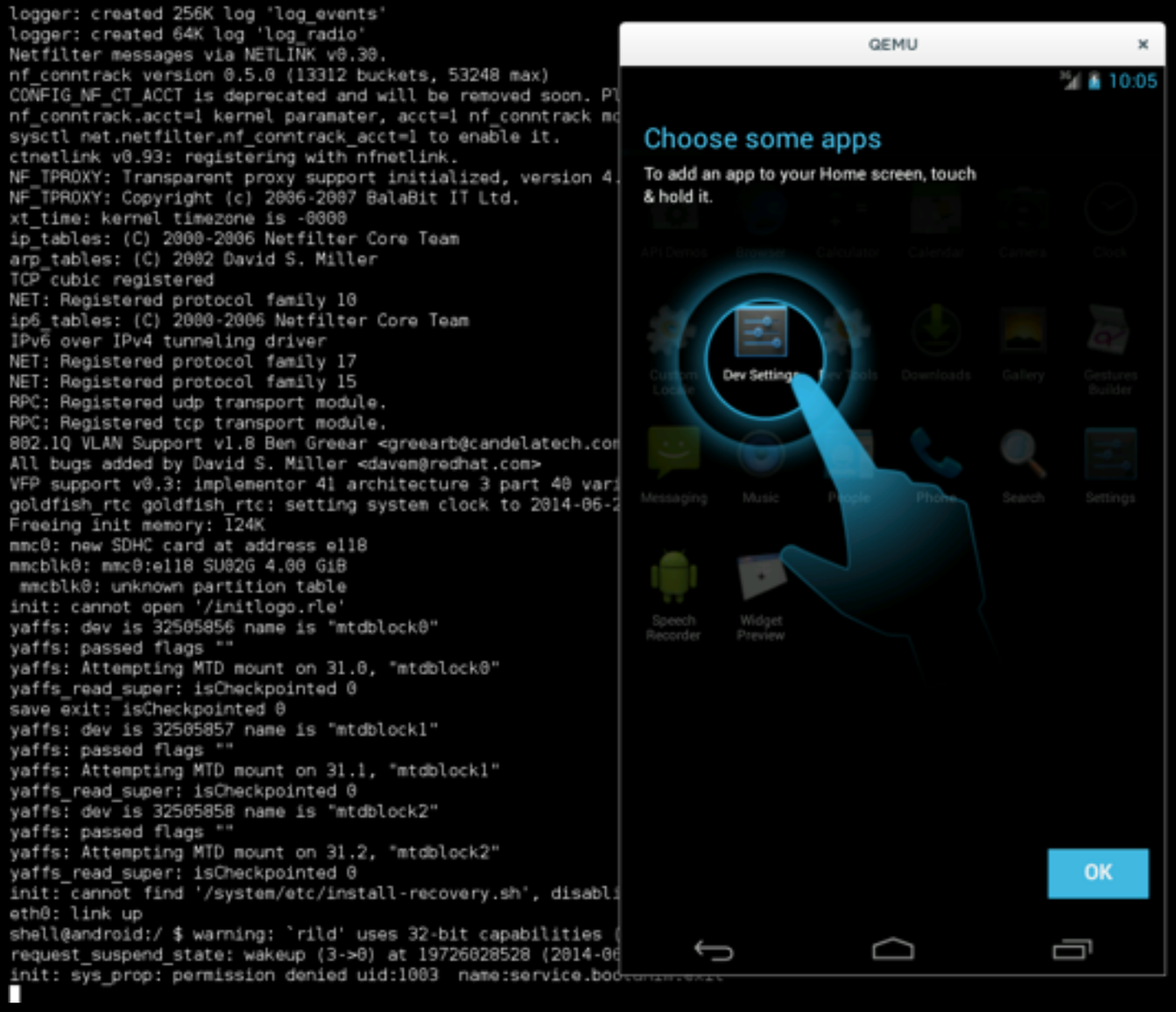
Other PANDA Features

- Android emulation
- Lifting binary code to LLVM
- Taint analysis
- System call tracing

Plugin Architecture

- Extend PANDA by writing plugins (C/C++)
- Implement functions that take action at various *instrumentation points*
- Can also instrument generated code in LLVM mode
- Plugin-plugin interaction: compose simple tools for complex functionality

Android Emulation



- Supports Android 2.x – 4.2
- Can make phone calls, send SMS, run native apps
- Record/replay
- Introspection into Android apps (Dalvik-level) for Android 2.3 (from DroidScope)
- System-level introspection supported on all Android versions

Memory Forensics on Replays

- In some ways, best of both worlds between debugging and memory image analysis
  - All memory accessible throughout entire lifetime of
  - Can pause, dump memory, run Volatility, etc.
  - But can still be triggered by things happening in execution

Conclusions

- Reproducibility is critical to achieving valid forensic results
- For some areas we have decent solutions – code sharing, testing, standard images
- For ephemera such as software execution, we propose *record and replay*, and a system, PANDA

Credits

- PANDA devs
  - Tim Leek (MIT Lincoln Lab)
  - Patrick Hulin (MIT Lincoln Lab)
  - Josh Hodosh (MIT Lincoln Lab)
  - Ryan Whelan (MIT Lincoln Lab)
  - Sam Coe (Northeastern University)
  - Andy Davis (MIT Lincoln Lab)

Contact

- Get in touch! @moyix on Twitter
  brendan@cs.columbia.edu
- Join the mailing list: panda-users@mit.edu
- IRC Channel: #panda-re on Freenode
- Contribute code:
  https://github.com/moyix/panda