Live Disk Forensics on Bare Metal

Hongyi Hu and Chad Spensky

{hongyi.hu,chad.spensky}@ll.mit.edu

Open-Source Digital Forensics Conference 2014

This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Who are we?

- **Chad Spensky**
  - **Lifetime hacker/tinkerer**
  - **Education**
    - BS @ University of Pittsburgh
    - MS @ University of North Carolina
  - **Research staff at MIT Lincoln Laboratory**
  - **3rd time at OSDF Con**
  - **User and modifier of TSK and Volatility**

Who are we?

- **Hongyi Hu**
  - **Computer scientist, tinkerer, lawyer**
  - **Education**
    - S.B., M.Eng @ MIT
    - J.D. @ Boston U.
  - **Research staff at MIT Lincoln Laboratory**
  - **2nd time at OSDF Con**
  - **My photos are not as cool as Chad's ☺**

Agenda

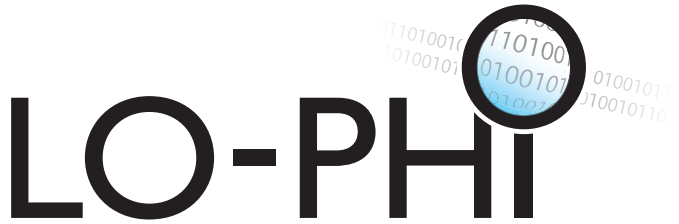
- Overview
- Motivation
- Architecture
- Live Disk Forensics
- Summary
- Future Directions

Overview

- **This talk is a small portion of a larger program**
  - **LO-PHI: Low-Observable Physical Host Instrumentation**

- **Problem Statement**
  - *Instrument physical and virtual machines while introducing as few artifacts as possible.*

- **Goals**
  - Be as difficult-to-detect as possible
  - Develop capabilities for bare-metal machines
  - Produce high-level semantic information



Why?

- **Malware analysis**
  - Malware can actively evade detectable analysis artifacts and may behave differently
- **Cleanroom execution environment**
  - Installing software on the system may not always be an option
    - E.g. Xbox 360
- **Low-artifact debugging**
  - Debuggers can be detected and evaded or mask real-world behavior

How?

- **Instrument interesting tap points in the system**
  - E.g. Hard Disk, Main Memory, CPU, Network
- **Bridge the *semantic gap* to obtain useful information from these raw data sources**
  - E.g. Volatility, Sleuthkit
- **Analyze the raw and semantic data to answer interesting questions**
  - “Is program X malware?”
  - “What files were accessed?”
  - “Is this machine compromised?”

Agenda

- Overview
- Motivation
- Architecture
- Live Disk Forensics
- Summary
- Future Directions

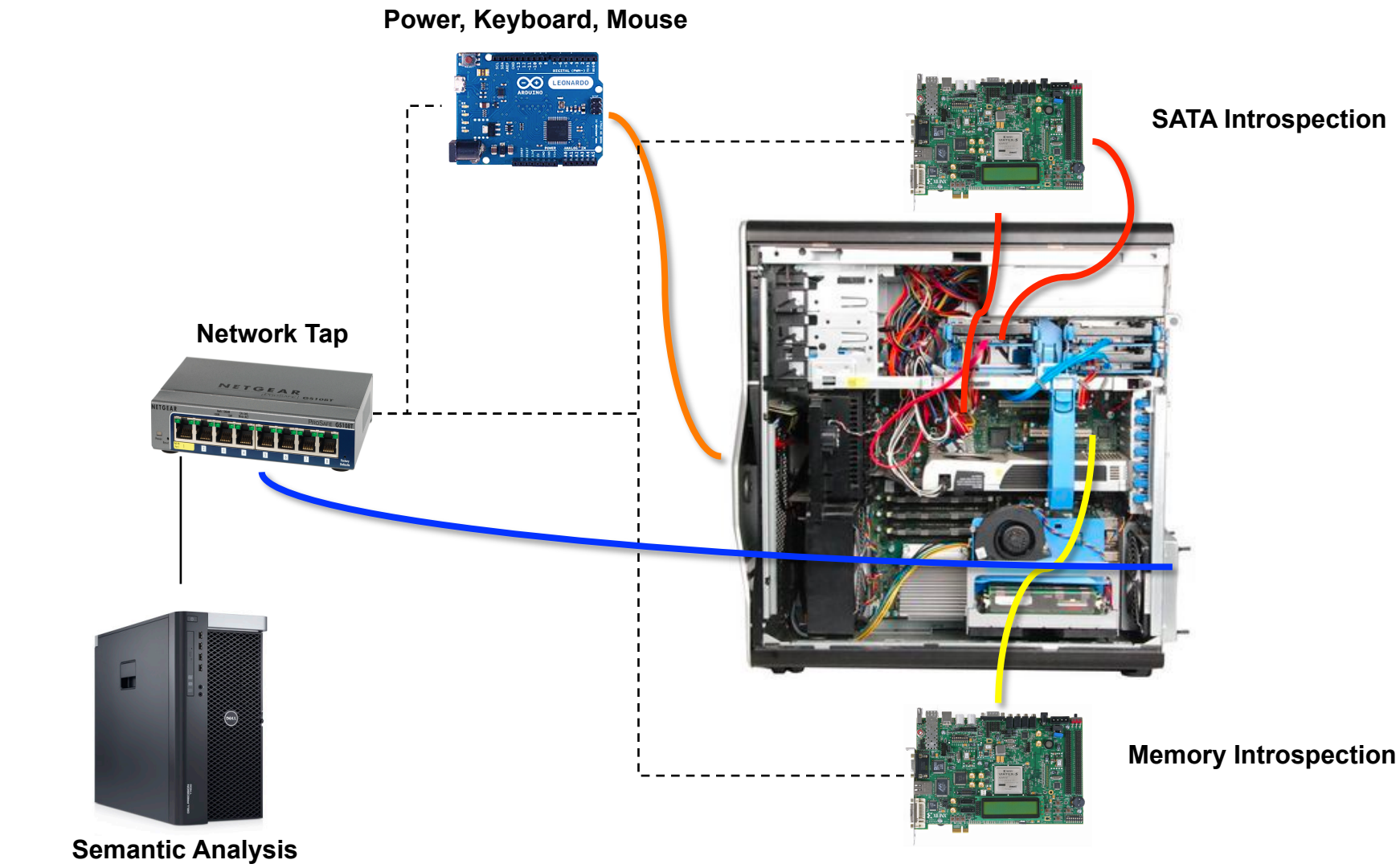
Current Instrumentation

- **Access physical memory**
  - **Virtual:** libvmi
  - **Physical:** PCI & PCI-express FPGA boards
- **Passively monitor disk activity**
  - **Virtual:** Custom hooks into QEMU block driver
  - **Physical:** SATA man-in-the-middle with custom FPGA
- **CPU Instrumentation**
  - **Virtual:** Custom hooks into QEMU KVM
  - **Physical:** Working with Intel's eXtended Debug Port (XDP) and ARM's DSTREAM debugger
- **Actuate inputs**
  - **Virtual:** libvirt
  - **Physical:** Arduino Leonardo

