DARPA/I2O
Transparent Computing Progra

THEIA: Tagging and Tracking of Multi-Level Host Events for Transparent Computing and Information Assurance

Mattia Fazzini
Georgia Institute of Technology

Nov 3rd, 2017

Agenda

- Project overview
- Technical discussion
  - THEIA-Panda
  - THEIA-KI
- Future work

Project Team

Project Team

PI

Wenke Lee

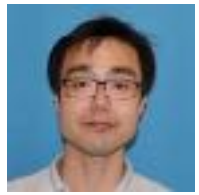
Project Team

PI	Co-PI	Co-PI	Co-PI
			
Wenke Lee	Simon Chung	Taesoo Kim	Alessandro Orso

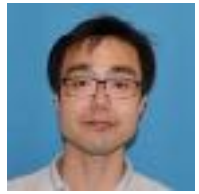
Project Team

PI	Co-PI	Co-PI	Co-PI	GTRI
				
Wenke Lee	Simon Chung	Taesoo Kim	Alessandro Orso	Trent Brunson

Project Team

PI	Co-PI	Co-PI	Co-PI	GTRI	Postdoc
					
Wenke Lee	Simon Chung	Taesoo Kim	Alessandro Orso	Trent Brunson	Sangho Lee

Project Team

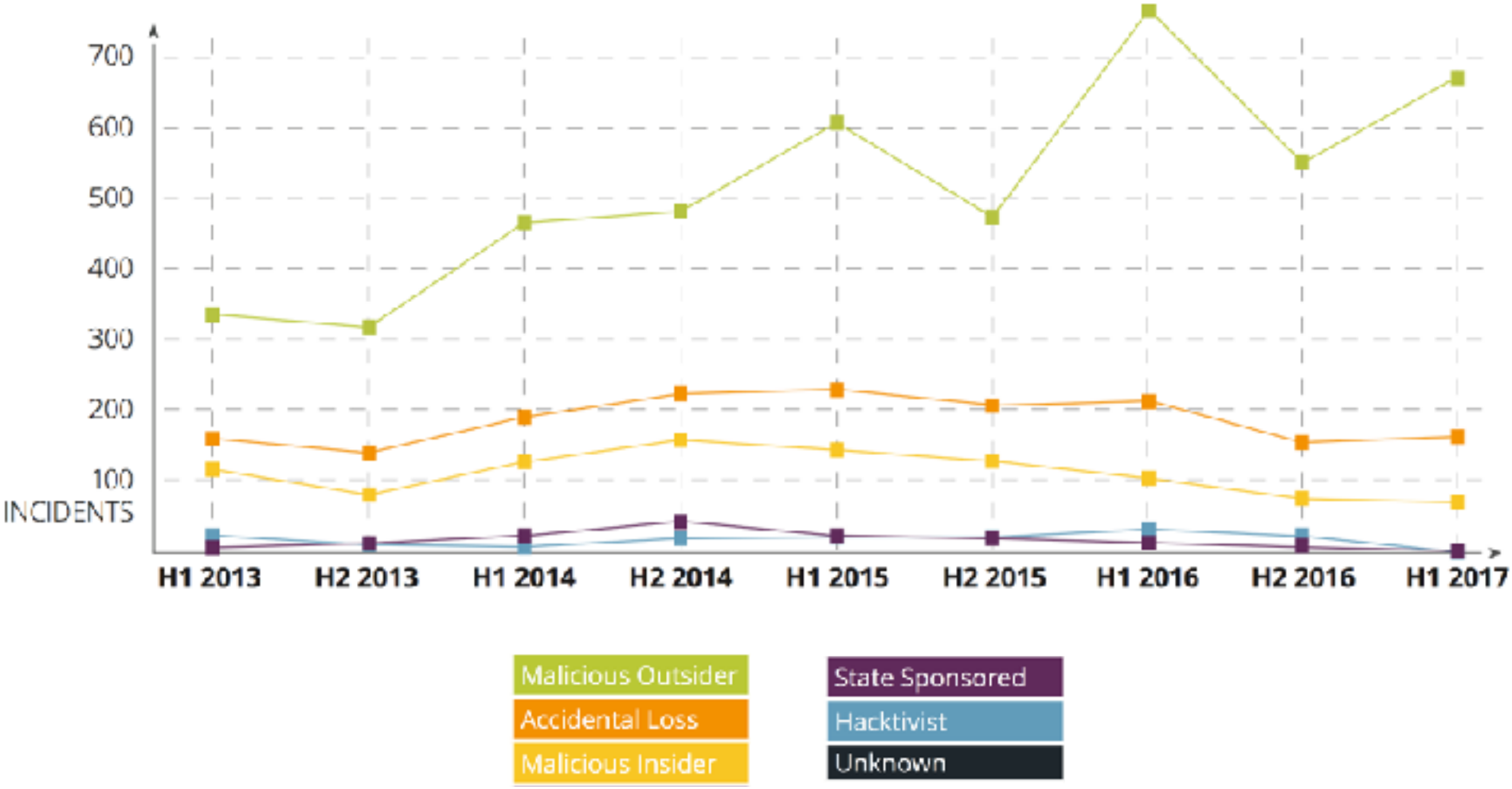
PI	Co-PI	Co-PI	Co-PI	GTRI	Postdoc
					
Wenke Lee	Simon Chung	Taesoo Kim	Alessandro Orso	Trent Brunson	Sangho Lee

Ph.D Student	Ph.D Student	Ph.D Student	Ph.D Student	Ph.D Student	Ph.D Student
					
Joey Allen	Evan Downing	Mattia Fazzini	Yang Ji	Weiren Wang	Carter Yagemann

Data Breaches

Data Breaches

Data Breaches Trend



THEIA

- **Objective:**
  - Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)
- **Efficiency:**
  - Decouple analyses from runtime through record and replay
- **Transparency:**
  - OS level
    - Establish causality relationship between system operations
  - Program level
    - Identify relations between program instructions
  - UI level
    - Capture user's intent to provide ground truth of intended behavior

THEIA

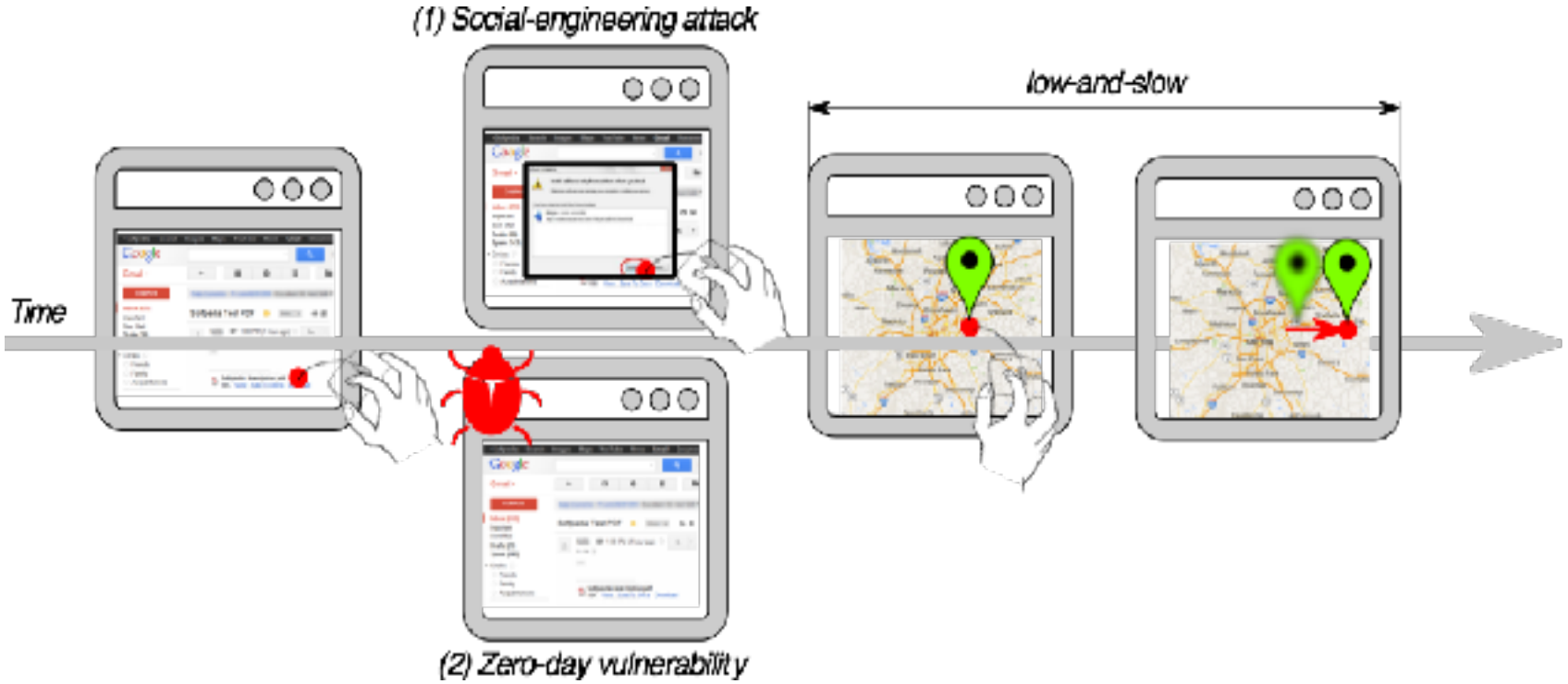
- **Objective:**
  - Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)
- **Efficiency:**
  - Decouple analyses from runtime through record and replay
- **Transparency:**
  - OS level
    - Establish causality relationship between system operations
  - Program level
    - Identify relations between program instructions
  - UI level
    - Capture user's intent to provide ground truth of intended behavior

THEIA

- **Objective:**
  - Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)
- **Efficiency:**
  - Decouple analyses from runtime through record and replay
- **Transparency:**
  - OS level
    - Establish causality relationship between system operations
  - Program level
    - Identify relations between program instructions
  - UI level
    - Capture user's intent to provide ground truth of intended behavior

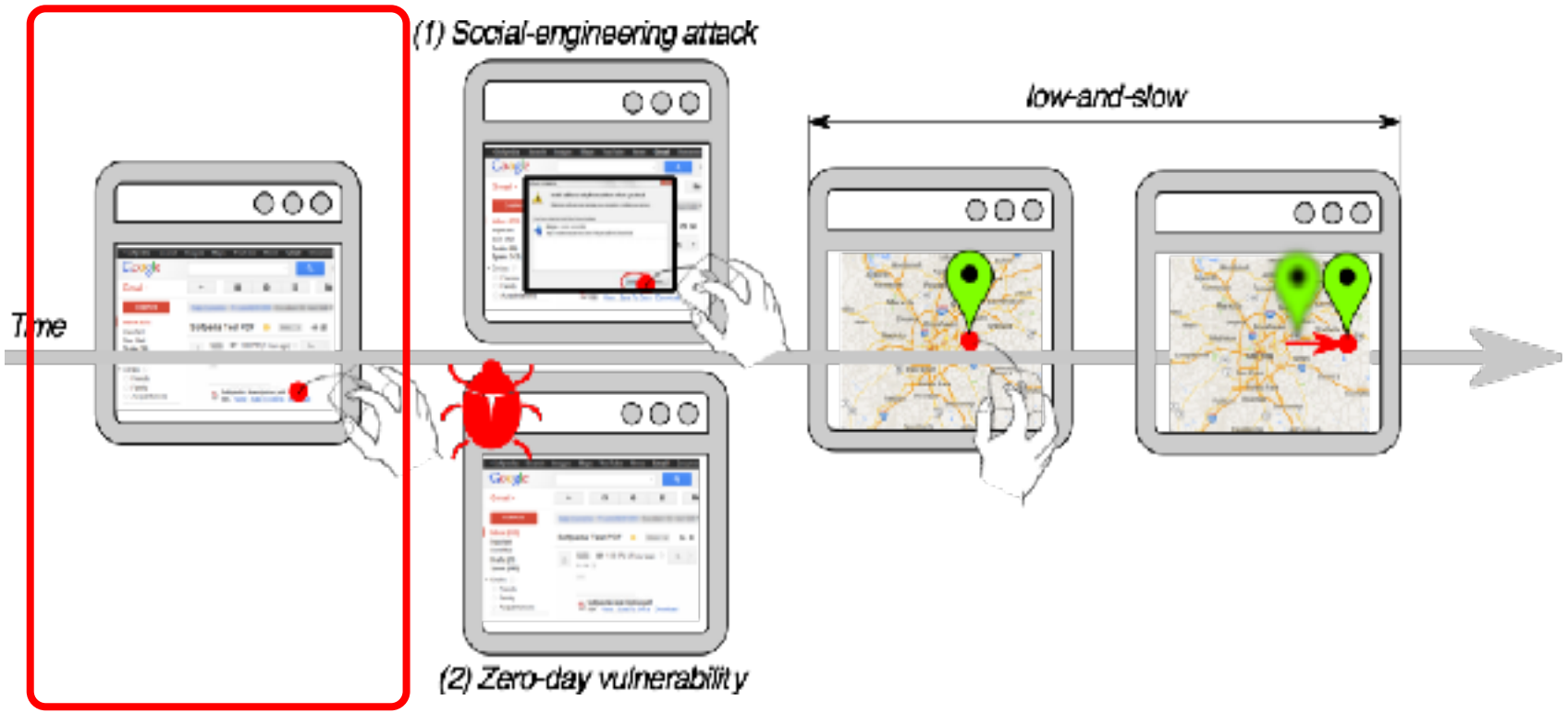
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities



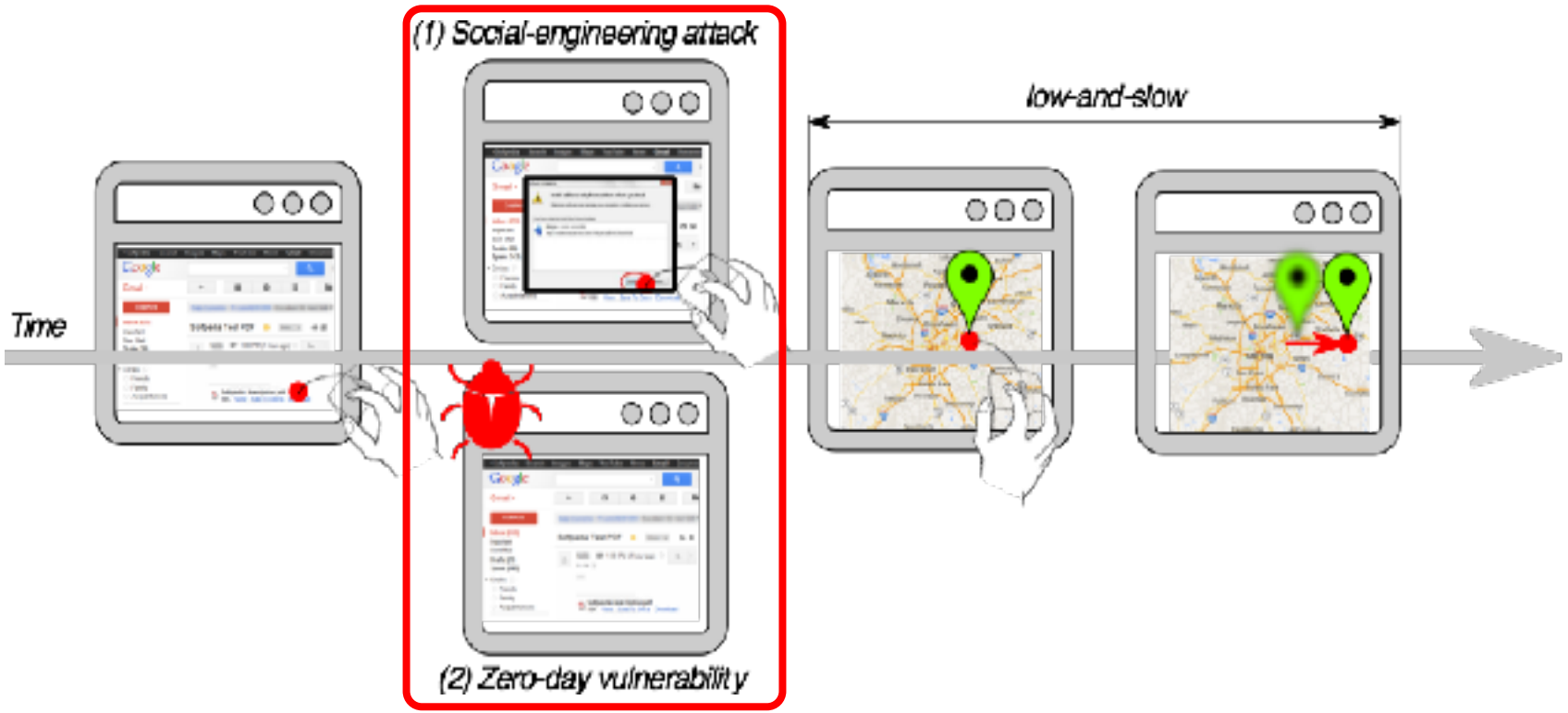
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities