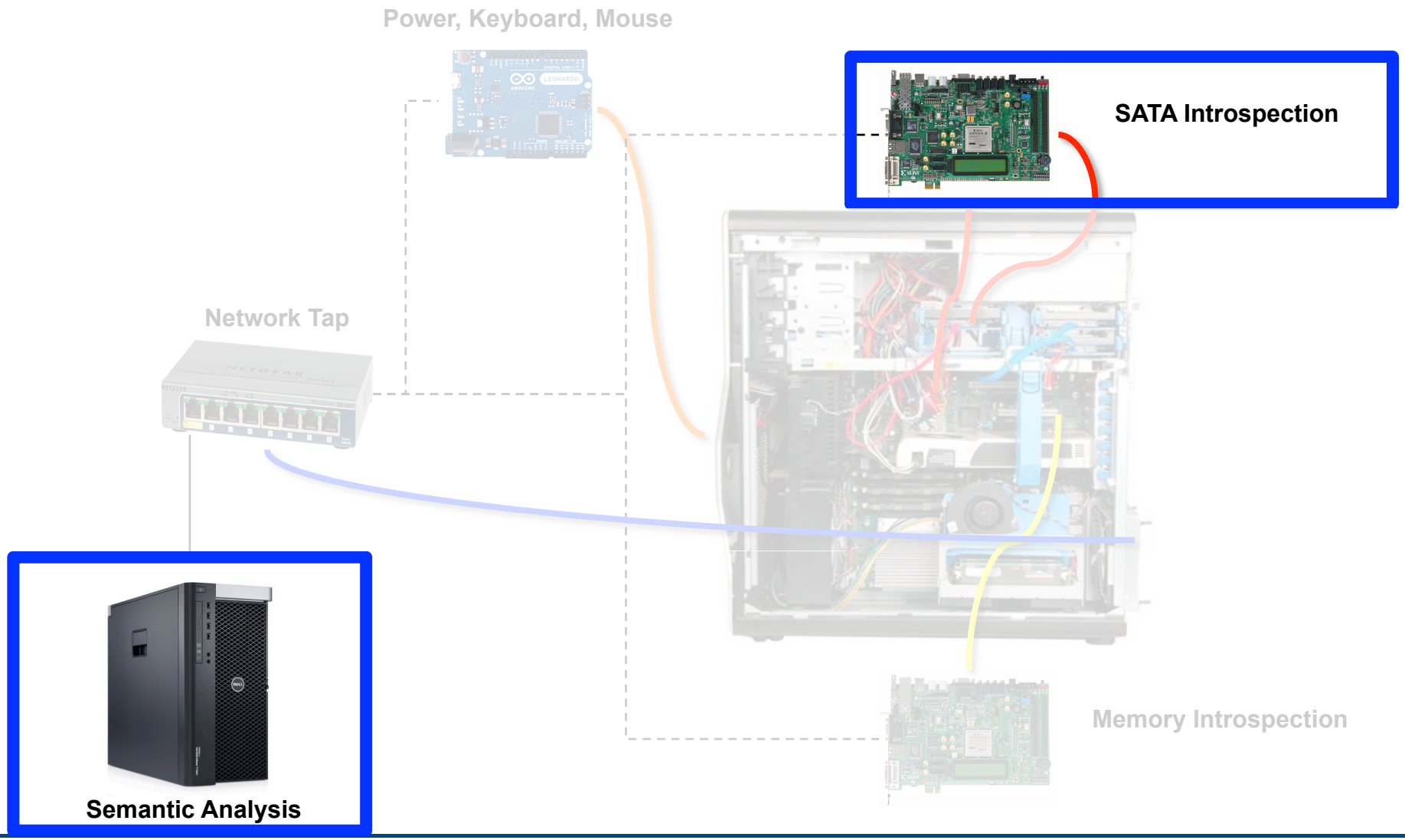
Current Instrumentation

- **Access physical memory**
  - **Virtual:** libvmi
  - **Physical:** PCI & PCI-express FPGA boards
- **Passively monitor disk activity**
  - **Virtual:** Custom hooks into QEMU block driver
  - **Physical:** SATA man-in-the-middle with custom FPGA
- **CPU Instrumentation**
  - **Virtual:** Custom hooks into QEMU KVM
  - **Physical:** Working with Intel's eXtended Debug Port (XDP) and ARM's DSTREAM debugger
- **Actuate inputs**
  - **Virtual:** libvirt
  - **Physical:** Arduino Leonardo

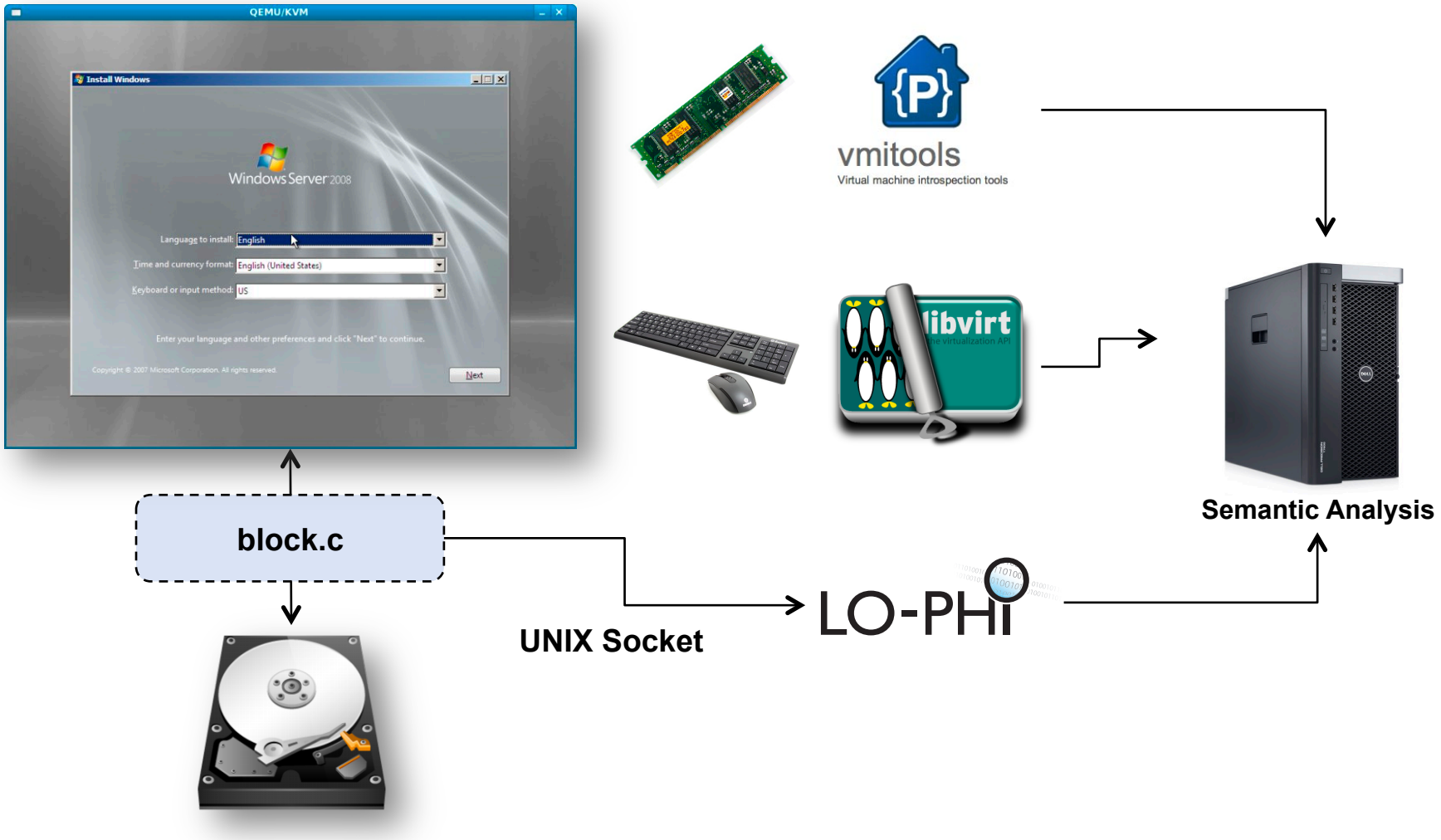
Physical Instrumentation



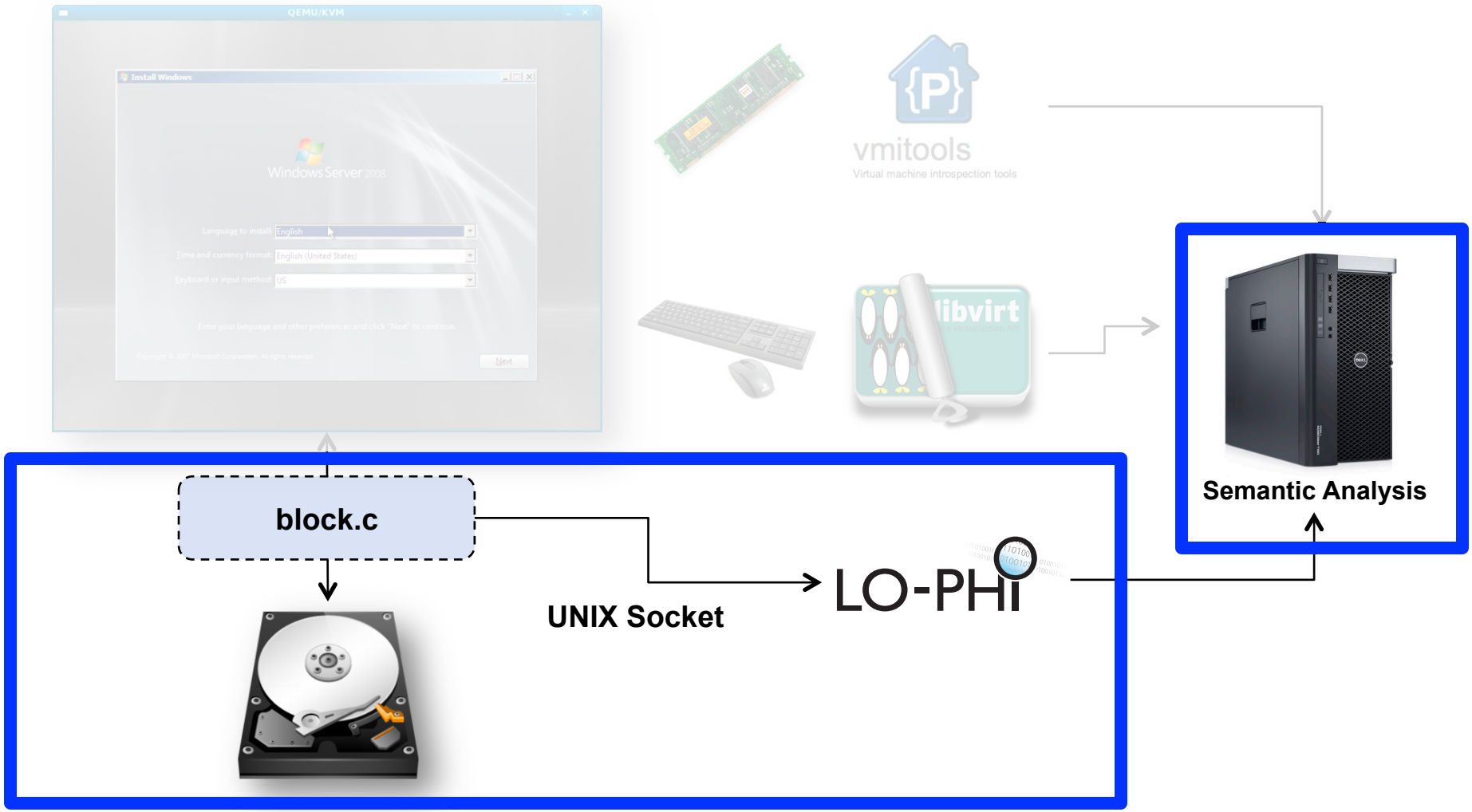
Physical Instrumentation



Virtual Instrumentation



Virtual Instrumentation



Bridging the Semantic Gap

- **Problem**
  - Most forensic tools, i.e. *Volatility* and *Sleuthkit*, assume static offline data
  - We need to analyze live data streams
- **Live Memory Introspection**
  - We were able to optimize Volatility to use a custom address space that speaks directly to our hardware
    - Other code to deal with smearing vs. snapshots etc.
- **Live Disk Forensics**
  - Far less straight-forward, especially on physical HDDs

Agenda

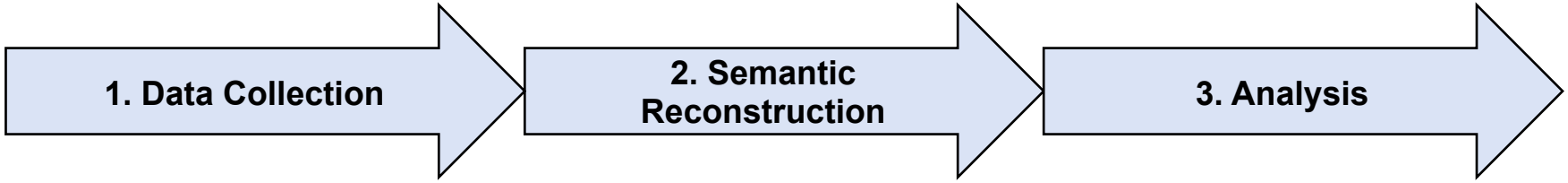
- Overview
- Motivation
- Architecture
- Live Disk Forensics
- Summary
- Future Directions

Live Disk Forensics

1. **Instrumentation: Obtain a stream of disk activity**
   - Read 1 sector from block 0, [DATA]
   - Write 1 sector to block 0, [DATA]
   - . . .

2. **Semantic Gap: Determine the meaning of this read/write**
   - Master Boot Record was modified
   - File read/write/rename/etc.

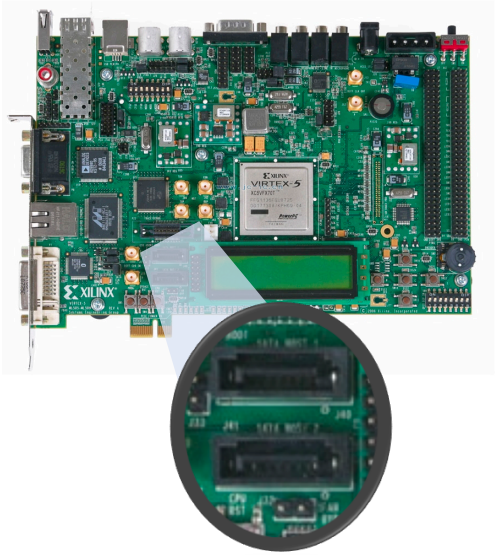
3. **Analyze data**
   - "Is that bad?"



Disk Instrumentation

- **Virtual (QEMU/KVM)**
  - Obtain block, sector count, data, and read/write directly from block driver

- **Physical**
  - Required developing specialized hardware
  - Currently using a Xilinx development board
  - Using off-the-shelf SATA core from Intelliprop
  - Custom code for C&C over Ethernet
  - Outputs raw SATA frames over UDP (~80MB/sec)