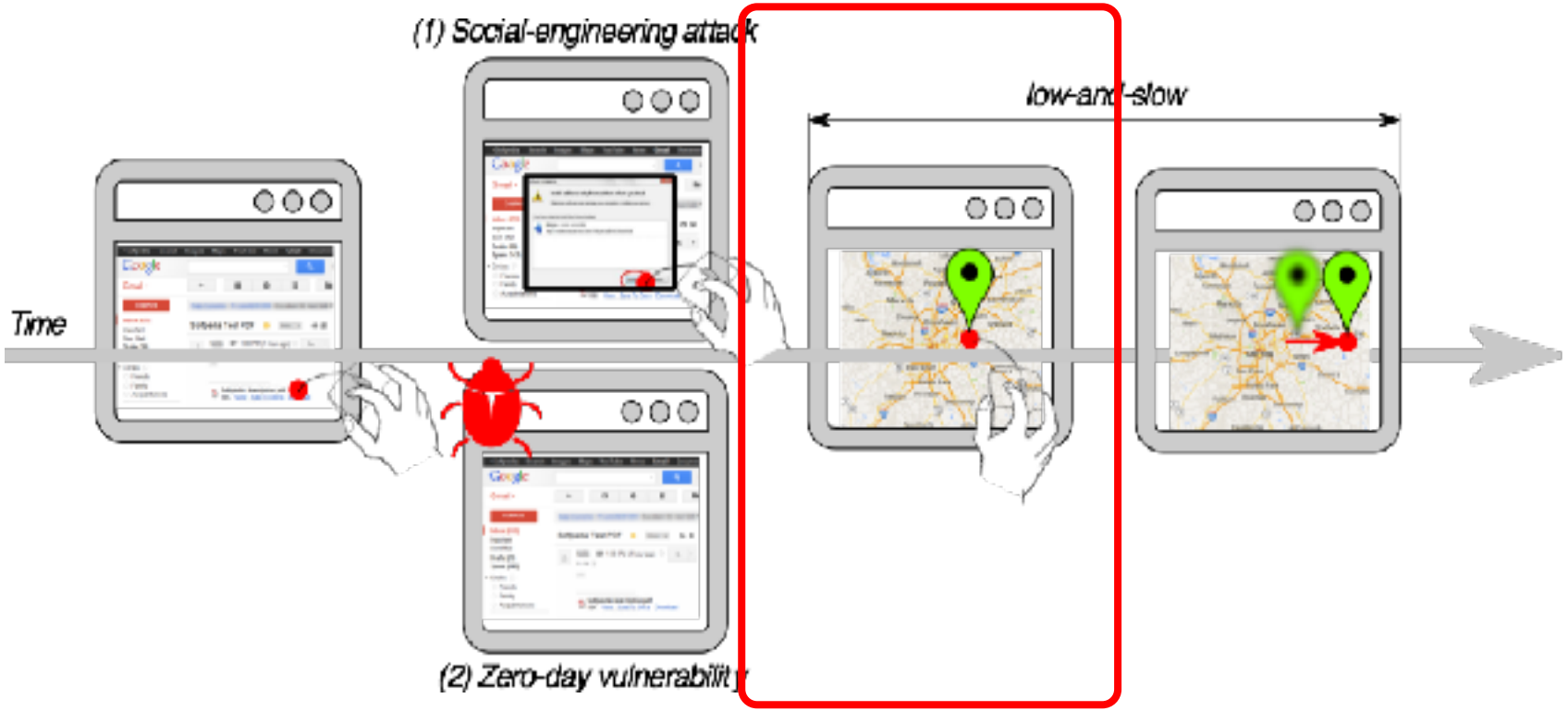
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities



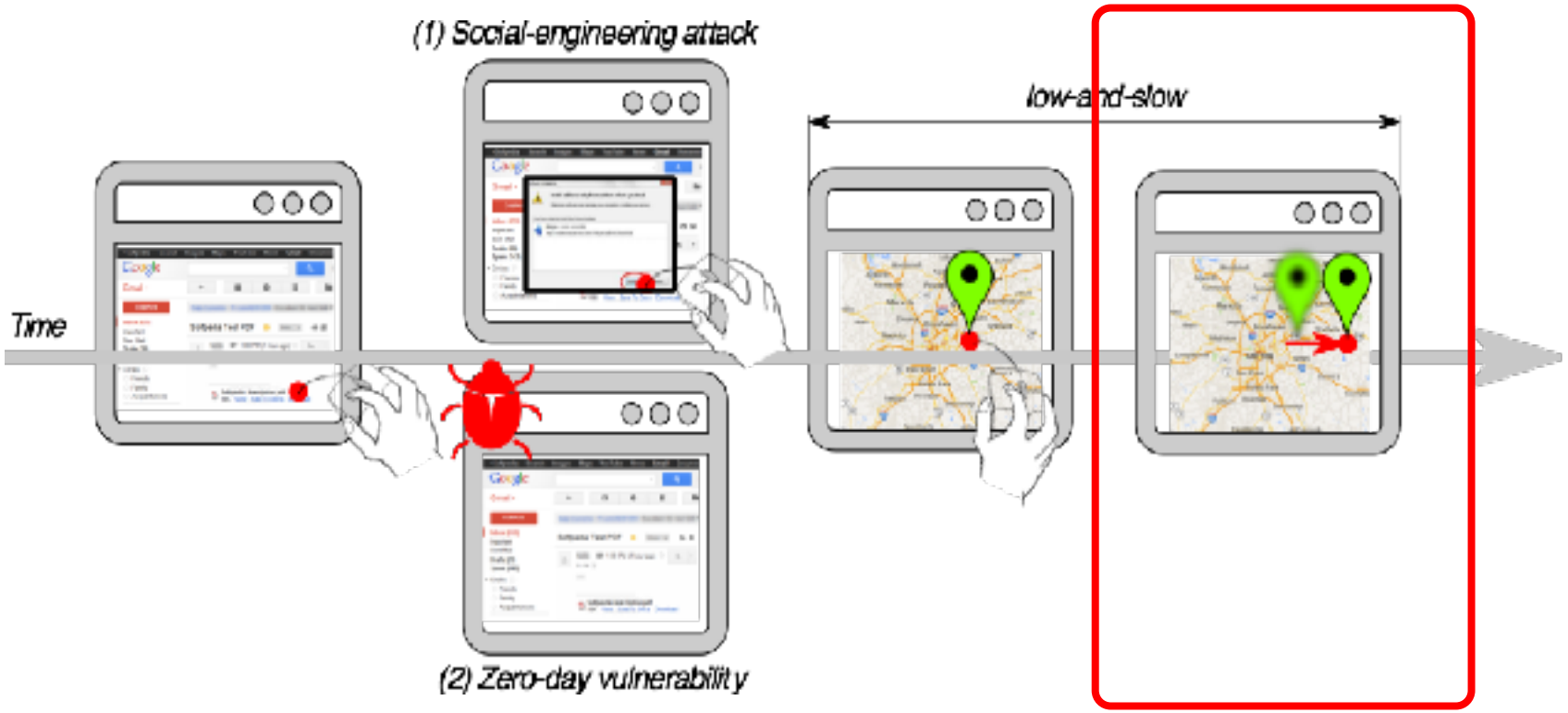
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities

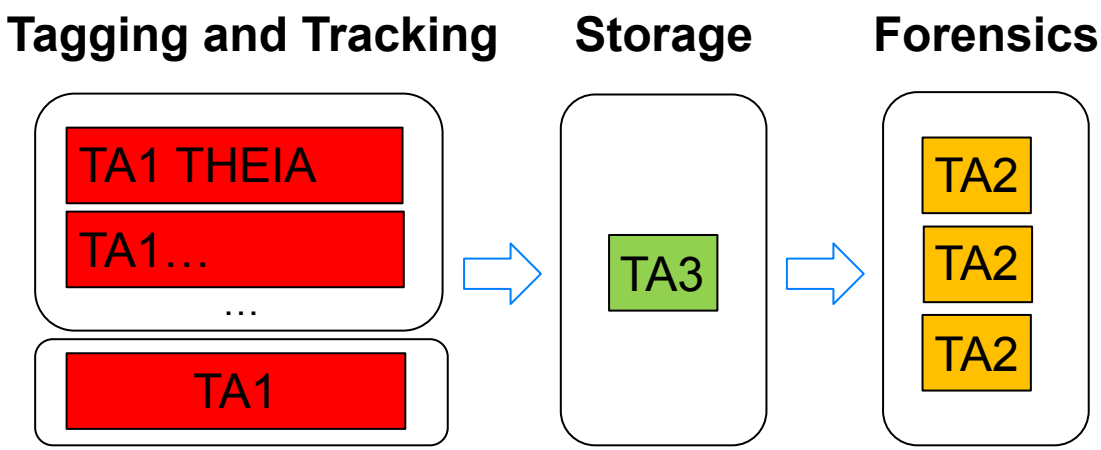


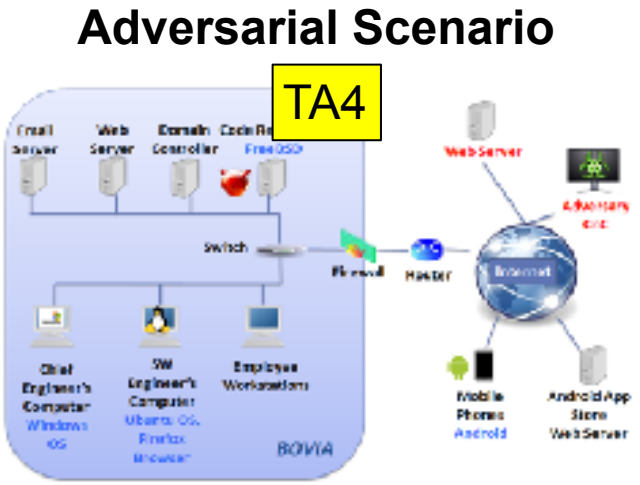
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities