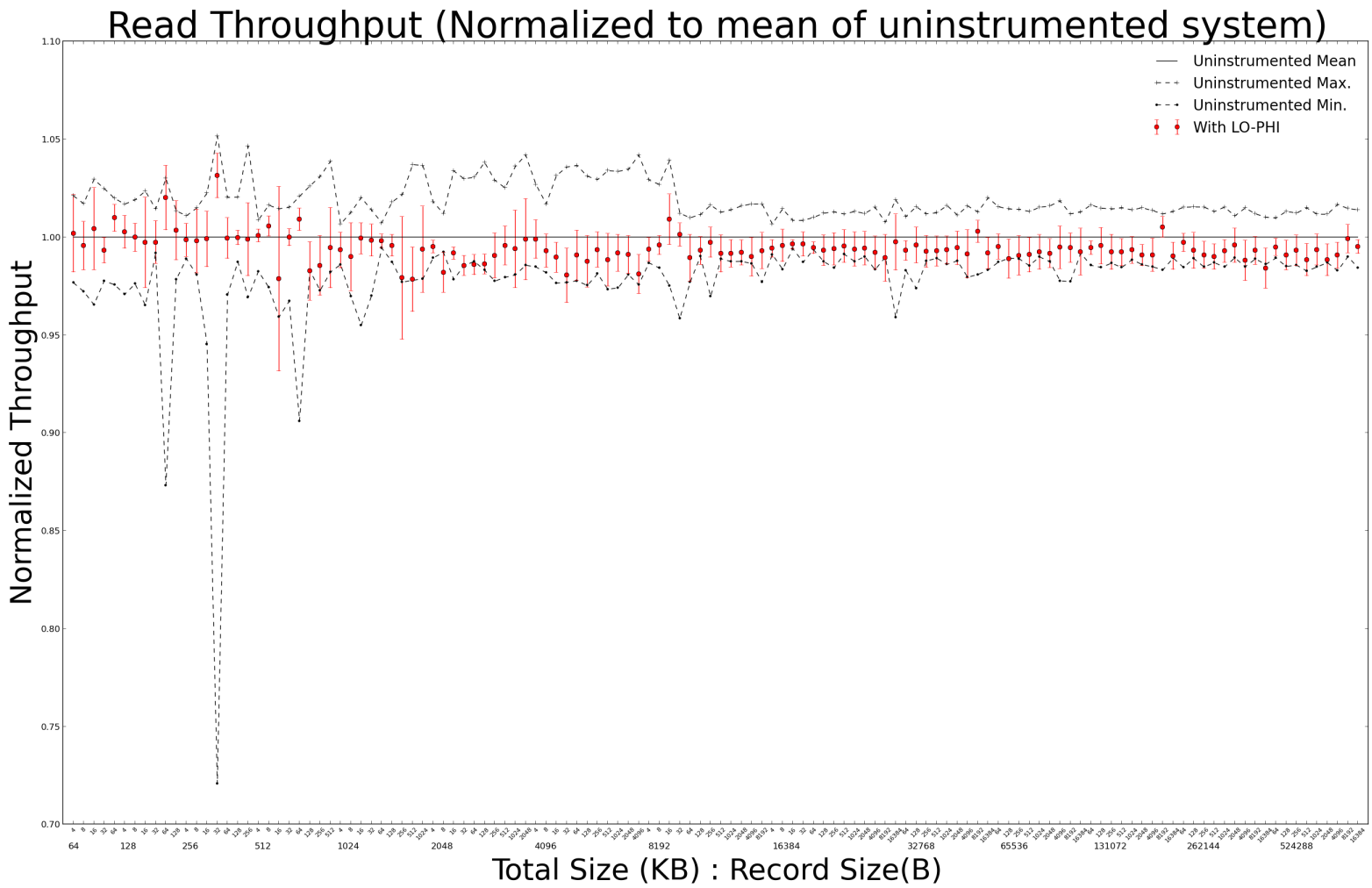
ML507

Disk Instrumentation

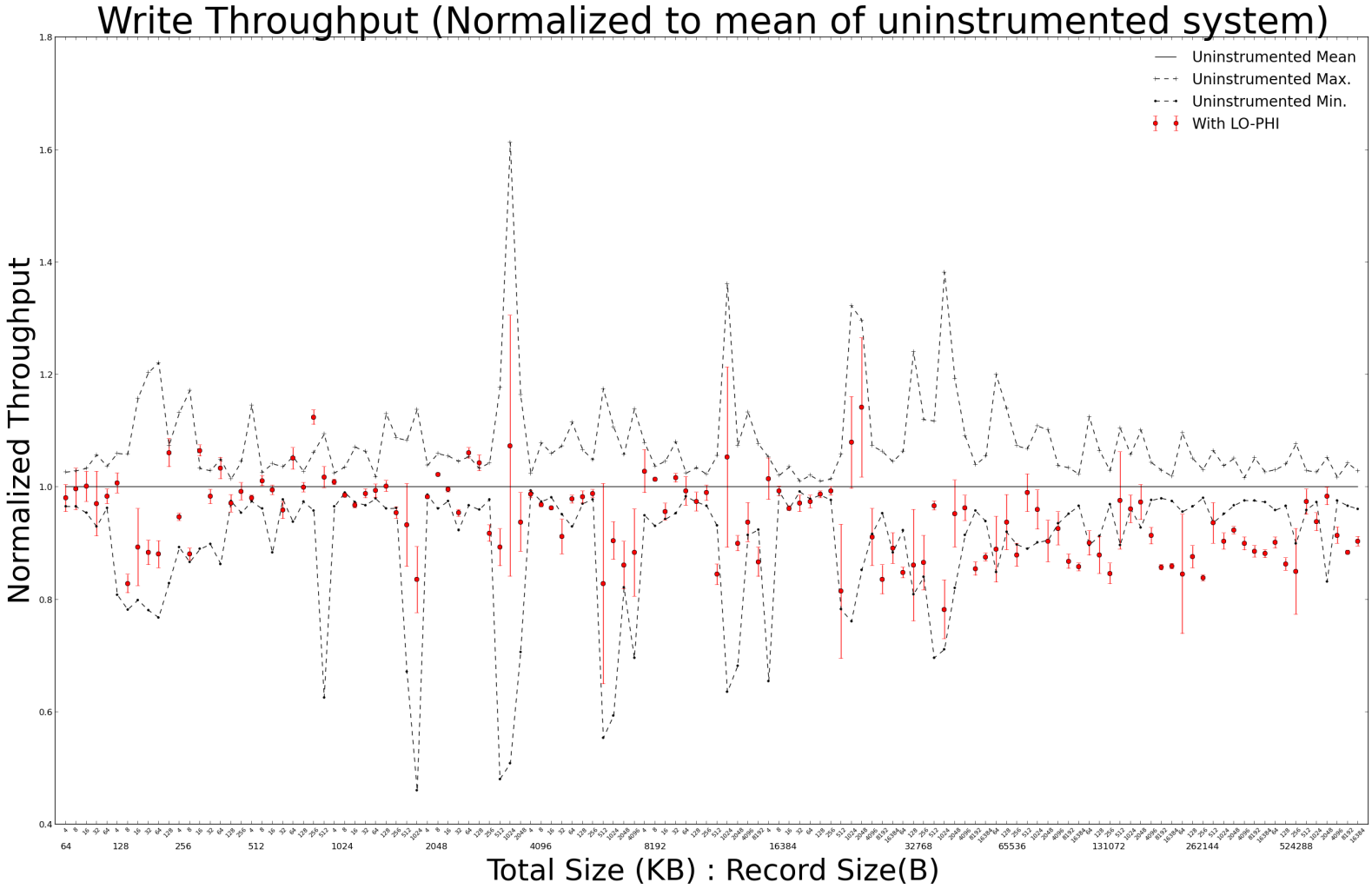
- **Virtual Limitations**
  - Artifacts
    - Same as QEMU
  - Requires modifications to QEMU source

- **Physical Limitations**
  - Artifacts
    - May sometimes need to throttle SATA to ensure full capture
  - Packet loss
    - UDP is a best-effort protocol

Disk Instrumentation: Physical



Disk Instrumentation: Physical



Semantic Reconstruction

1. **Start with a forensic copy of the instrumented disk**

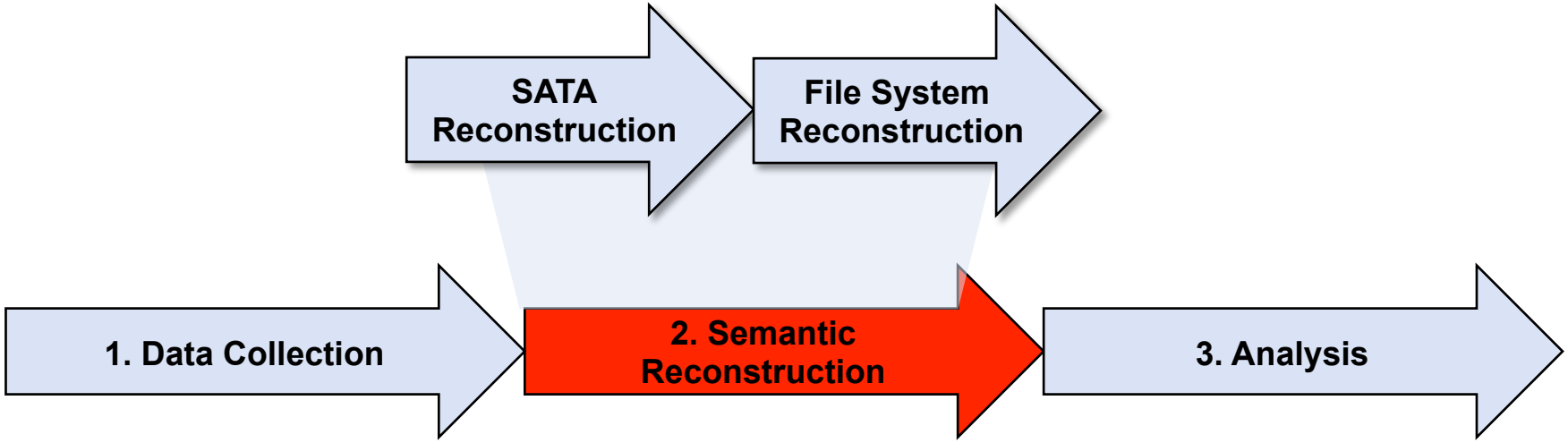
2. **Identify the file system on the disk**
   - E.g. magic numbers, expert knowledge

3. **Obtain stream of accesses to the instrumented disk in a common format**
   - E.g. (Logical Block Address, Data, Operation)

4. **Utilize forensic tools to identify subsequent file system operation**

SATA Reconstruction

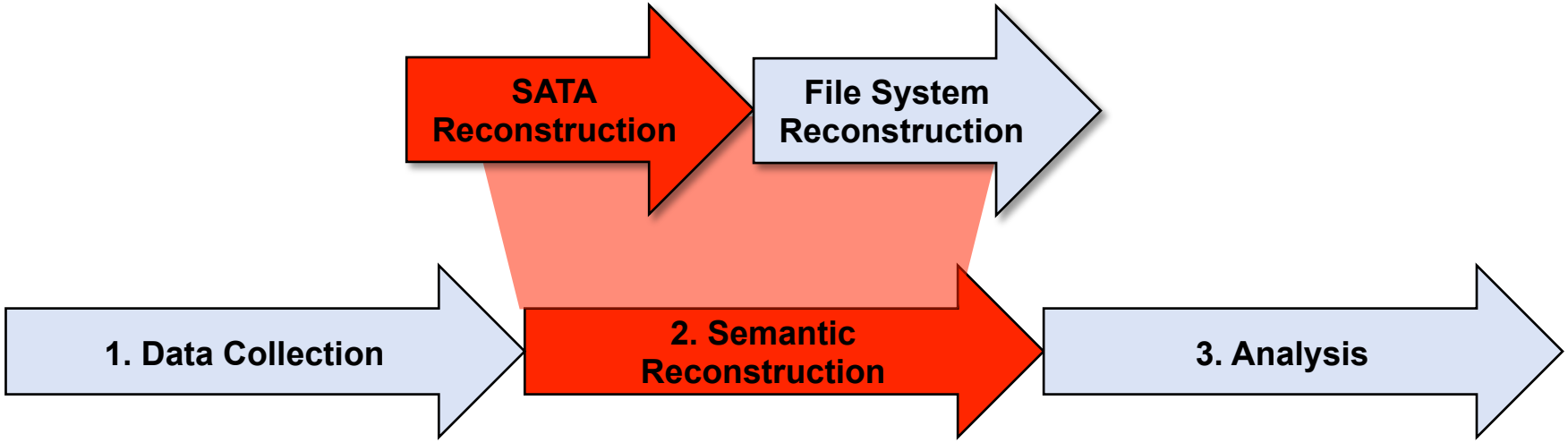
- **Multiple layers of abstraction that we must bridge**
  - **Analog Signal → Raw bits**
  - **Raw bits → SATA Frames**
  - **SATA Frames → Sector manipulation**
  - **Sector manipulation → File System Manipulation**



SATA Reconstruction

- **Multiple layers of abstraction that we must bridge**
  - ✓ **Analog Signal → Raw bits**
  - ✓ **Raw bits → SATA Frames**

  } **Xilinx ML507**
  - – **SATA Frames → Sector manipulation**
  - – **Sector manipulation → File System Manipulation**



SATA Reconstruction
A Brief Primer on SATA (1)

- **Serial ATA – bus interface that replaces older IDE/ATA standards**
- **SATA uses frames (FIS) to communicate between host and device**

HIGH SPEED SERIALIZED AT ATTACHMENT
Serial ATA International Organization

Type field value	Description
27h	Register FIS – Host to Device
34h	Register FIS – Device to Host
39h	DMA Activate FIS – Device to Host
41h	DMA Setup FIS – Bi-directional
46h	Data FIS – Bi-directional
58h	BIST Activate FIS – Bi-directional
5Fh	PIO Setup FIS – Device to Host
A1h	Set Device Bits FIS – Device to Host
A6h	Reserved for future Serial ATA definition
B8h	Reserved for future Serial ATA definition
BFh	Reserved for future Serial ATA definition
C7h	Vendor specific
D4h	Vendor specific
D9h	Reserved for future Serial ATA definition

10.3.4 Register - Host to Device

0	Features(7:0)								Command				C	R	R	R	PM Port				FIS Type (27h)			
1	Device								LBA(23:16)				LBA(15:8)								LBA(7:0)			
2	Features(15:8)								LBA(47:40)				LBA(39:32)								LBA(31:24)			
3	Control								ICC								Count(15:8)				Count(7:0)			
									7	6	5	4	3	2	1	0								
4	Reserved (0)								Reserved (0)								Reserved (0)				Reserved (0)			

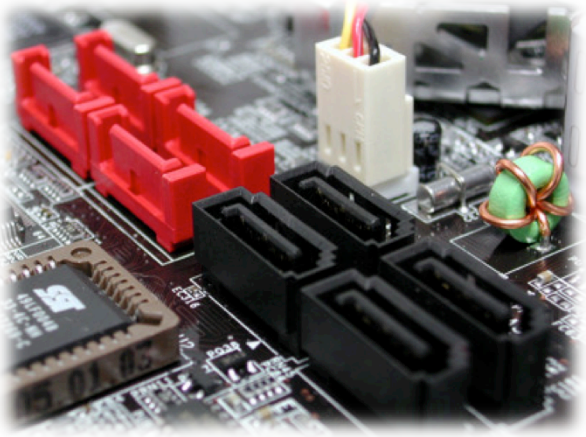
Figure 194 – Register - Host to Device FIS layout

FIS – Frame Information Structure

SATA Reconstruction
A Brief Primer on SATA (2)

- **Multi-layer protocol (physical, link, transport, command)**
  - Reconstruction focuses on the command layer

- **Read SATA standard**
  - Appendix B is useful!



SATA Reconstruction
A Brief Primer on SATA (3)

HIGH SPEED SERIALIZED AT ATTACHMENT
Serial ATA International Organization

Type field value	Description
27h	Register FIS – Host to Device
34h	Register FIS – Device to Host
39h	DMA Activate FIS – Device to Host
41h	DMA Setup FIS – Bi-directional
46h	Data FIS – Bi-directional
58h	BIST Activate FIS – Bi-directional
5Fh	PIO Setup FIS – Device to Host
A1h	Set Device Bits FIS – Device to Host

- **Register FIS Host to Device**
  - Marks the beginning of SATA transaction
  - Contains the logical block address (LBA) and operation information (read or write)
- **Register FIS Device to Host**
  - Often marks completion of SATA transaction
  - Also used in software reset protocol, device diagnostic, etc.

SATA Reconstruction
A Brief Primer on SATA (4)

HIGH SPEED SERIALIZED AT ATTACHMENT
Serial ATA International Organization

Type field value	Description
27h	Register FIS – Host to Device
34h	Register FIS – Device to Host
39h	DMA Activate FIS – Device to Host
41h	DMA Setup FIS – Bi-directional
46h	Data FIS – Bi-directional
58h	BIST Activate FIS – Bi-directional
5Fh	PIO Setup FIS – Device to Host
A1h	Set Device Bits FIS – Device to Host

- **DMA Activate**
  - Device declares that it is ready to receive DMA data (for a write)
- **DMA Setup**
  - Precedes Data frames (for NCQ, AFAIK)

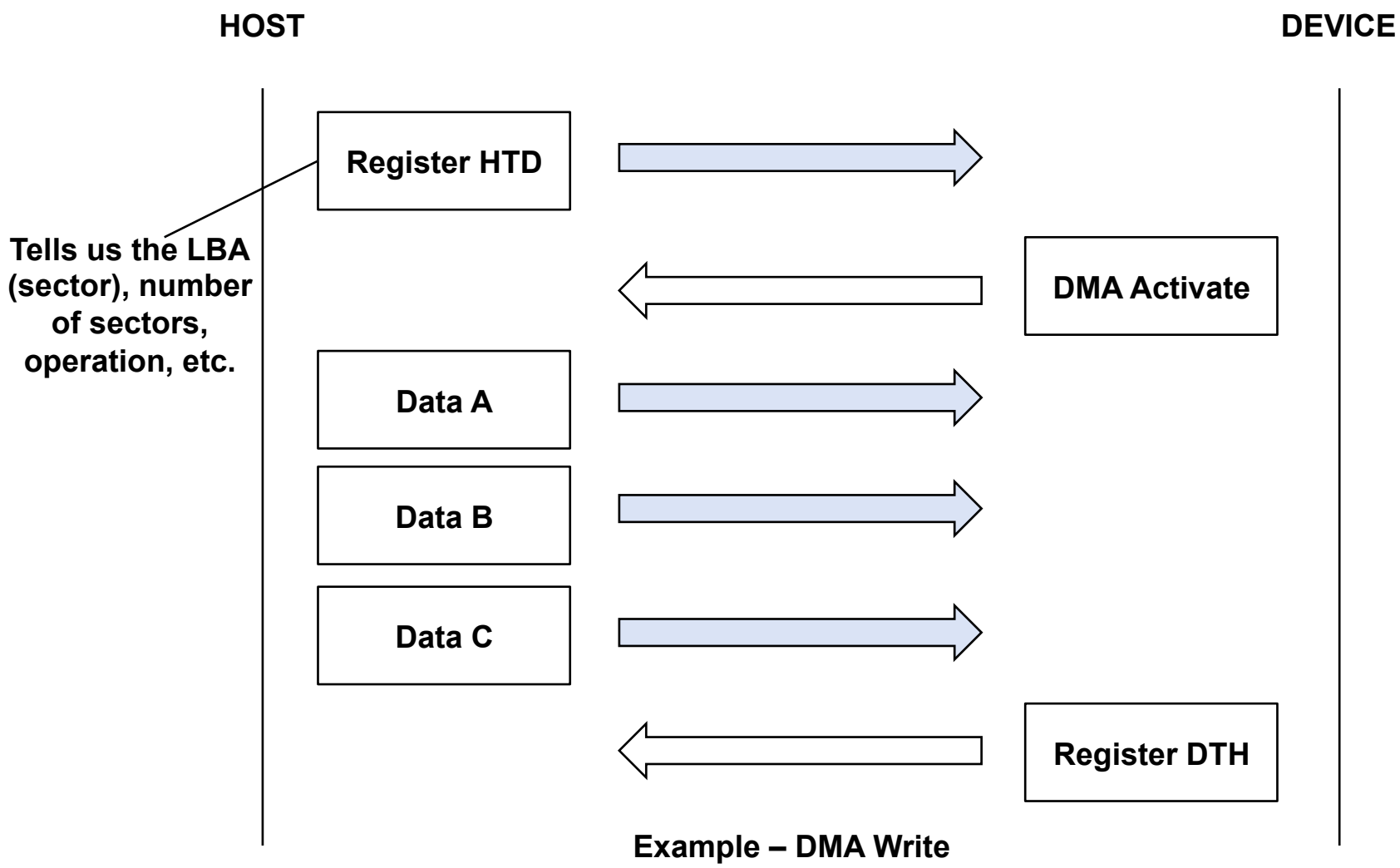
SATA Reconstruction
A Brief Primer on SATA (5)

HIGH SPEED SERIALIZED AT ATTACHMENT
Serial ATA International Organization

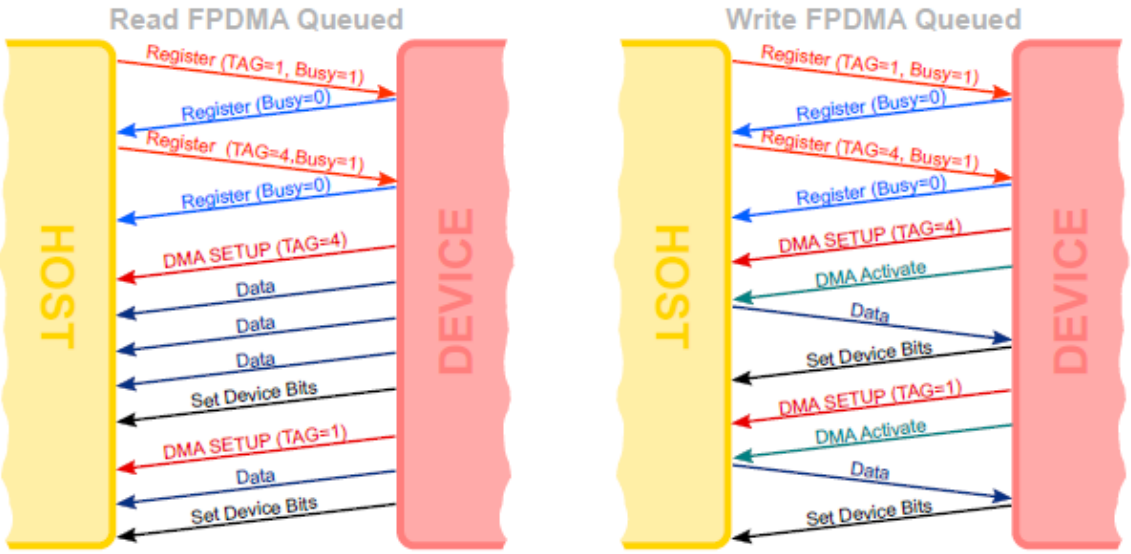
Type field value	Description
27h	Register FIS – Host to Device
34h	Register FIS – Device to Host
39h	DMA Activate FIS – Device to Host
41h	DMA Setup FIS – Bi-directional
46h	Data FIS – Bi-directional
58h	BIST Activate FIS – Bi-directional
5Fh	PIO Setup FIS – Device to Host
A1h	Set Device Bits FIS – Device to Host

- **Data – contains data!**
- **BIST (Built In Self Test)**
- **PIO (Programmed I/O)**
  - Older mode of data transfer before DMA
- **Other protocols not mentioned here**
  - Software reset, device diagnostic, device reset, packet
  - Read the SATA spec for more info