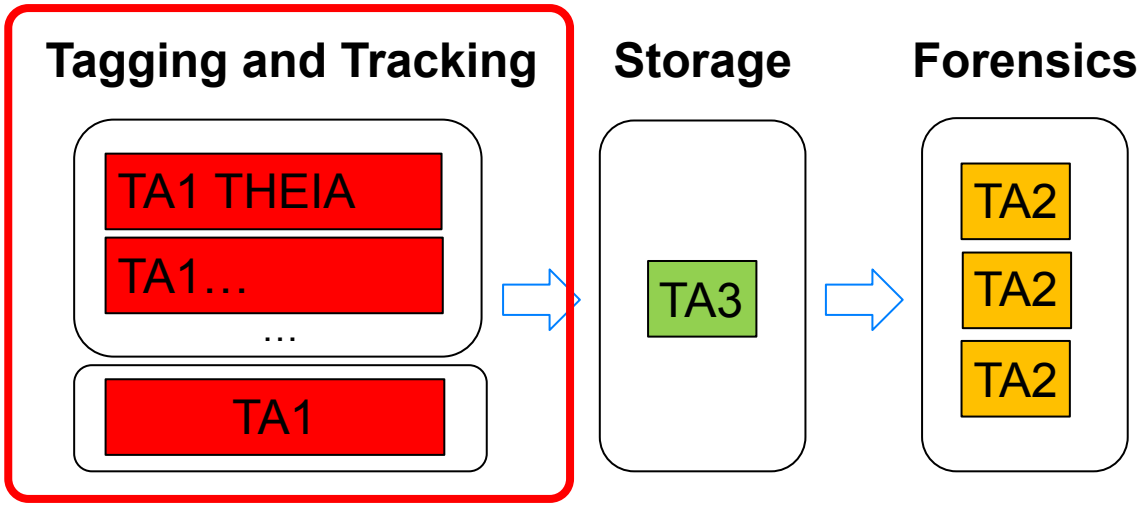


DARPA Transparent Computing



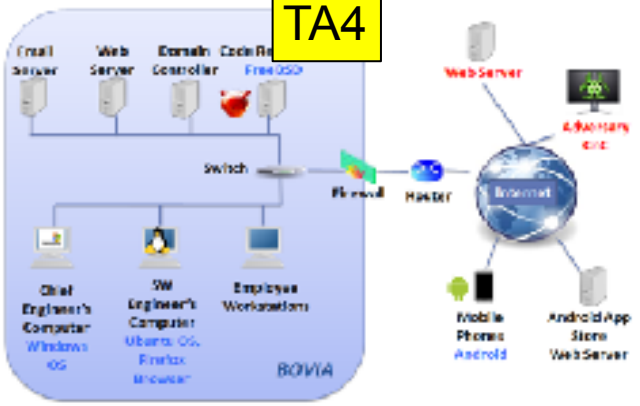


DARPA Transparent Computing



Adversarial Scenario

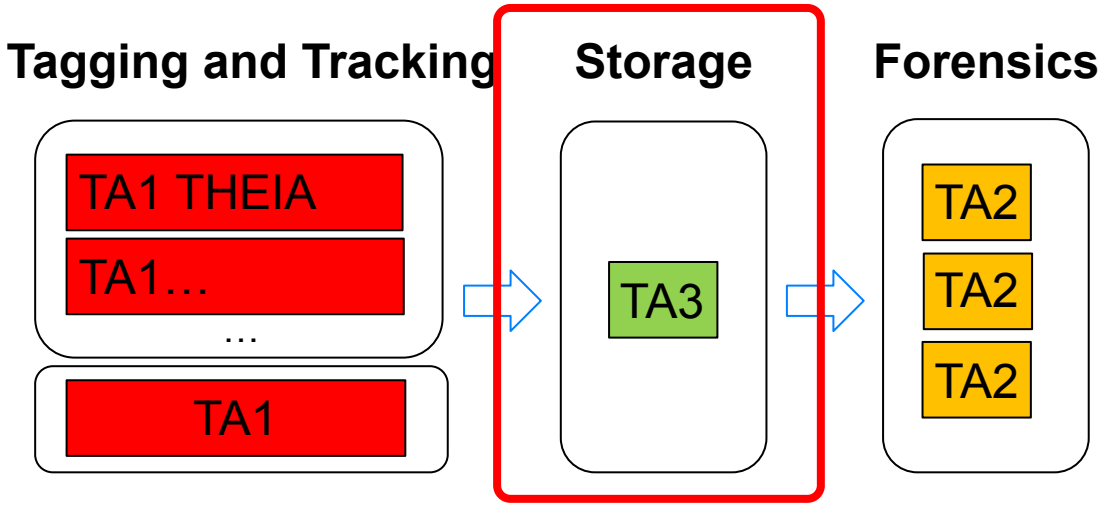
TA4



Malware

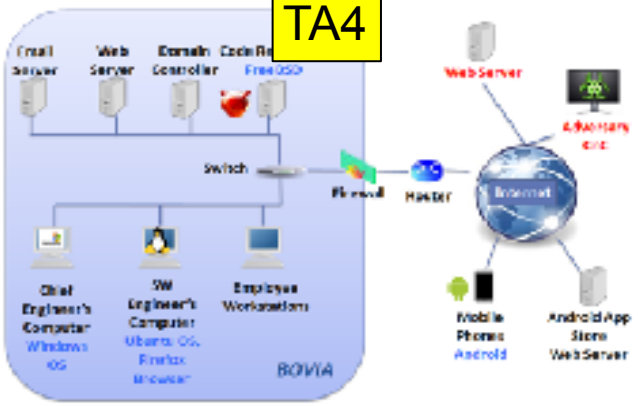
TA5

DARPA Transparent Computing



Adversarial Scenario

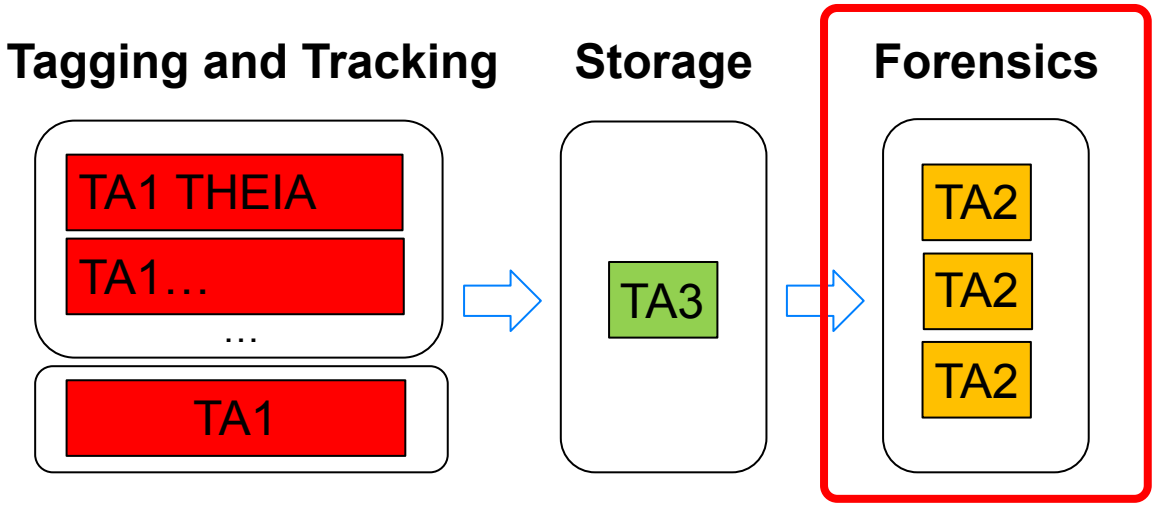
TA4



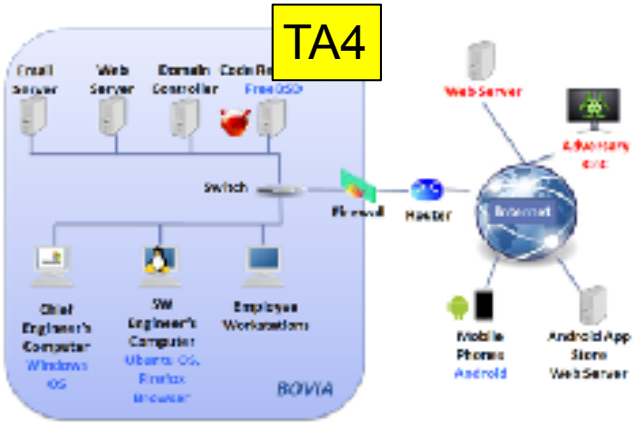
Malware

TA5

DARPA Transparent Computing

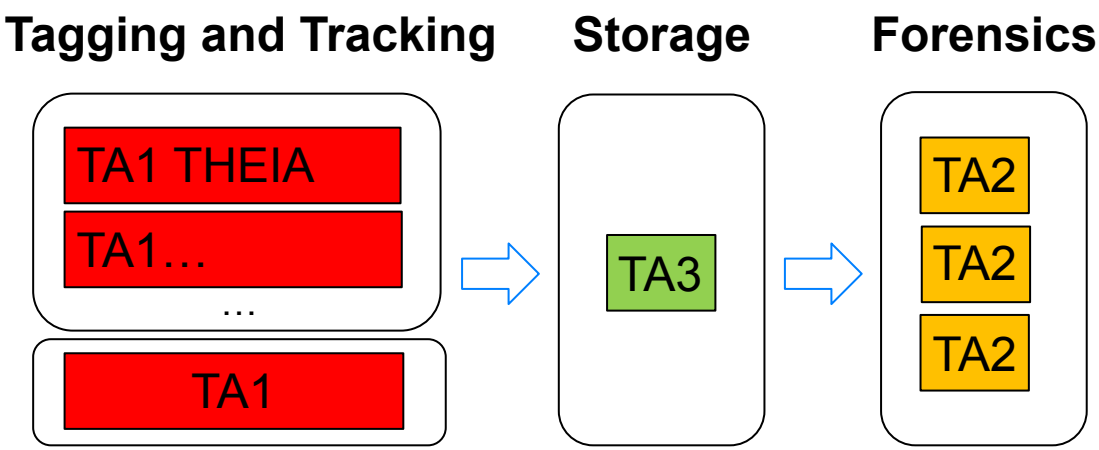


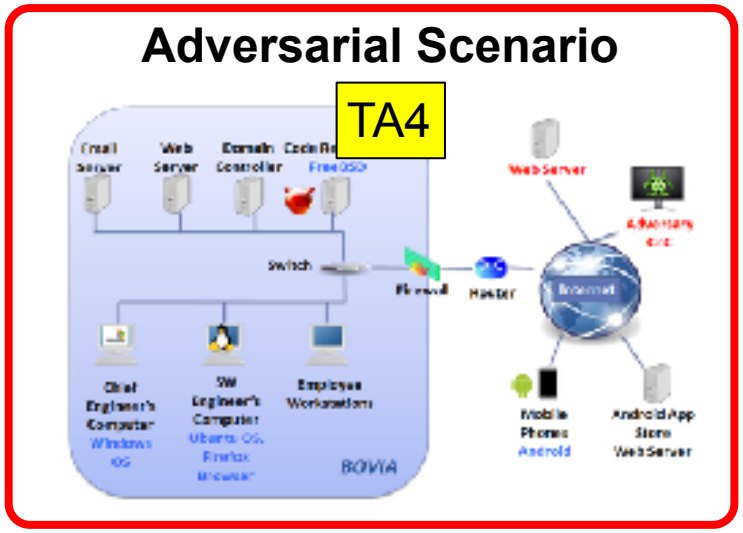
Adversarial Scenario



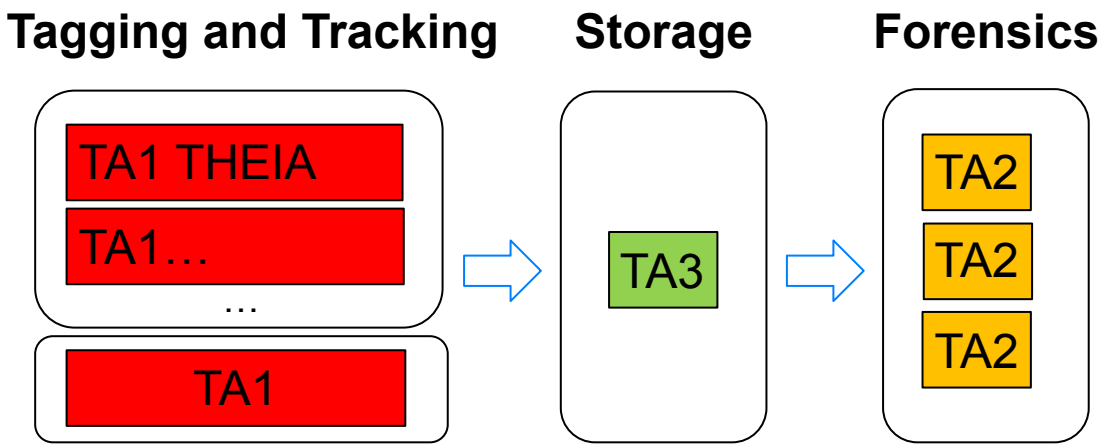
Malware

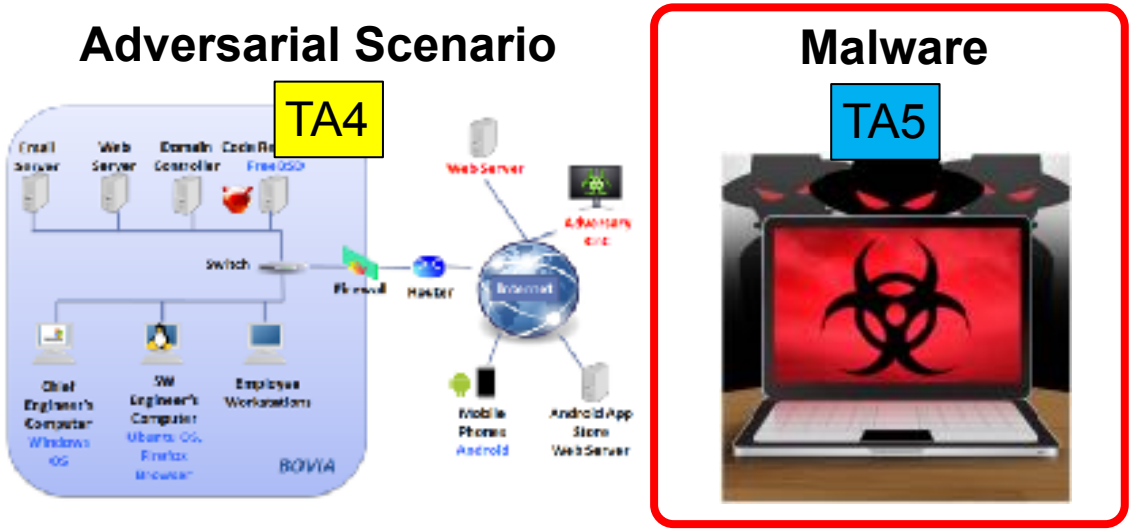
DARPA Transparent Computing



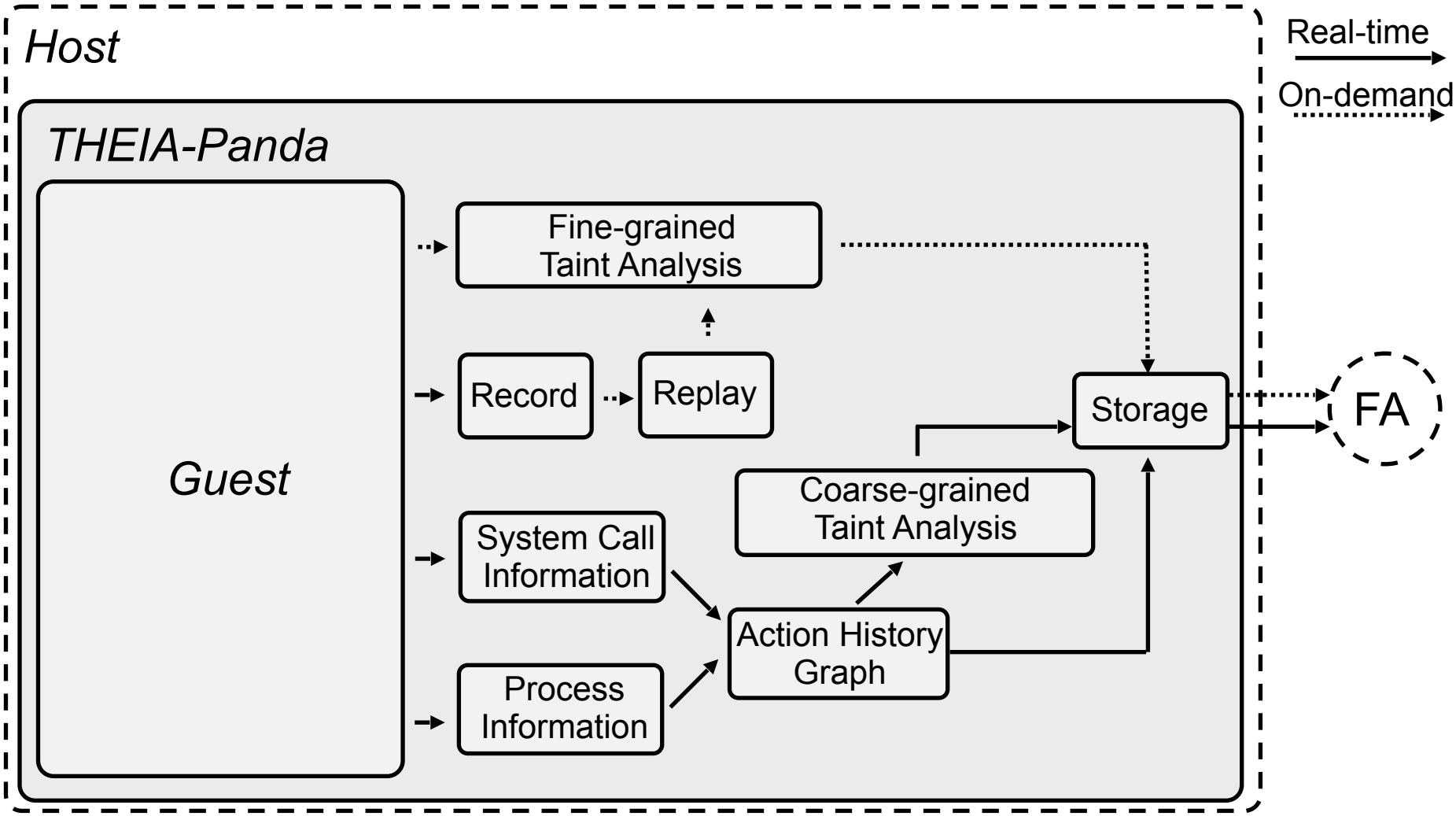


DARPA Transparent Computing

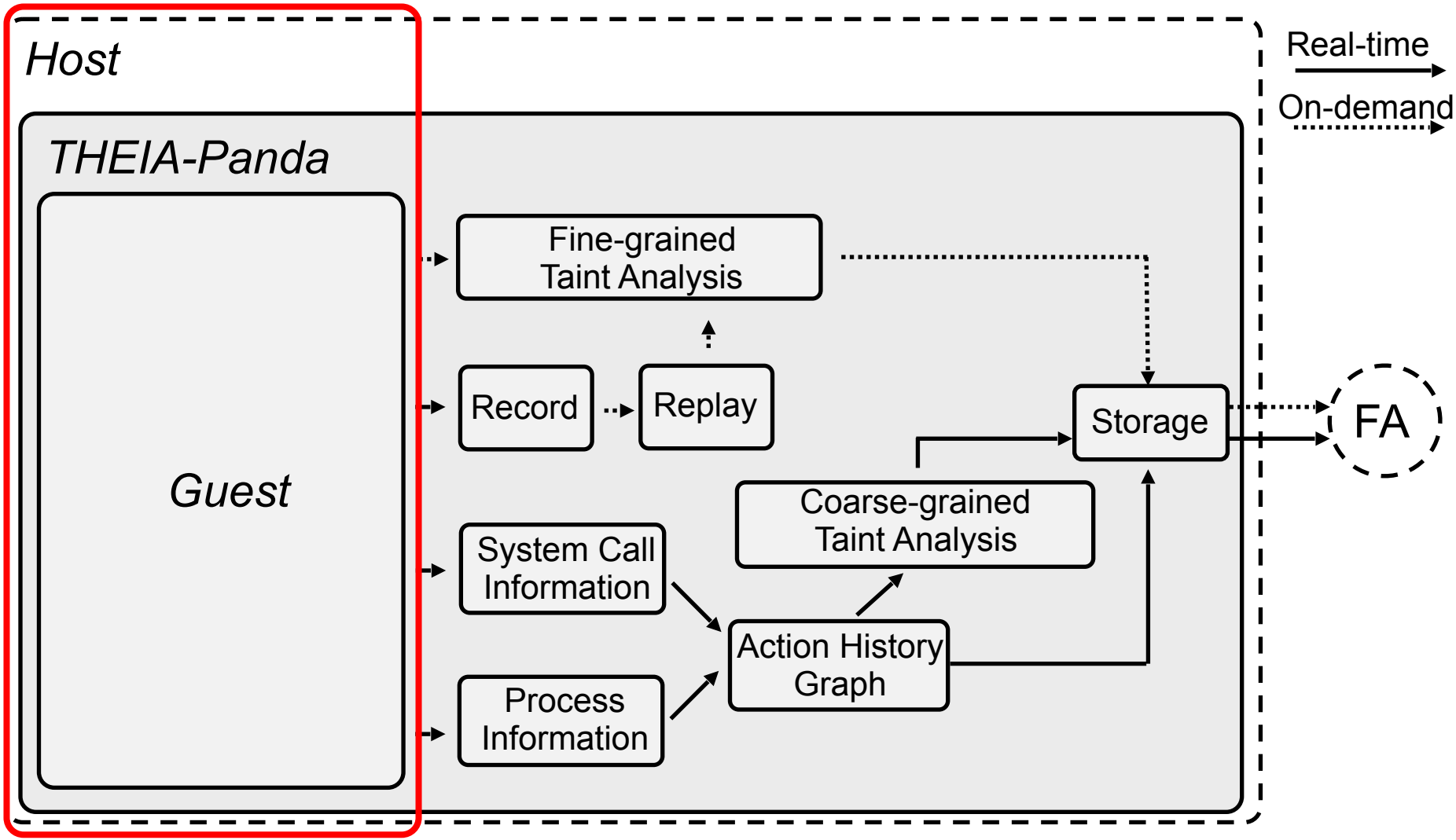




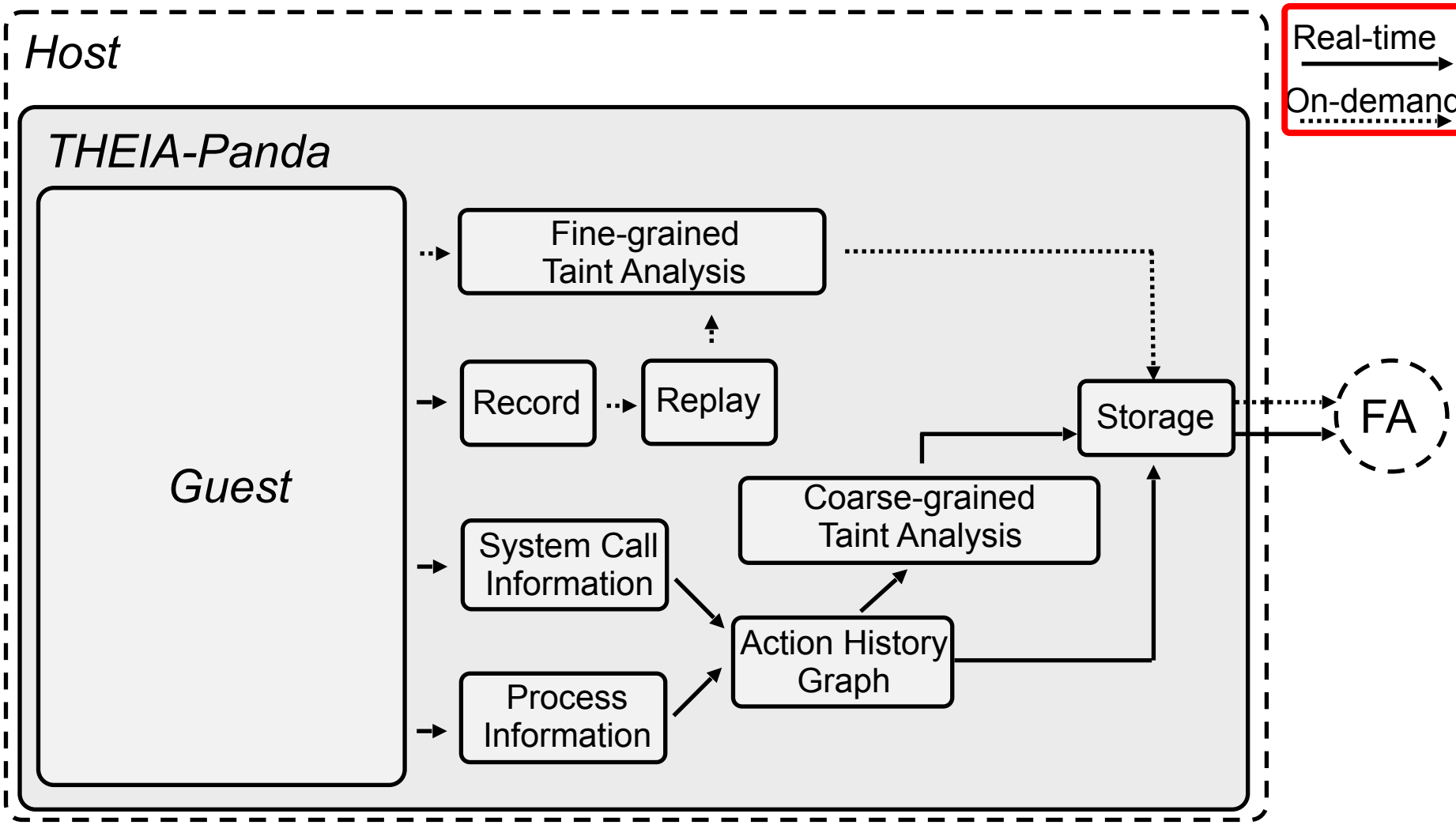
THEIA-Panda Overview



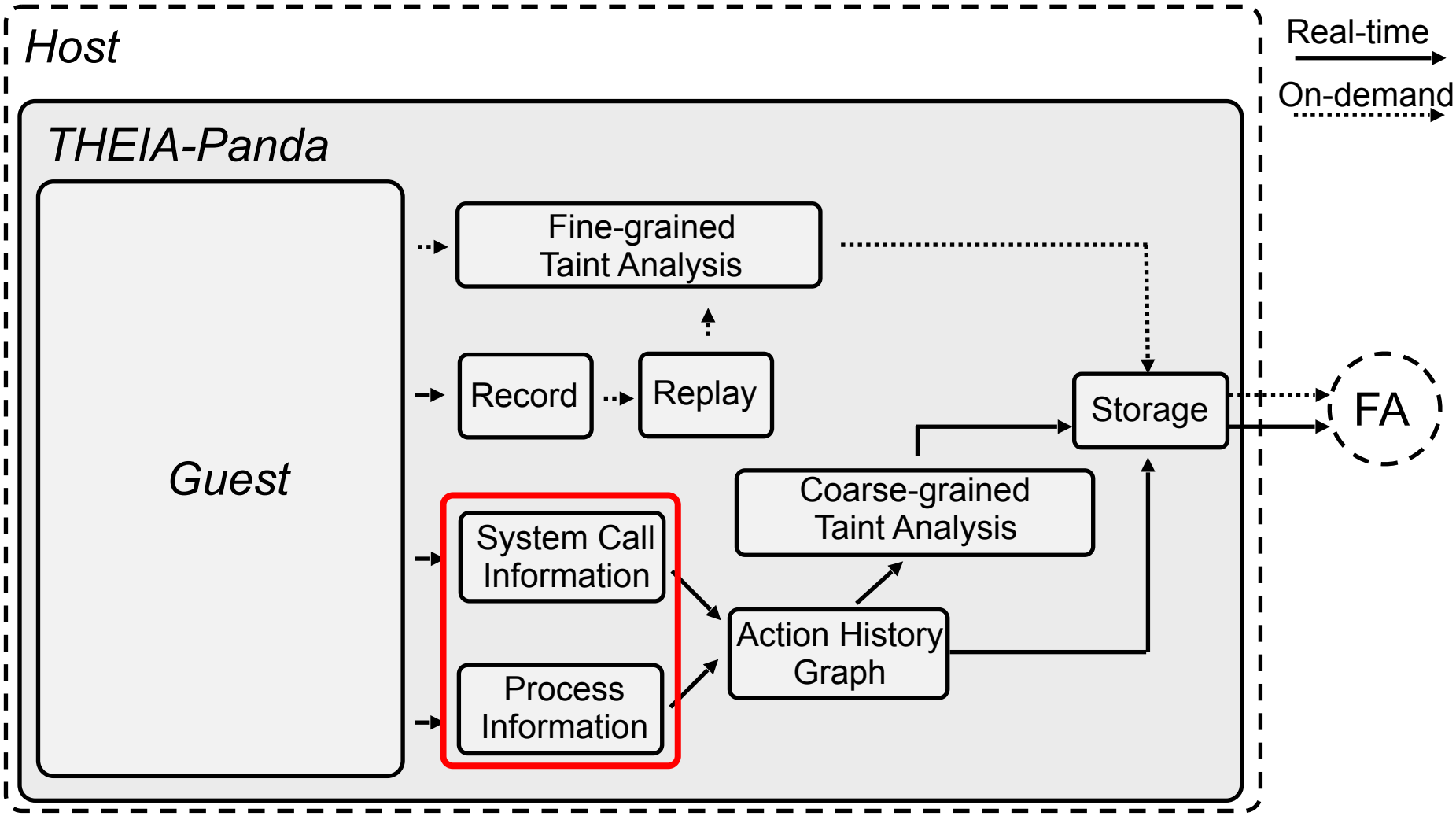
THEIA-Panda Overview



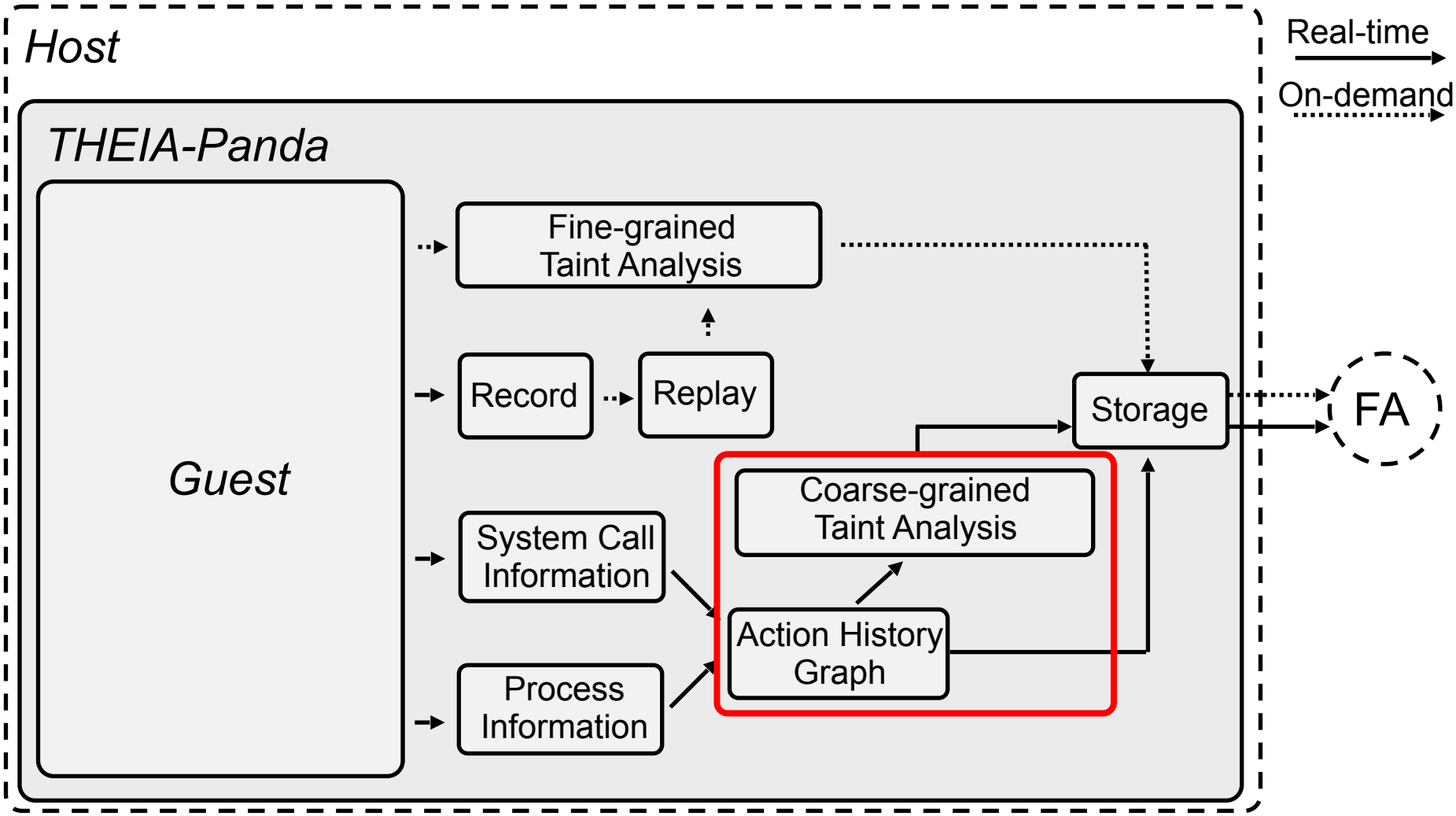
THEIA-Panda Overview



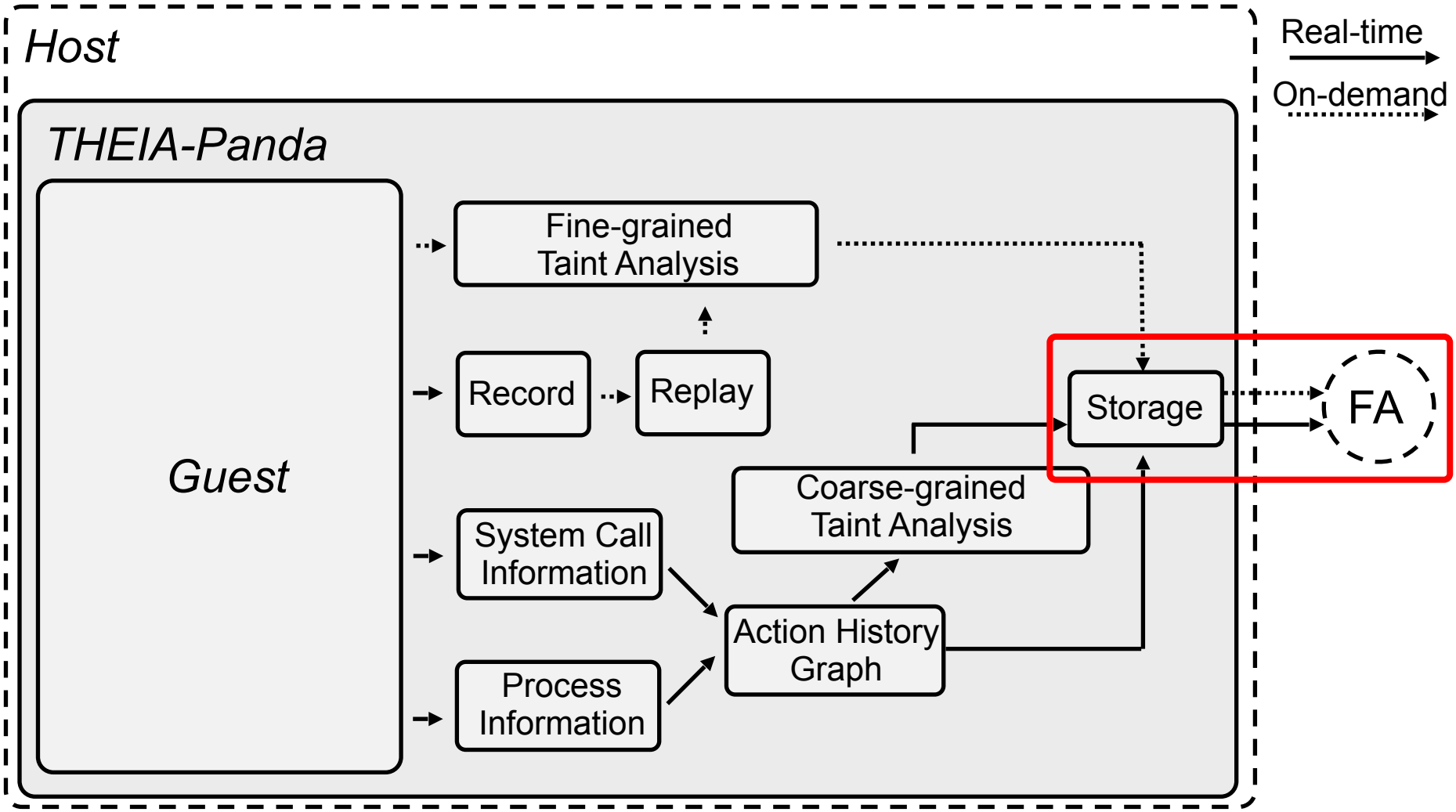
THEIA-Panda Overview



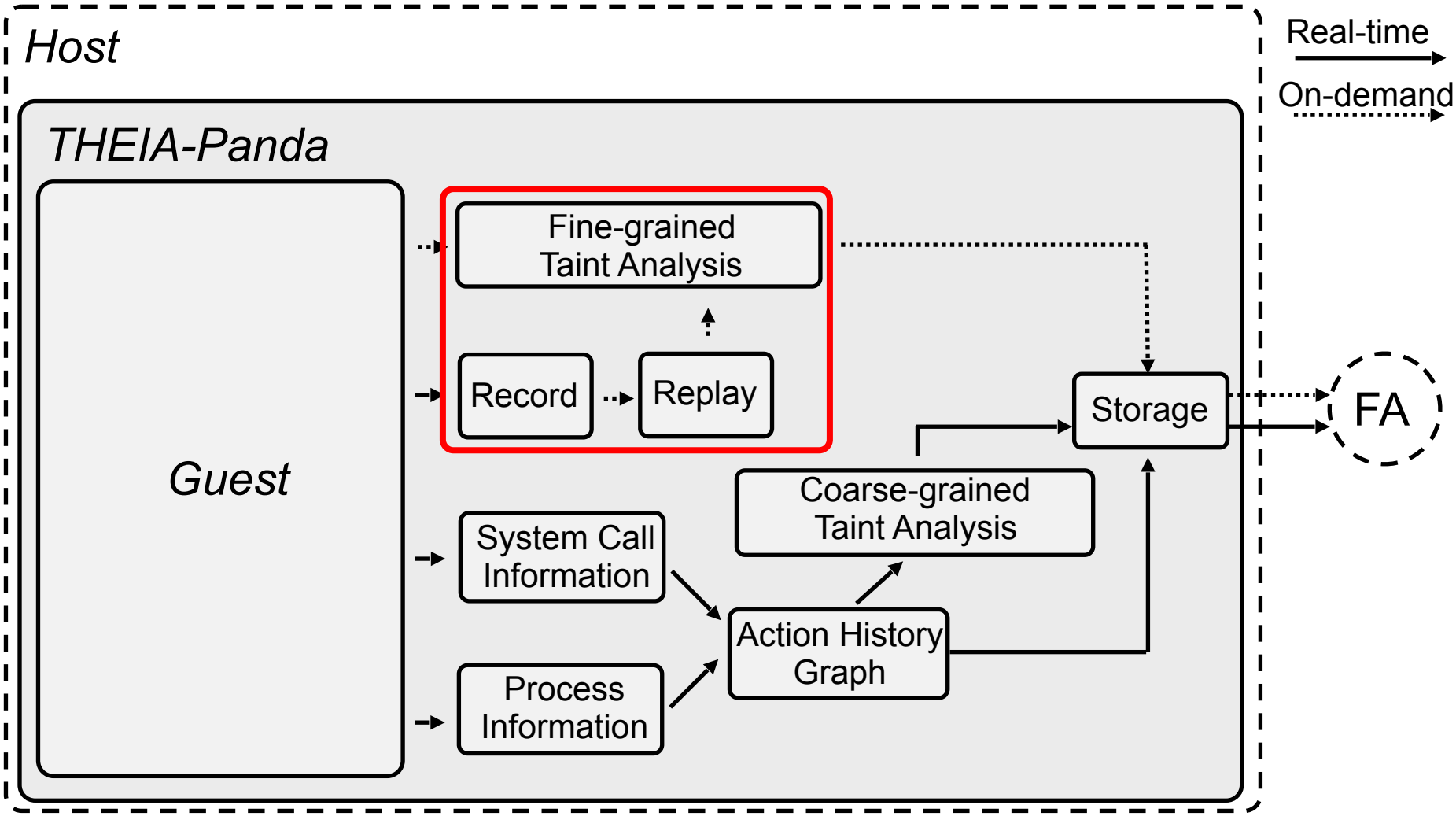
THEIA-Panda Overview



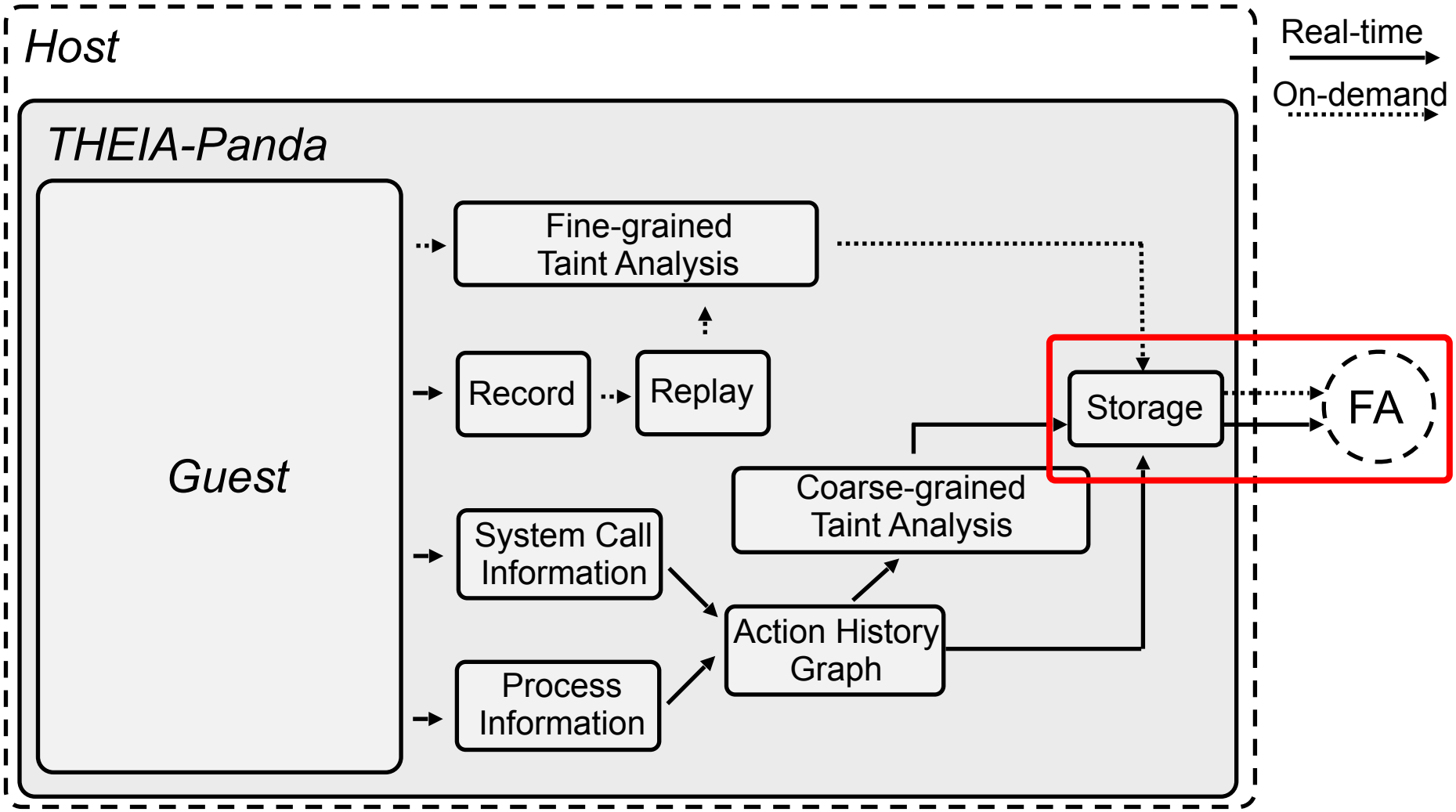
THEIA-Panda Overview



THEIA-Panda Overview



THEIA-Panda Overview



Record and Replay

- **Record:**
  - Take a snapshot of the machine state
  - Log non-deterministic inputs
    - Data entering CPU on port input
    - Hardware interrupts and their parameters
    - Data written to RAM during direct memory operation from peripheral
- **Replay:**
  - Replay activity (data) starting from snapshot of machine state
- **Implementation:**
  - QEMU/PANDA* and 64-bit Linux Guest

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. **Repeatable Reverse Engineering with PANDA**. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015

Record and Replay

- **Record:**
  - Take a snapshot of the machine state
  - Log non-deterministic inputs