SATA Reconstruction
A Brief Primer on SATA (6)

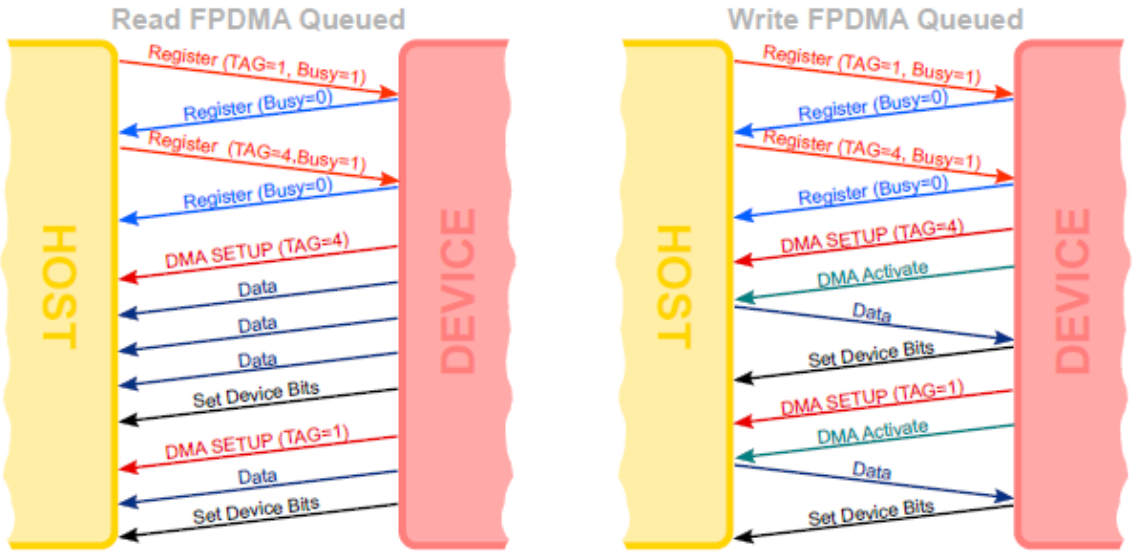


SATA Reconstruction
Native Command Queuing (1)



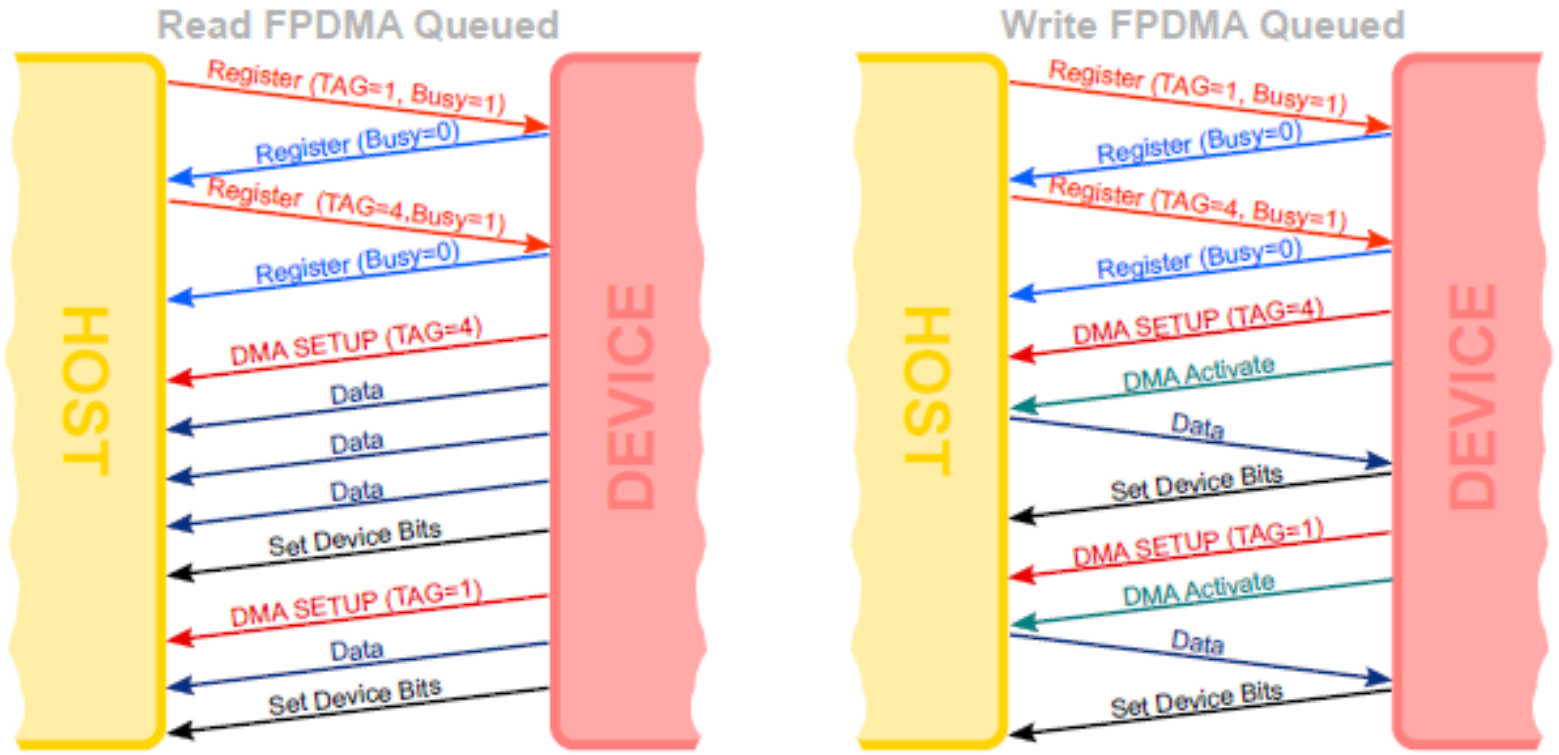
- **Native Command Queuing (NCQ) makes reconstruction harder**
- **NCQ allows for up to 32 separate, concurrent, asynchronous disk transactions**
  - Many SATA devices implement NCQ
- **NCQ identifies transactions by 5-bit TAG field (0-31)**

SATA Reconstruction
Native Command Queuing (2)



- **Not all NCQ frames are tagged (e.g. DATA), so we perform reconstruction to correctly de-interleave transactions**
- **State machine to track status of each transaction (including error conditions)**
- **Very tricky in practice – often differences between the official documentation and actual disk manufacturer practice**

SATA Reconstruction
Native Command Queuing (3)



Example

SATA Reconstruction

- **Wrote a Python module to handle all of these transactions**
  - Consumes raw SATA frames
  - Supports all of the existing SATA versions
  - Outputs stream of logical sector operations

- **Traditional SATA analyzers are expensive and don't provide analysis-friendly interfaces**

Lecroy Catalyst Stx230 2 Port **Sata** Serial Bus Protocol **Analyzer** W/
$1,550.00 used from eBay
Lecroy Catalyst STX230 2 Port **SATA** Serial Bus Protocol **Analyzer** Includes:• Carrying Case • USB

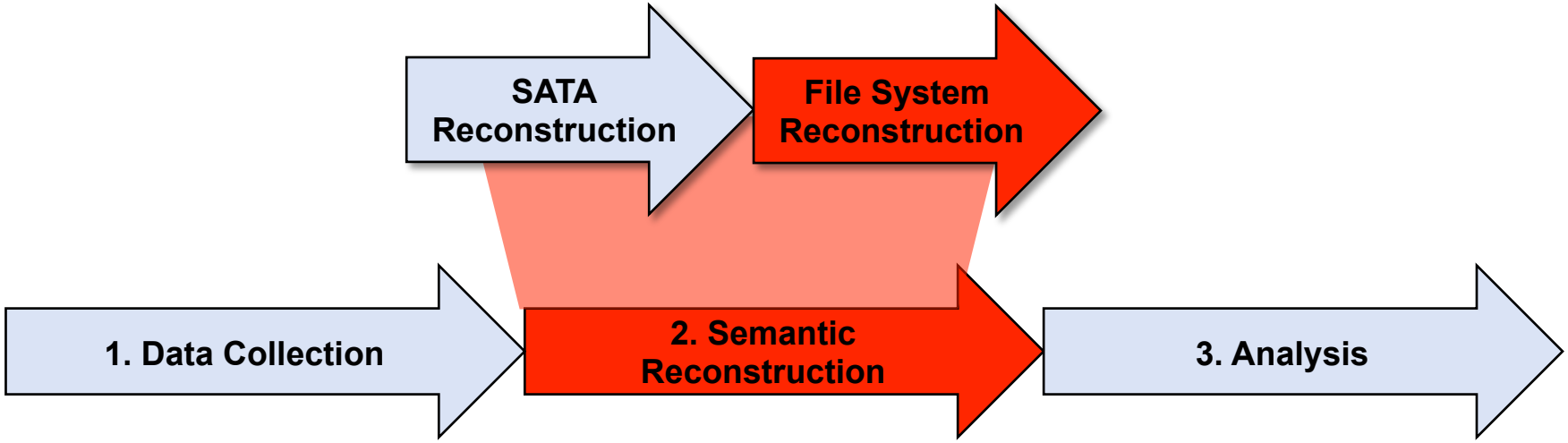
Finisar Xgig-C004 XGIG-C041 w/ 2X Xgig-B830Sa 8-Port SAS/**SATA** …
$3,995.00 used from 2 stores

Lecroy St2-31-2a **Sata** 1.5g/3g Bus
$4,000.00 refurbished from eBay
LeCroy ST2-31-2A **SATA** 1.5G/3G Bus **Analyzer** Buffer Size:1GB,1port:(Host/Device),Real Time Ev
STTAP2 ...

File System Reconstruction

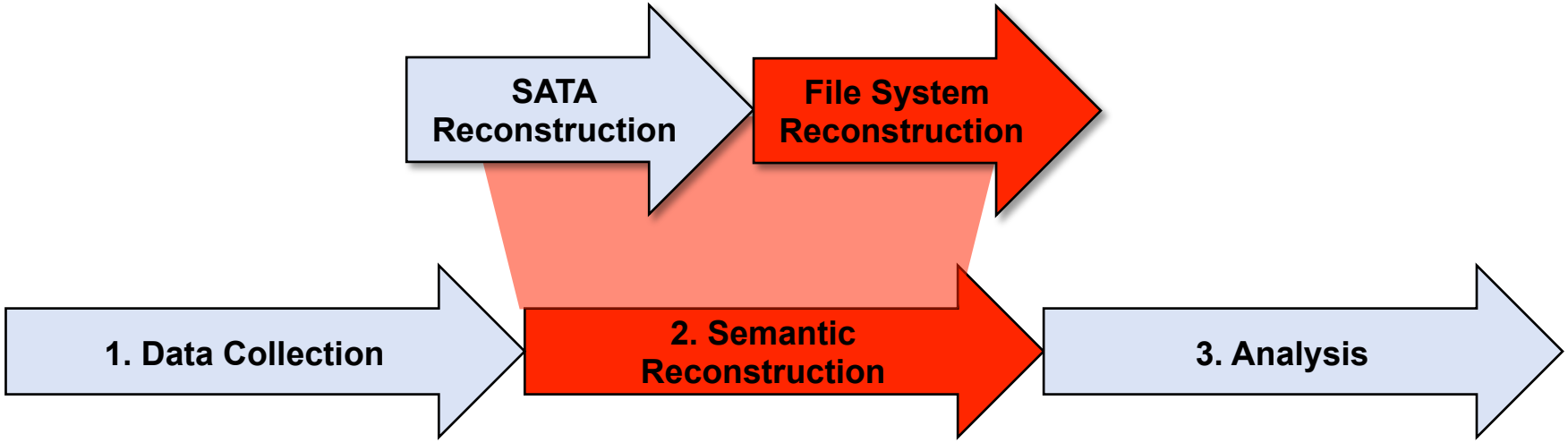
- **Multiple layers of abstraction that we must bridge**
  - ✓ **Analog Signal → Raw bits**
  - ✓ **Raw bits → SATA Frames**

  } **Xilinx ML507**
  - – **SATA Frames → Sector manipulation**
  - – **Sector manipulation → File System Manipulation**



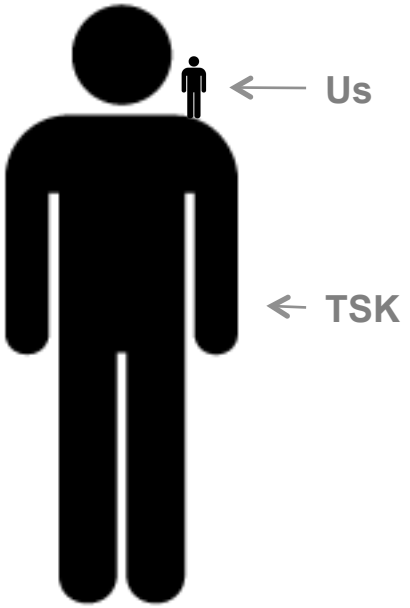
File System Reconstruction

- **Multiple layers of abstraction that we must bridge**
  - ✓ Analog Signal → Raw bits
  - ✓ Raw bits → SATA Frames
  } Xilinx ML507
  - ✓ SATA Frames → Sector manipulation SATA Reconstruction
  - – Sector manipulation → File System Manipulation

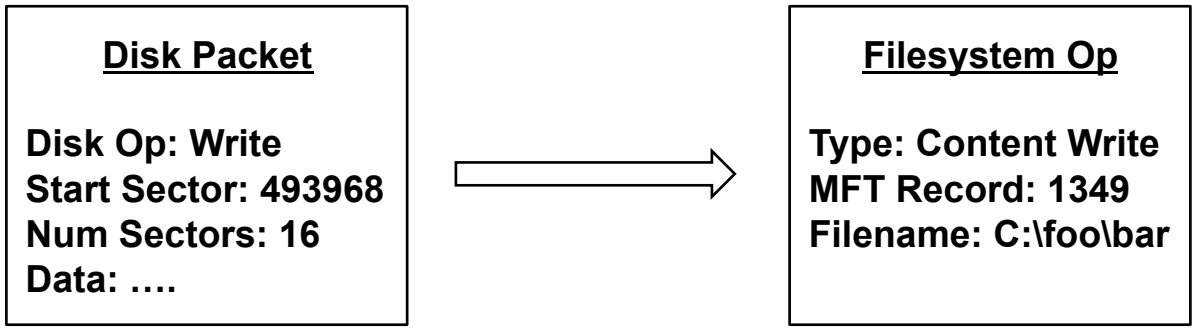


File System Reconstruction

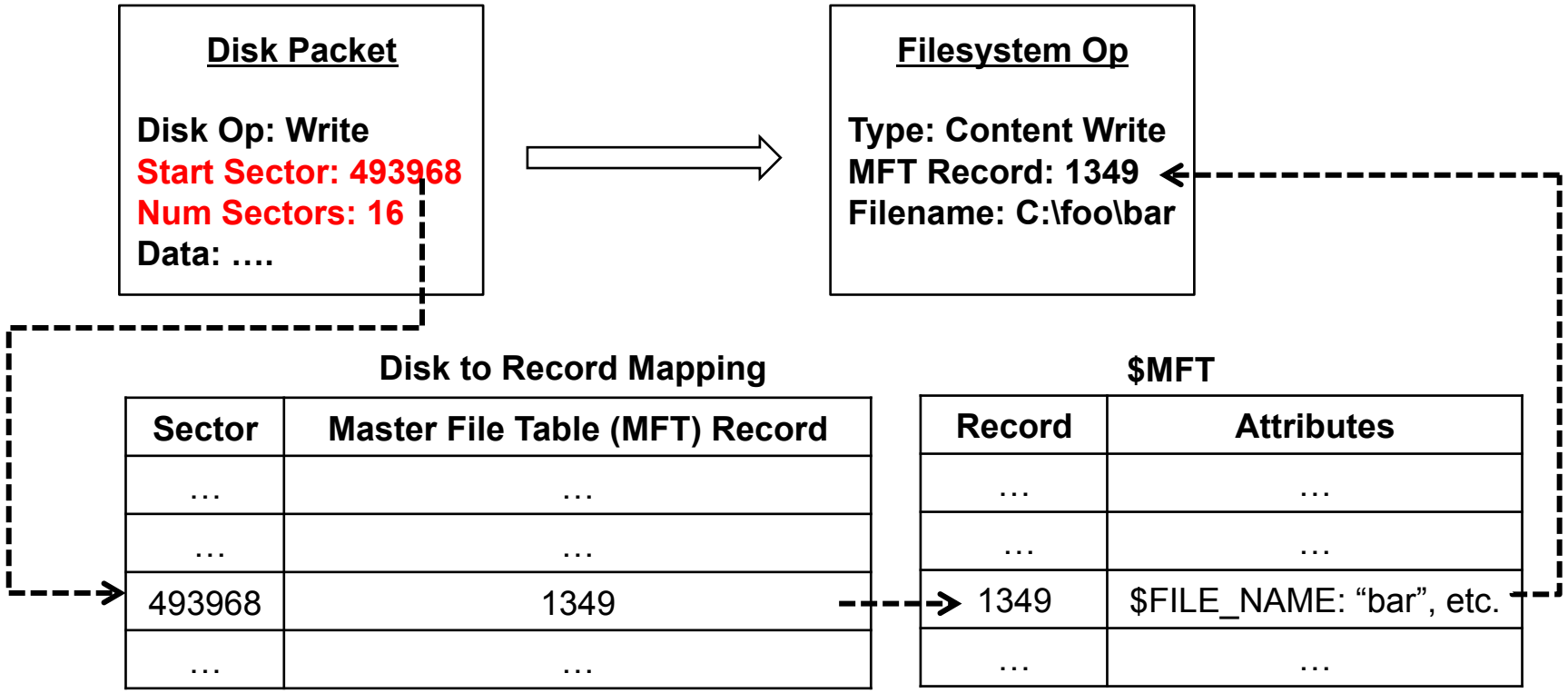
- **Sector to file mapping handled by existing forensic tools**
  - E.g. Sleuthkit
- **We use TSK for our base case and only need to track changes**
- **Read Operations**
  - Report context with associated index node (inode)
- **Write operations**
  - Update mapping if needed
  - Report context with associated inode