    - Data entering CPU on port input
    - Hardware interrupts and their parameters
    - Data written to RAM during direct memory operation from peripheral
- **Replay:**
  - Replay activity (data) starting from snapshot of machine state
- **Implementation:**
  - QEMU/PANDA* and 64-bit Linux Guest

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. **Repeatable Reverse Engineering with PANDA**. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015

Record and Replay

- **Record:**
  - Take a snapshot of the machine state
  - Log non-deterministic inputs
    - Data entering CPU on port input
    - Hardware interrupts and their parameters
    - Data written to RAM during direct memory operation from peripheral
- **Replay:**
  - Replay activity (data) starting from snapshot of machine state
- **Implementation:**
  - QEMU/PANDA* and 64-bit Linux Guest

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. **Repeatable Reverse Engineering with PANDA**. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015

Record and Replay Implementation Example

```
static ssize_t
e1000_receive(VLANClientState *nc,
const uint8_t *buf, size_t size)
{
    …
    do {
        …
        pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr),
         (void *)(buf + desc_offset + vlan_offset), copy_size);
        rr_record_handle_packet_call(
                RR_CALLSITE_E1000_RECEIVE_2, (void *)(
                buf + desc_offset + vlan_offset),
                copy_size, NET_TRANSFER_IOB_TO_RAM)
         …
    } while (desc_offset < total_size);

    …
}
```

Record and Replay Implementation Example

```
static ssize_t
e1000_receive(VLANClientState *nc,
const uint8_t *buf, size_t size)
{
   …
   do {
      …
      pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr),
       (void *)(buf + desc_offset + vlan_offset), copy_size);
      rr_record_handle_packet_call(