File System Reconstruction: NTFS



File System Reconstruction: NTFS



File System Reconstruction: NTFS

- **Problem**
  - Sleuthkit was not made with incremental updates in mind
  - Naïve solution of re-parsing the disk after updates is very slow
- **Solution**
  - Only parse minimal information required to update given file system
- **Drawback**
  - Optimizations are file system specific
    - E.g. Only monitor MFT updates in NTFS

File System Reconstruction: NTFS

- **Current Solution**
  - Utilizes PyTSK to keep a unified codebase in Python
    - Props to Joachim, Michael, et al. for the awesome work!
  - Utilizes AnalyzeMFT to parse individual MFT entries
    - Props to David Kovar, bug fixes are on their way!
- **Implementation**
  - MFT modification
    - Diff previous MFT entry with new MFT entry
    - Update internal caching structures
    - Report changes
  - Non-MFT
    - Report if sector is associated with a run of a know MFT structure
    - Otherwise report as unknown to be resolved later

File System Reconstruction

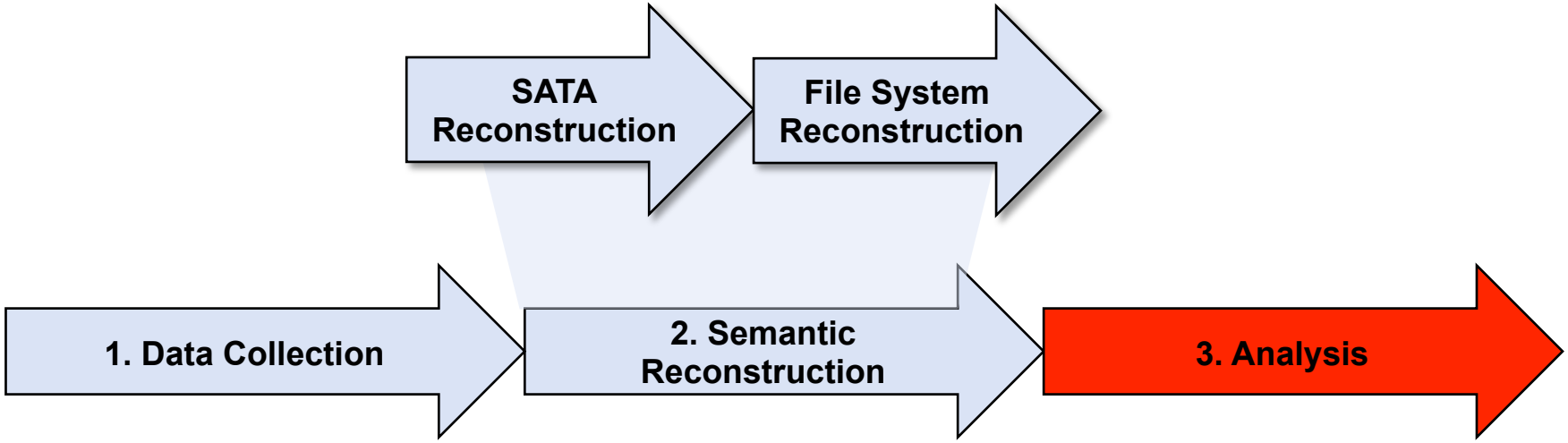
- **Currently have a stable mostly-optimized implementation for NTFS**
  - Could still reduce memory footprint
  - Want to push AnalyzeMFT-like functionality into TSK
- **Working on expanding to other file systems**
  - Need to identify all of the potential regions that update the underlying structure per file system
- **In the process of pushing the code out to the community to solicit feedback**

Analysis

- Multiple layers of abstraction that we must bridge
  - ✓ Analog Signal → Raw bits
  - ✓ Raw bits → SATA Frames

  } Xilinx ML507

  - ✓ SATA Frames → Sector manipulation SATA Reconstruction
  - ✓ Sector manipulation → File System Manipulation TSK & analyzeMFT



Analysis

- **Analysis step is application-dependent and open to the user**
- **Flexible and easy to use API**
- **Example uses:**
  - Simple filtering on specific files or disk regions (e.g. /bootmgr)
  - Detect writes to slack space
  - Feature extraction and machine learning for malware analysis

Analysis

- **We are currently using our framework to detect VM-aware malware**
  - Results and future publication pending . . .

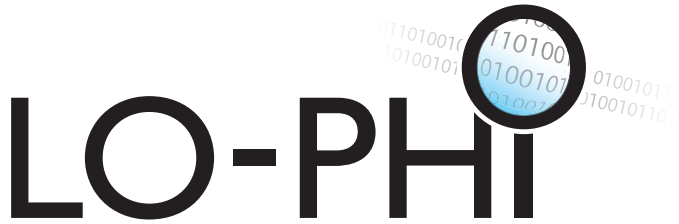
- **However, we foresee there being numerous use cases that we have not yet thought of**

Agenda

- Overview
- Motivation
- Architecture
- Live Disk Forensics
- Summary
- Future Directions

Advantages

- Less divergence from real environments
- Introspection at the hardware level (difficult to subvert from software)
- Ability to instrument proprietary, legacy, or embedded systems that can't be virtualized
- Open and flexible framework

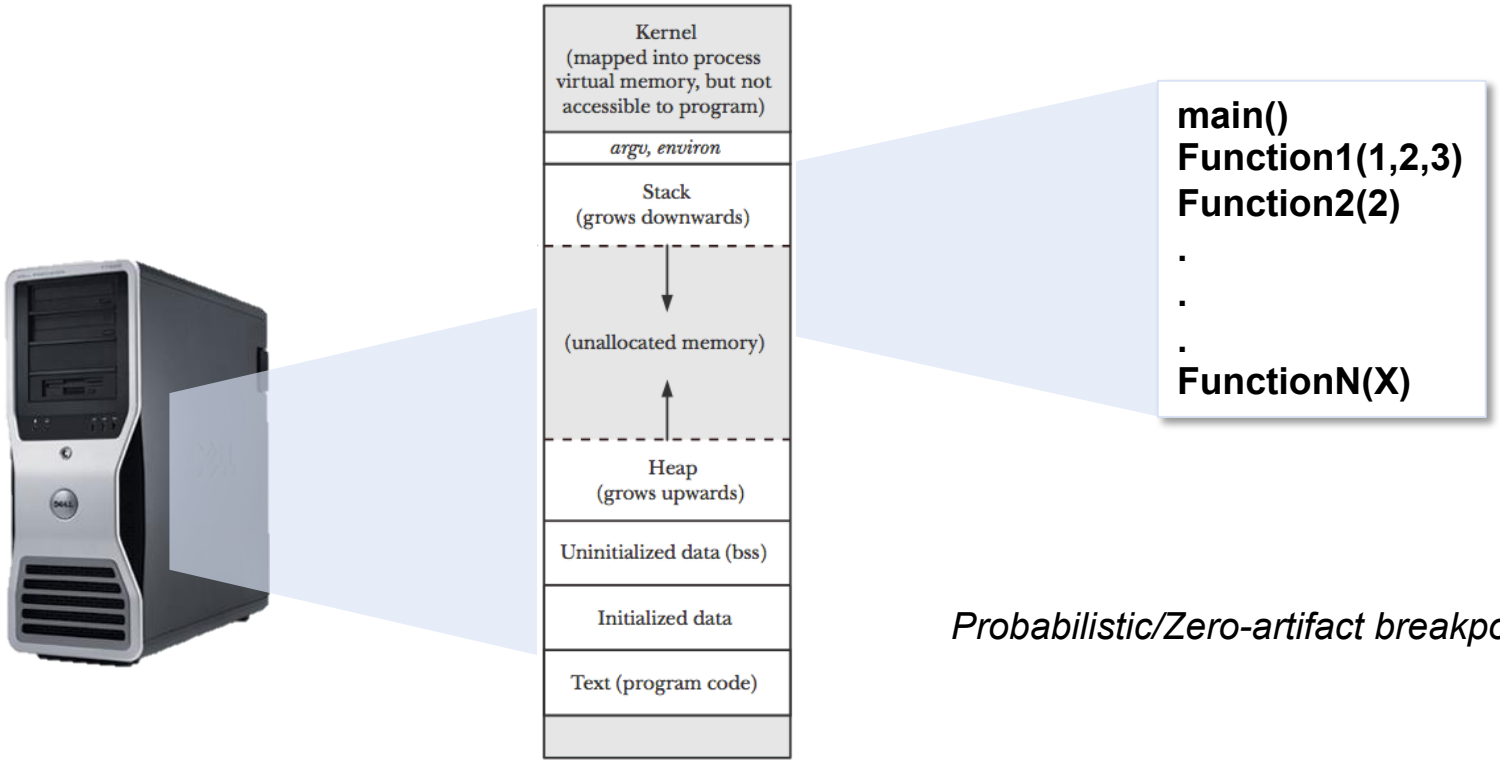


Summary

- **Developed an instrumentation suite for both physical and virtual machines**
- **Showed that this instrumentation is capable of collecting complete real-time data with minimal artifacts**
- **Adapted popular forensics tools to bridge the semantic gap in real-time on live systems**
- **Provides entire instrumentation suite so that researchers can focus on higher-level problems**

What's Next?

- **Process introspection / zero-artifact debugging**



Probabilistic/Zero-artifact breakpoints

Questions?

LOPHI@ll.mit.edu