            RR_CALLSITE_E1000_RECEIVE_2, (void *)(
            buf + desc_offset + vlan_offset),
            copy_size, NET_TRANSFER_IOB_TO_RAM)
       …
   } while (desc_offset < total_size);

   …
}
```

Record and Replay Implementation Example

```
static ssize_t
e1000_receive(VLANClientState *nc,
const uint8_t *buf, size_t size)
{
   …
   do {
      …
      pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr),
       (void *)(buf + desc_offset + vlan_offset), copy_size);
      rr_record_handle_packet_call(
            RR_CALLSITE_E1000_RECEIVE_2, (void *)(
            buf + desc_offset + vlan_offset),
            copy_size, NET_TRANSFER_IOB_TO_RAM)
       …
   } while (desc_offset < total_size);

   …
}
```

OS-level Transparency

- **Goal:**
  - Capture events and dependencies of OS-level events
- **Approach:**
  - Based on VM introspection
- **Events analyzed:**
  - Process operations:
    - clone, fork, execve, exit, etc.
  - File operations:
    - open, read, write, unlink, etc.
  - Network operations:
    - socket, connect, recvmsg, etc.
  - Memory operations:
    - mmap, mprotect, shmget, etc.

OS-level Transparency

- **Goal:**
  - Capture events and dependencies of OS-level events
- **Approach:**
  - Based on VM introspection
- **Events analyzed:**
  - Process operations:
    - clone, fork, execve, exit, etc.
  - File operations:
    - open, read, write, unlink, etc.
  - Network operations:
    - socket, connect, recvmsg, etc.
  - Memory operations:
    - mmap, mprotect, shmget, etc.

OS-level Transparency

- **Goal:**
  - Capture events and dependencies of OS-level events
- **Approach:**
  - Based on VM introspection
- **Events analyzed:**
  - Process operations:
    - clone, fork, execve, exit, etc.
  - File operations:
    - open, read, write, unlink, etc.
  - Network operations:
    - socket, connect, recvmsg, etc.
  - Memory operations:
    - mmap, mprotect, shmget, etc.

OS-level Transparency Implementation Example

```
#ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{

        panda_cb_list *plist;
        for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
plist != NULL; plist = panda_cb_list_next(plist))
        {
                plist->entry.before_syscall(env);
        }
        …
}
```

OS-level Transparency Implementation Example

```
#ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{
        panda_cb_list *plist;
        for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
plist != NULL; plist = panda_cb_list_next(plist))
        {
                plist->entry.before_syscall(env);
        }
        …
}
```

OS-level Transparency Implementation Example

```
#ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{

        panda_cb_list *plist;
        for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
plist != NULL; plist = panda_cb_list_next(plist))
        {
                plist->entry.before_syscall(env);
        }
        …
}
```

Action History Graph (AHG)

- **Goal:**
  - Represent causality across events
- **Causality:**
  - Process->Process (e.g., fork)
  - Process->File (e.g., write)
  - File->Process (e.g., read)
  - Process->Host (e.g., send)
  - Host->Process (e.g., recv)

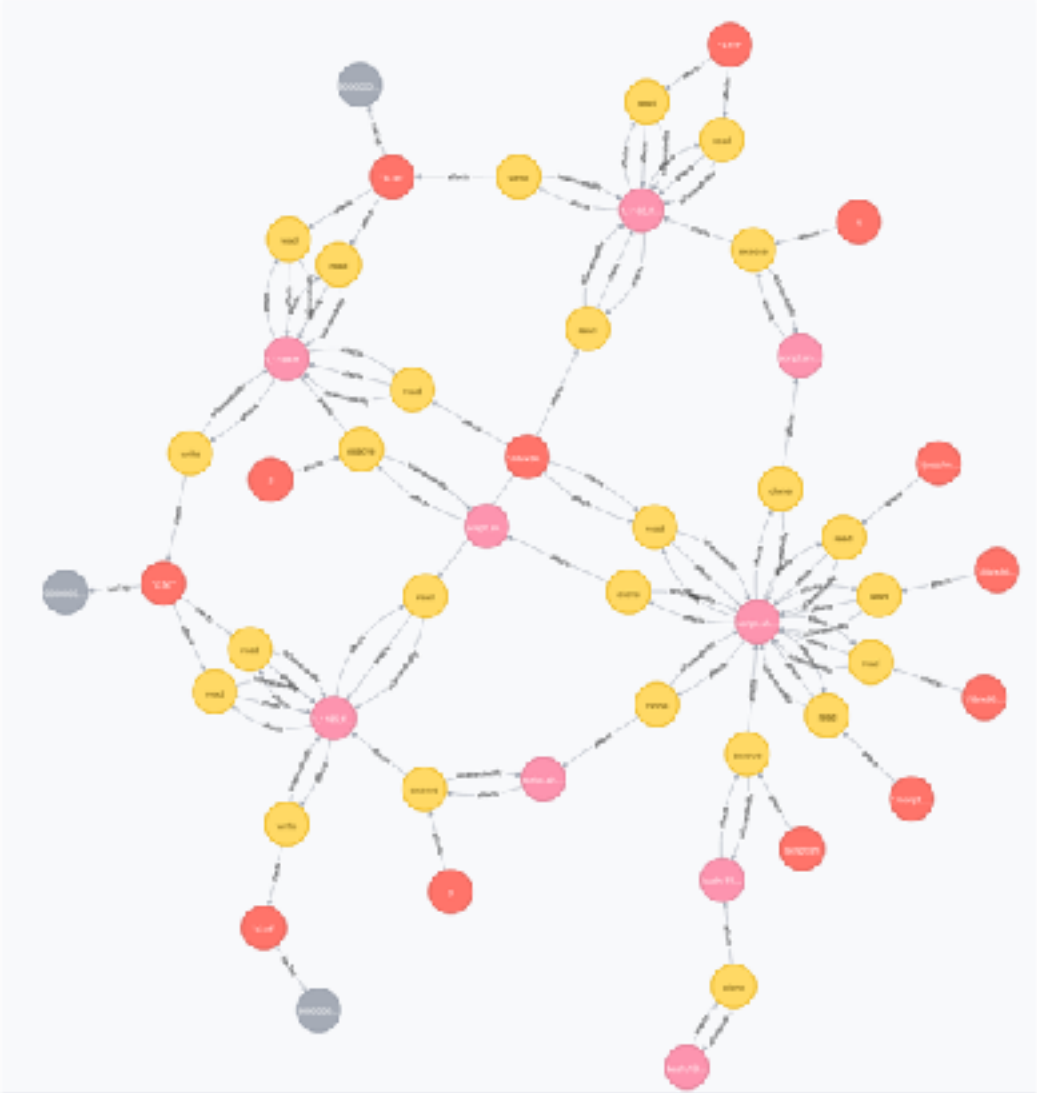
Action History Graph (AHG)

- **Goal:**
  - Represent causality across events
- **Causality:**
  - Process->Process (e.g., fork)
  - Process->File (e.g., write)
  - File->Process (e.g., read)
  - Process->Host (e.g., send)
  - Host->Process (e.g., recv)

Action History Graph (AHG)

- **Goal:**
  - Represent causality across events
- **Causality:**
  - Process->Process (e.g., fork)
  - Process->File (e.g., write)
  - File->Process (e.g., read)
  - Process->Host (e.g., send)
  - Host->Process (e.g., recv)

Action History Graph Example



Coarse-grained Taint Analysis

- **Goal:**
  - Quickly capture the provenance of objects in the AHG
- **Working mechanism:**
  - Runs while building AHG
  - Processes have a provenance set
  - Process operations:
    - fork, clone: copy provenance of parent to child process
  - File and network operations
    - read, recv: associate provenance of object to process
    - write, send: associate provenance of process to object

Coarse-grained Taint Analysis

- **Goal:**
  - Quickly capture the provenance of objects in the AHG
- **Working mechanism:**
  - Runs while building AHG
  - Processes have a provenance set
  - Process operations:
    - fork, clone: copy provenance of parent to child process
  - File and network operations
    - read, recv: associate provenance of object to process
    - write, send: associate provenance of process to object

Coarse-grained Taint Analysis

- **Goal:**
  - Quickly capture the provenance of objects in the AHG
- **Working mechanism:**
  - Runs while building AHG
  - Processes have a provenance set
  - Process operations:
    - fork, clone: copy provenance of parent to child process
  - File and network operations
    - read, recv: associate provenance of object to process
    - write, send: associate provenance of process to object

Fine-grained Taint Analysis

- **Goal:**
  - Accurately capture provenance of objects in the AHG
- **Working mechanism:**
  - Decoupled from program execution
  - Instruction level propagation
  - Taint tags at byte level granularity
- **Optimizations:**
  - Trace-based dynamic taint analysis

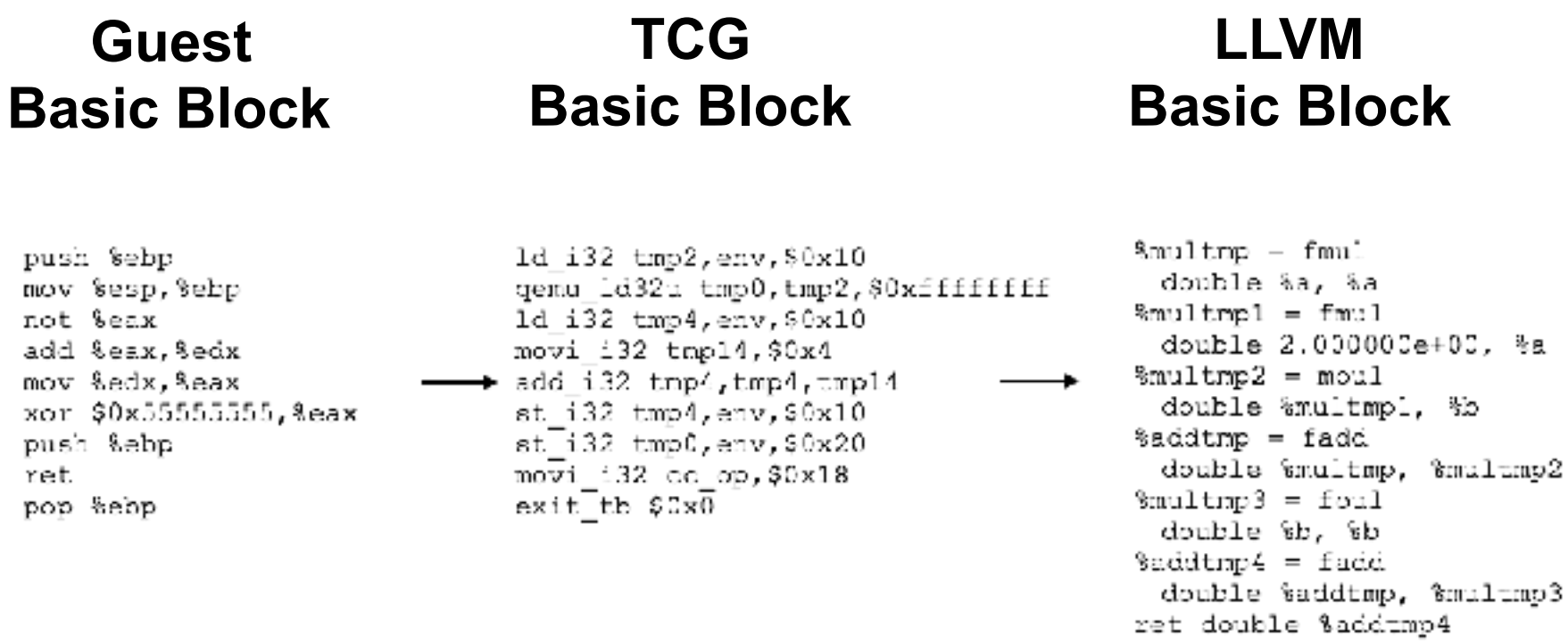
Fine-grained Taint Analysis

- **Goal:**
  - Accurately capture provenance of objects in the AHG
- **Working mechanism:**
  - Decoupled from program execution
  - Instruction level propagation
  - Taint tags at byte level granularity
- **Optimizations:**
  - Trace-based dynamic taint analysis

Fine-grained Taint Analysis

- **Goal:**
  - Accurately capture provenance of objects in the AHG
- **Working mechanism:**
  - Decoupled from program execution
  - Instruction level propagation
  - Taint tags at byte level granularity
- **Optimizations:**
  - Trace-based dynamic taint analysis

Fine-grained Taint Analysis Implementation



Fine-grained Taint Analysis Implementation

Guest Basic Block

```
push %ebp
mov %esp,%ebp
not %eax
add %eax,%edx
mov %edx,%eax
xor $0x55555555,%eax
push %ebp
ret
pop %ebp
```

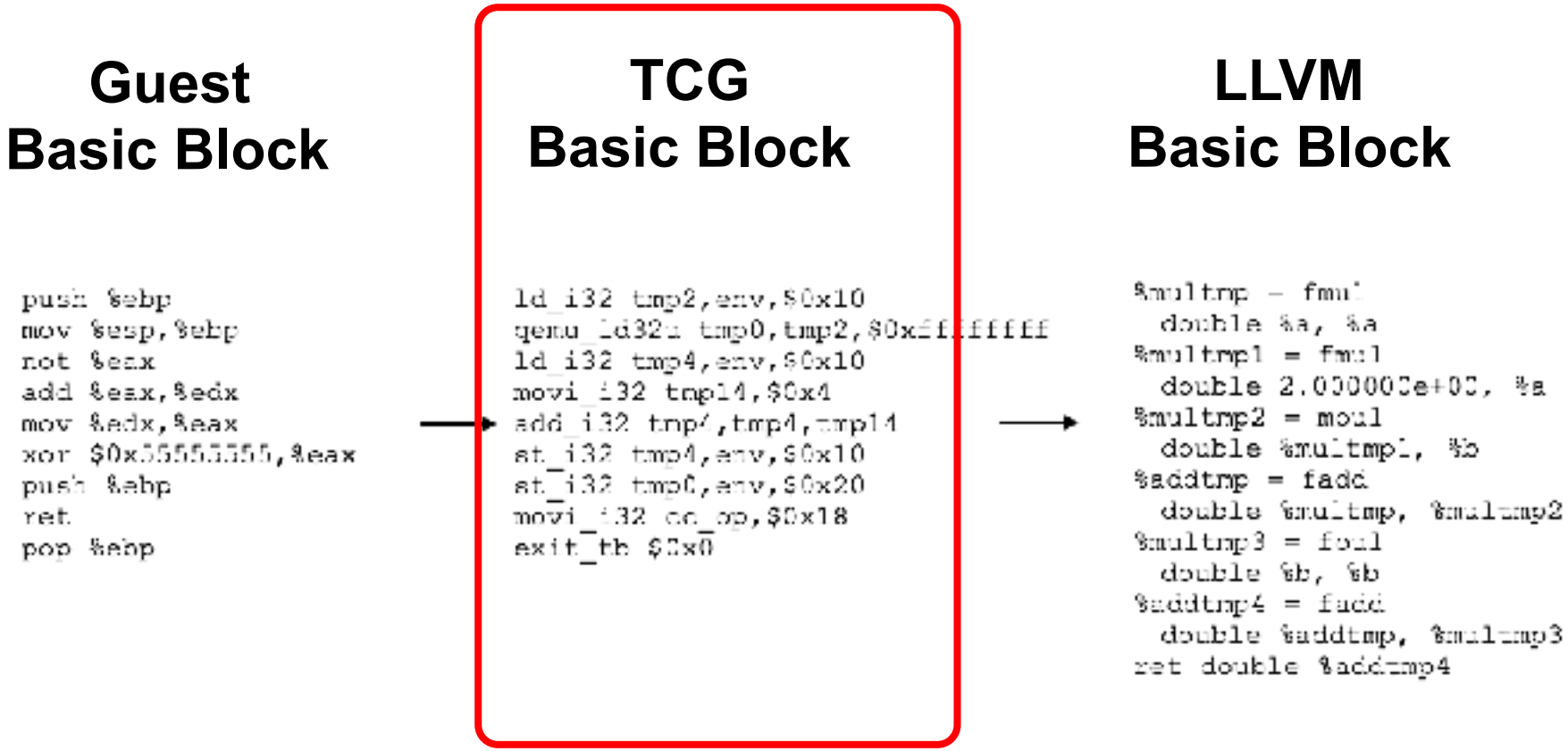
TCG Basic Block

```
ld_i32 tmp2,env,$0x10
qemu_ld32u tmp0,tmp2,$0xffffffff
ld_i32 tmp4,env,$0x10
movi_i32 tmp14,$0x4
add_i32 tmp4,tmp4,tmp14
st_i32 tmp4,env,$0x10
st_i32 tmp0,env,$0x20
movi_i32 cc_op,$0x18
exit_tb $0x0
```

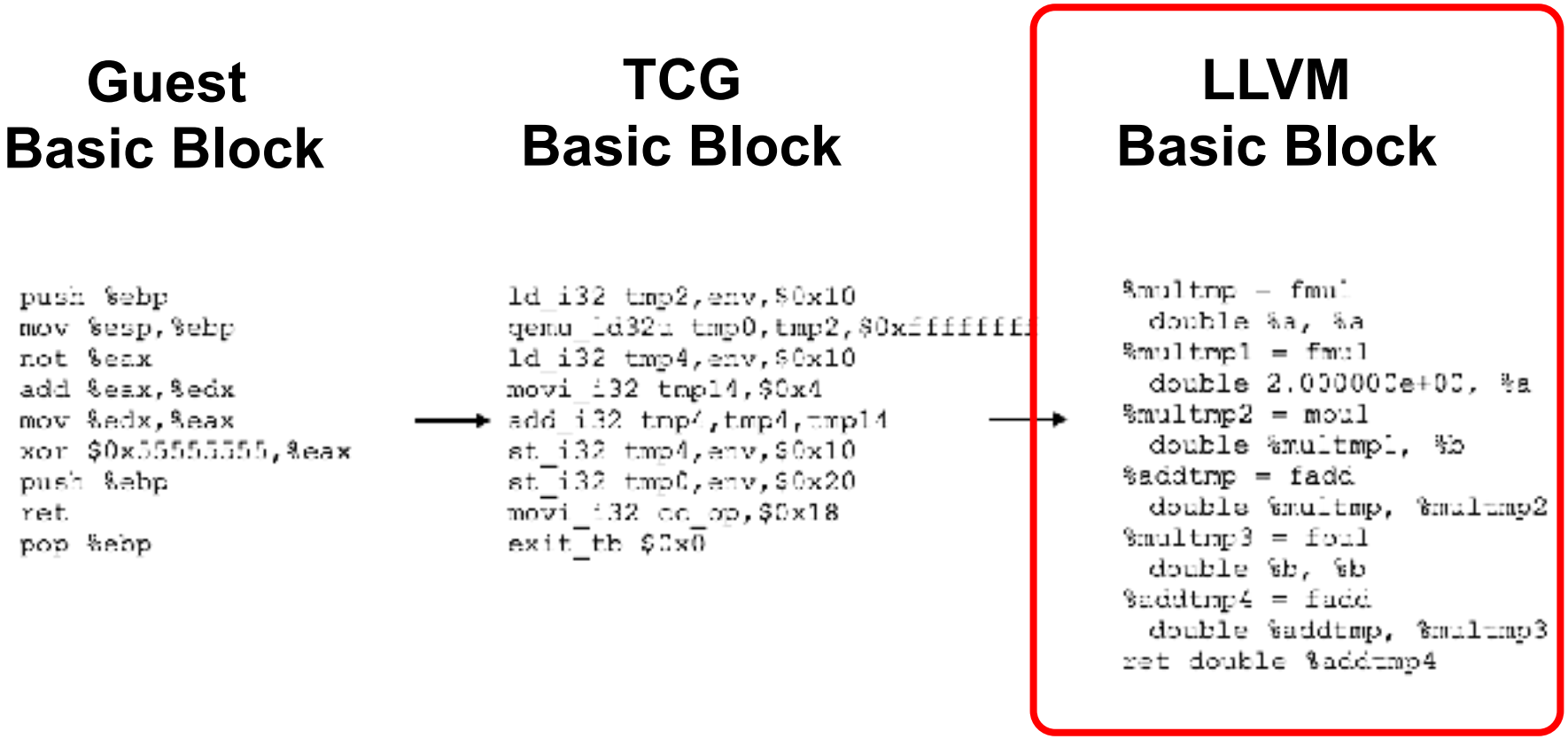
LLVM Basic Block

```
%multmp = fmul
  double %a, %a
%multmp1 = fmul
  double 2.000000e+00, %a
%multmp2 = mul
  double %multmp1, %b
%addtmp = fadd
  double %multmp, %multmp2
%multmp3 = fmul
  double %b, %b
%addtmp4 = fadd
  double %addtmp, %multmp3
ret double %addtmp4
```

Fine-grained Taint Analysis Implementation



Fine-grained Taint Analysis Implementation



Trace-based Taint Analysis

- **Objective:**
  - Improve performance of fine-grained taint analysis
- **Key intuition:**
  - Within a trace instruction sequences are executed multiple times
- **Working mechanism:**
  - Based on the execution trace of the system/program
  - Computes taint summaries for sequences of instructions
  - Re-use taint summaries on the trace and possible across traces
- **Implementation:**
  - Sequitur algorithm: recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  - Analyze grammar and reuse taint results when possible

Trace-based Taint Analysis

- **Objective:**
  - Improve performance of fine-grained taint analysis
- **Key intuition:**
  - Within a trace instruction sequences are executed multiple times
- **Working mechanism:**
  - Based on the execution trace of the system/program
  - Computes taint summaries for sequences of instructions
  - Re-use taint summaries on the trace and possible across traces
- **Implementation:**
  - Sequitur algorithm: recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  - Analyze grammar and reuse taint results when possible

Trace-based Taint Analysis

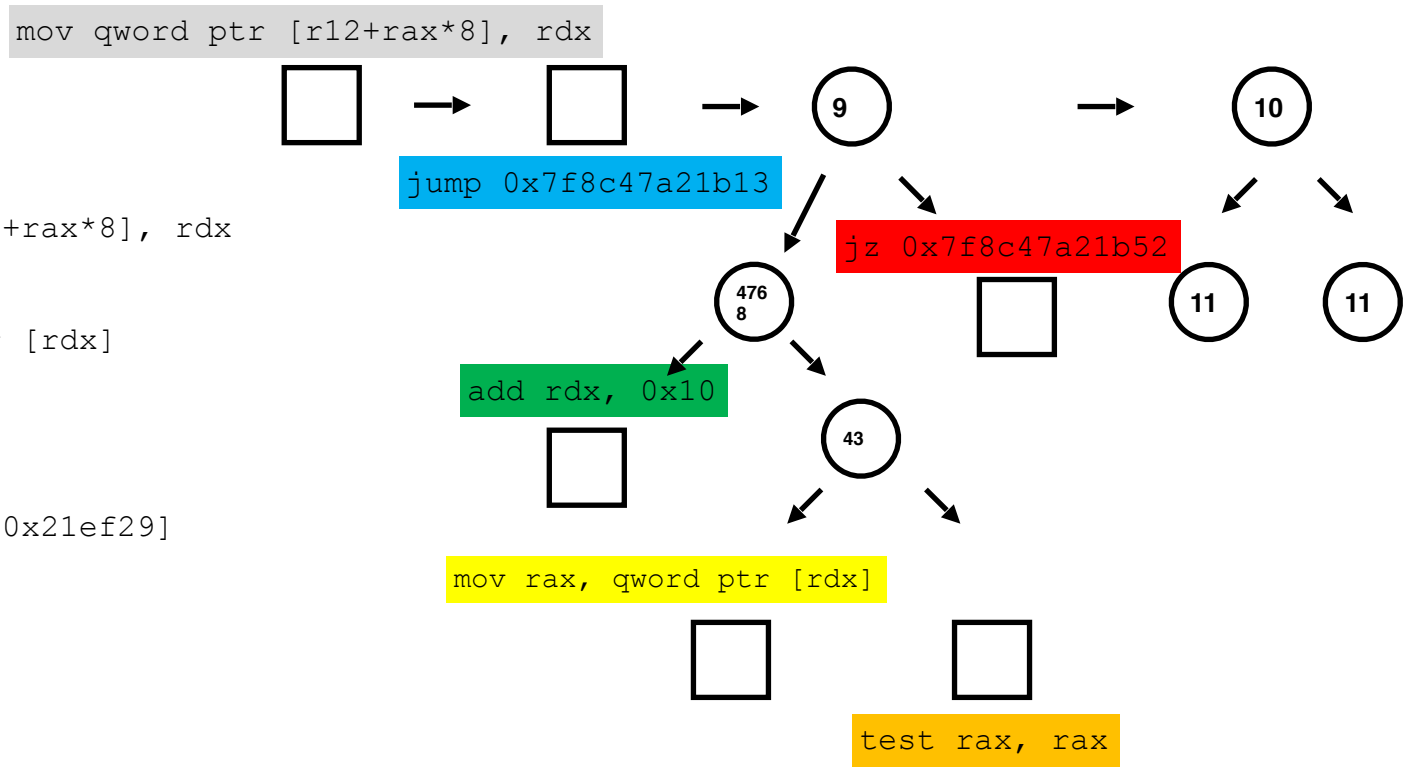
- **Objective:**
  - Improve performance of fine-grained taint analysis
- **Key intuition:**
  - Within a trace instruction sequences are executed multiple times
- **Working mechanism:**
  - Based on the execution trace of the system/program
  - Computes taint summaries for sequences of instructions
  - Re-use taint summaries on the trace and possible across traces
- **Implementation:**
  - Sequitur algorithm: recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  - Analyze grammar and reuse taint results when possible

Trace-based Taint Analysis Example

Execution Trace

```
…
mov qword ptr [r12+rax*8], rdx
jmp 0x7f8c47a21b13
add rdx, 0x10
mov rax, qword ptr [rdx]
test rax, rax
jz 0x7f8c47a21b52
cmp rax, 0x21
jbe 0x7f8c47a21b08
lea rcx, ptr [rip+0x21ef29]
…
```

Grammar

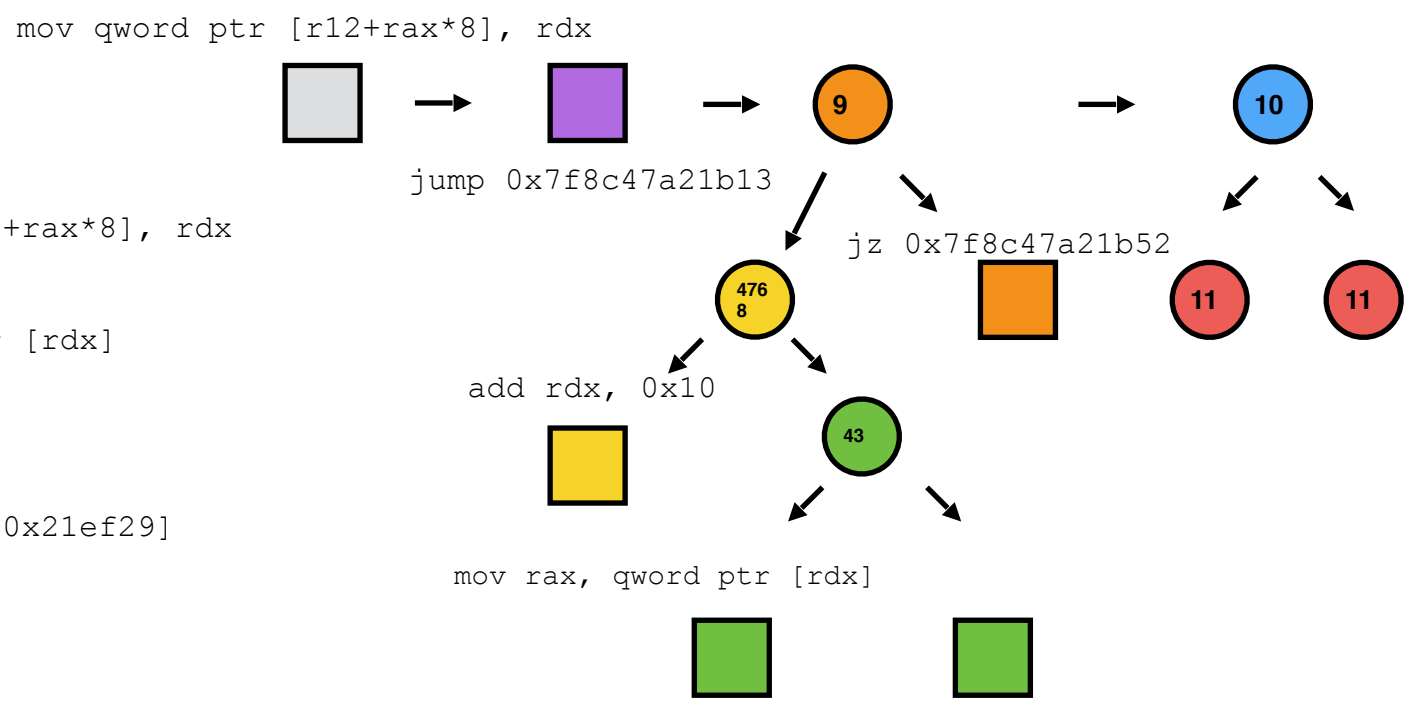


Trace-based Taint Analysis Example

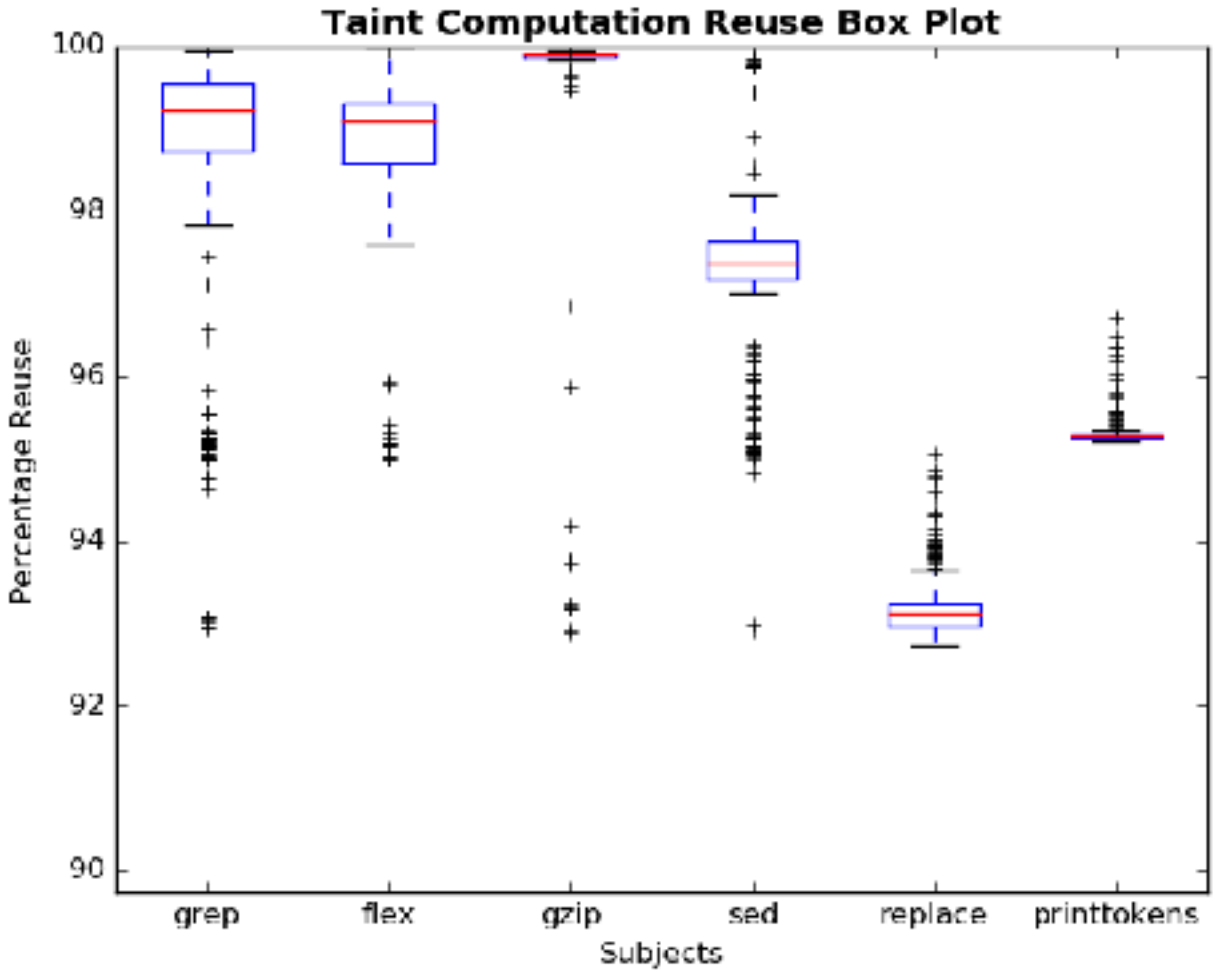
Execution Trace

```
…
mov qword ptr [r12+rax*8], rdx
jmp 0x7f8c47a21b13
add rdx, 0x10
mov rax, qword ptr [rdx]
test rax, rax
jz 0x7f8c47a21b52
cmp rax, 0x21
jbe 0x7f8c47a21b08
lea rcx, ptr [rip+0x21ef29]
…
```

Grammar

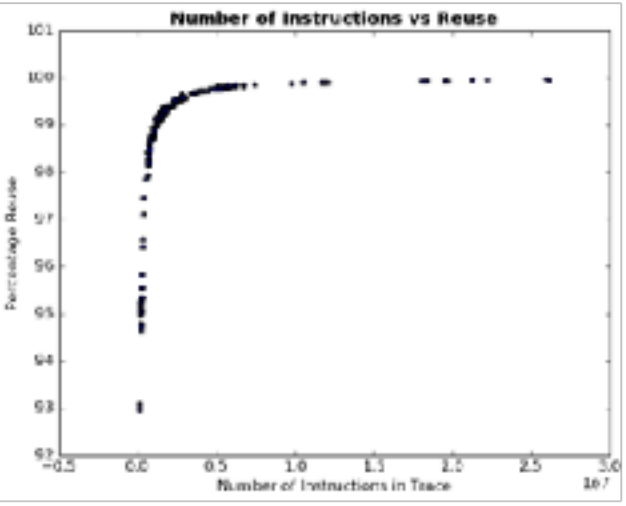


Fine-grained Taint Analysis

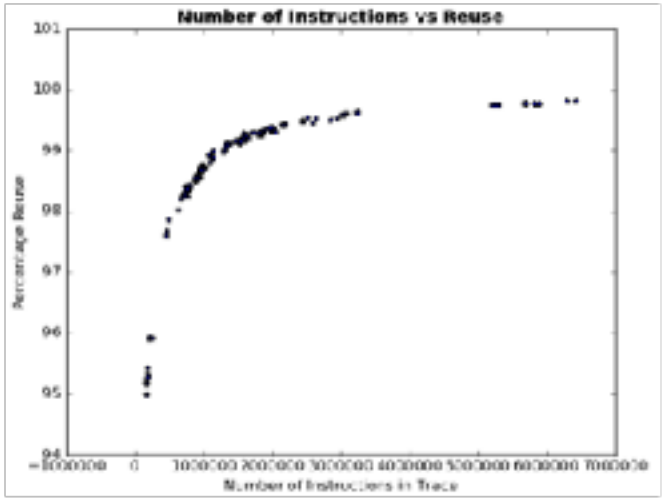


Fine-grained Taint Analysis

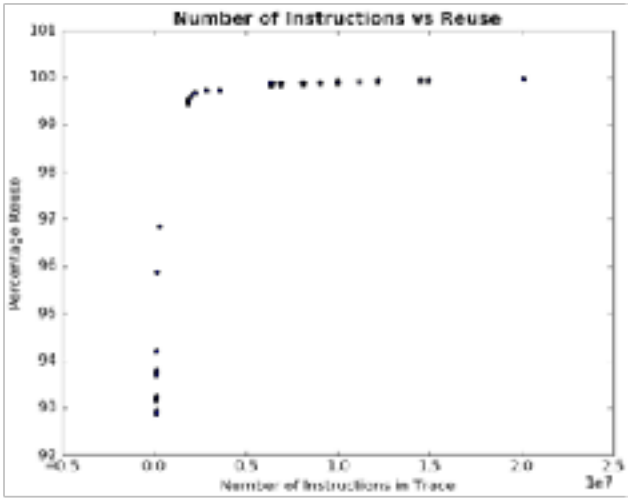
flex



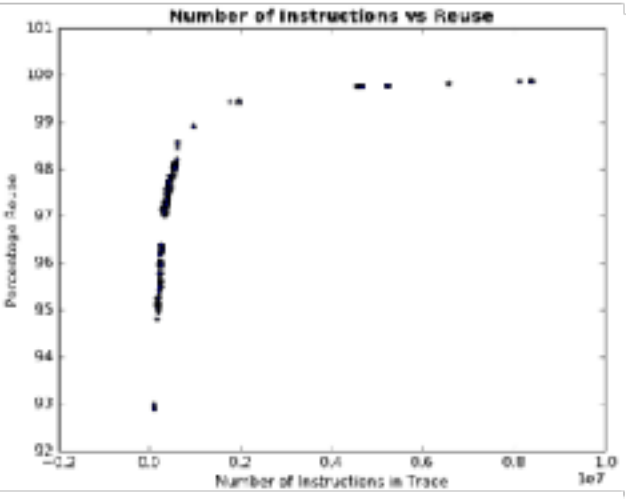
grep



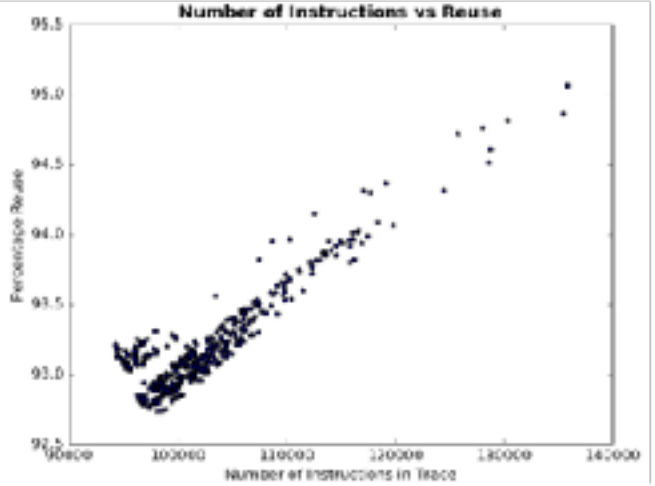
gzip



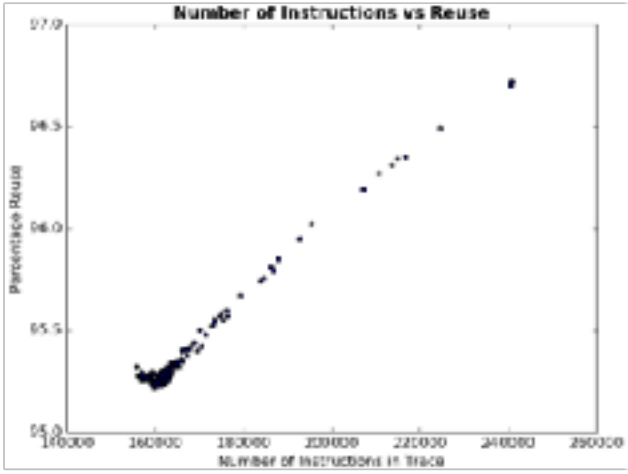
sed



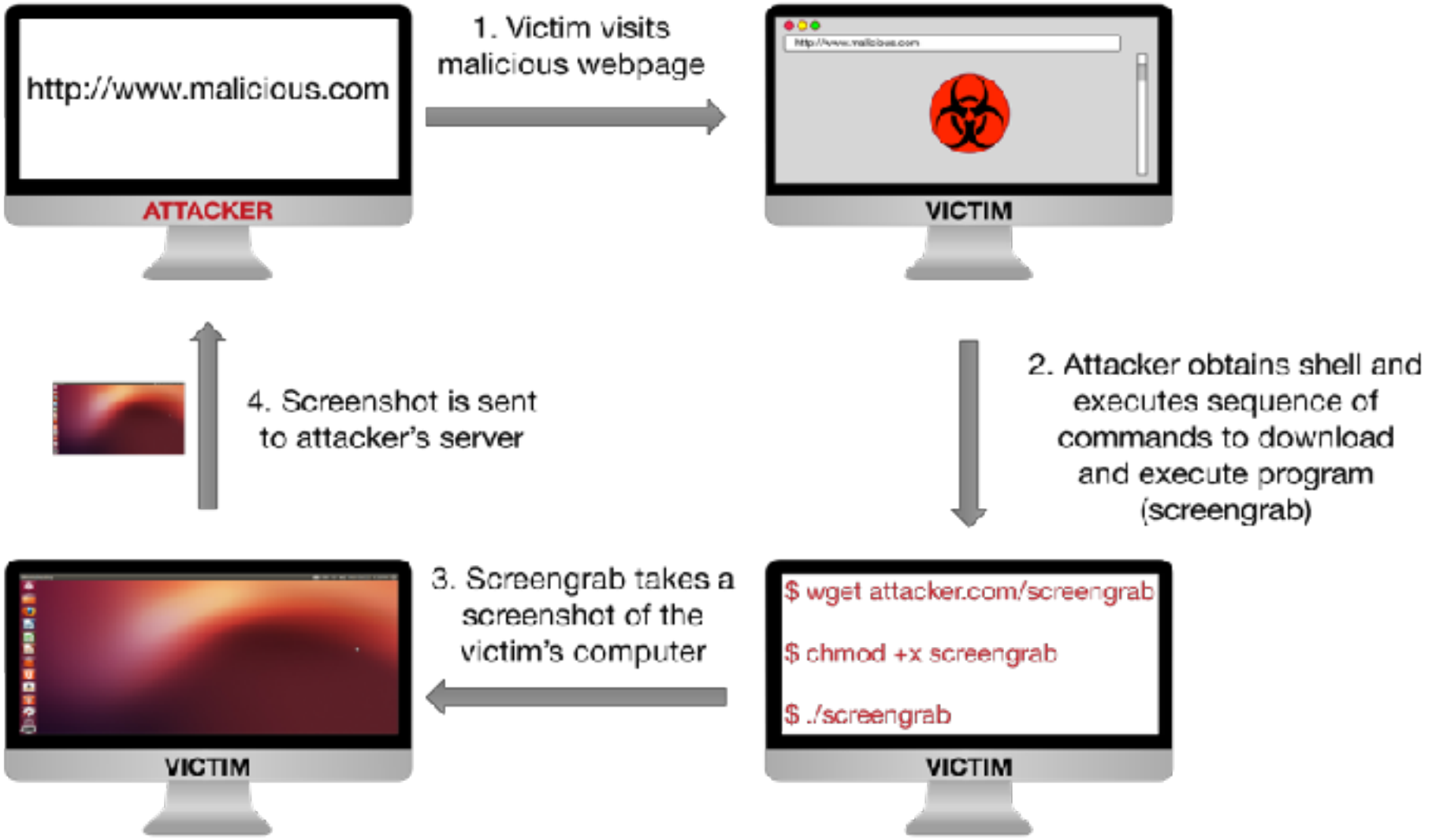
schedule



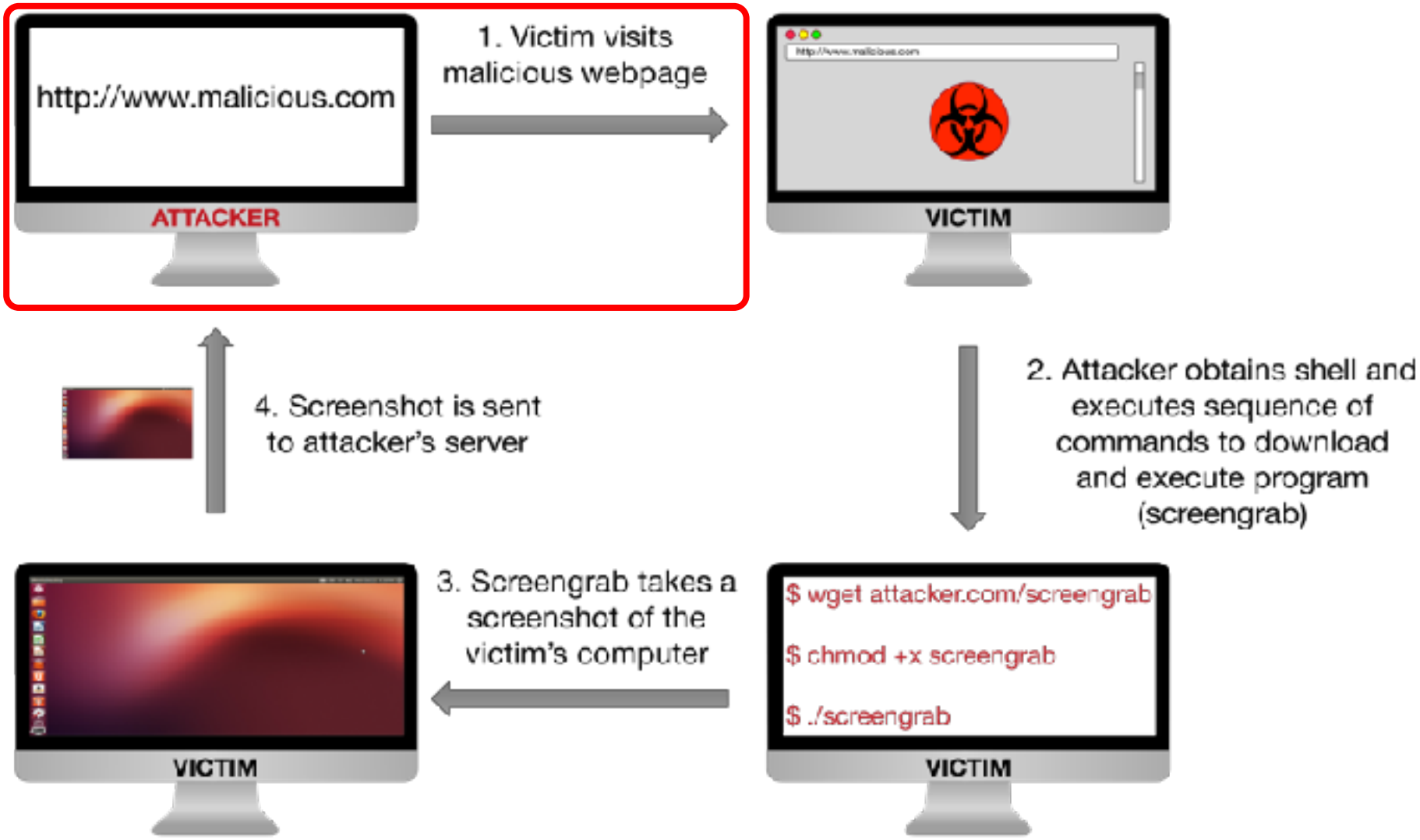
printtokens



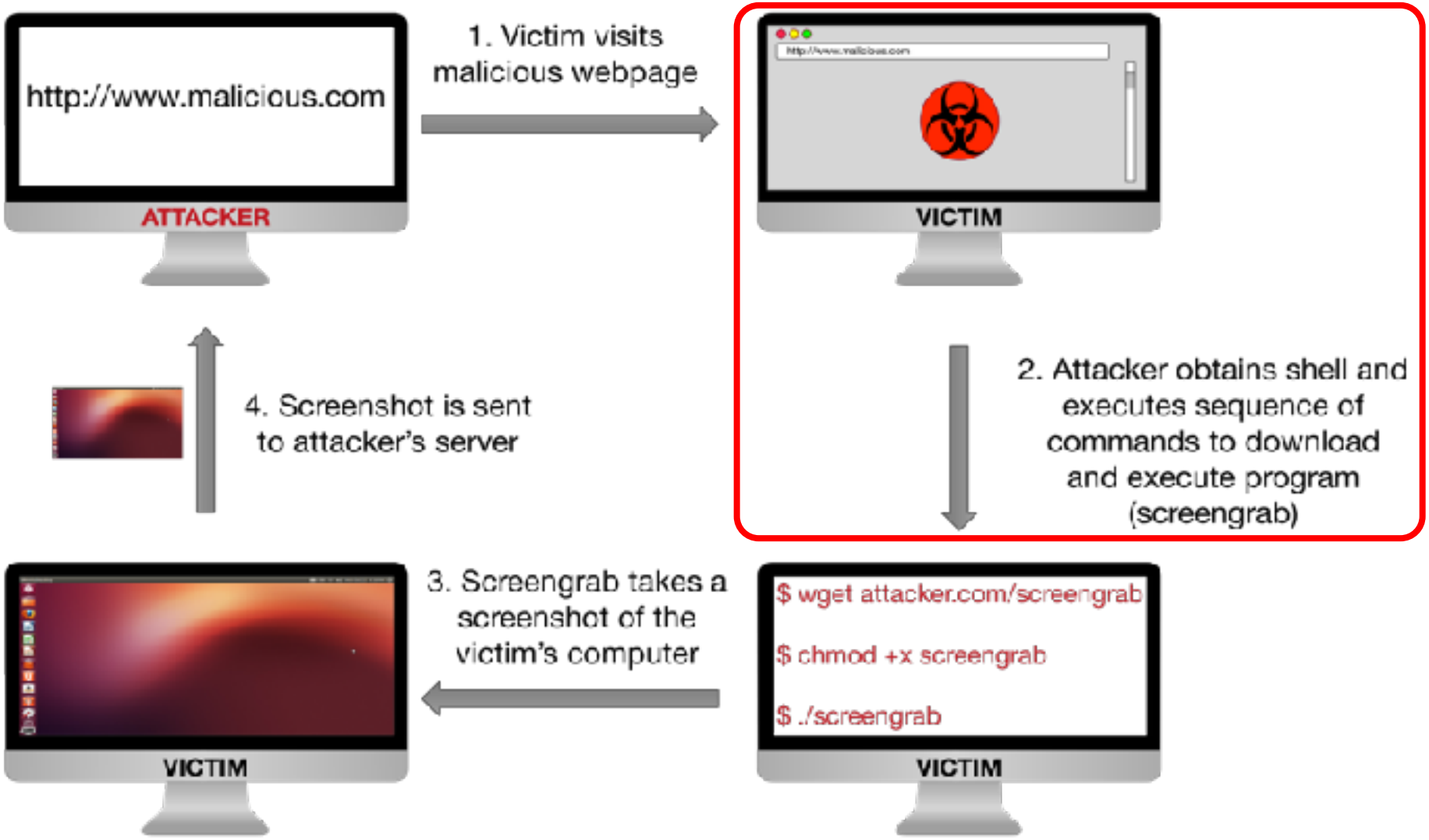
Case Study Overview



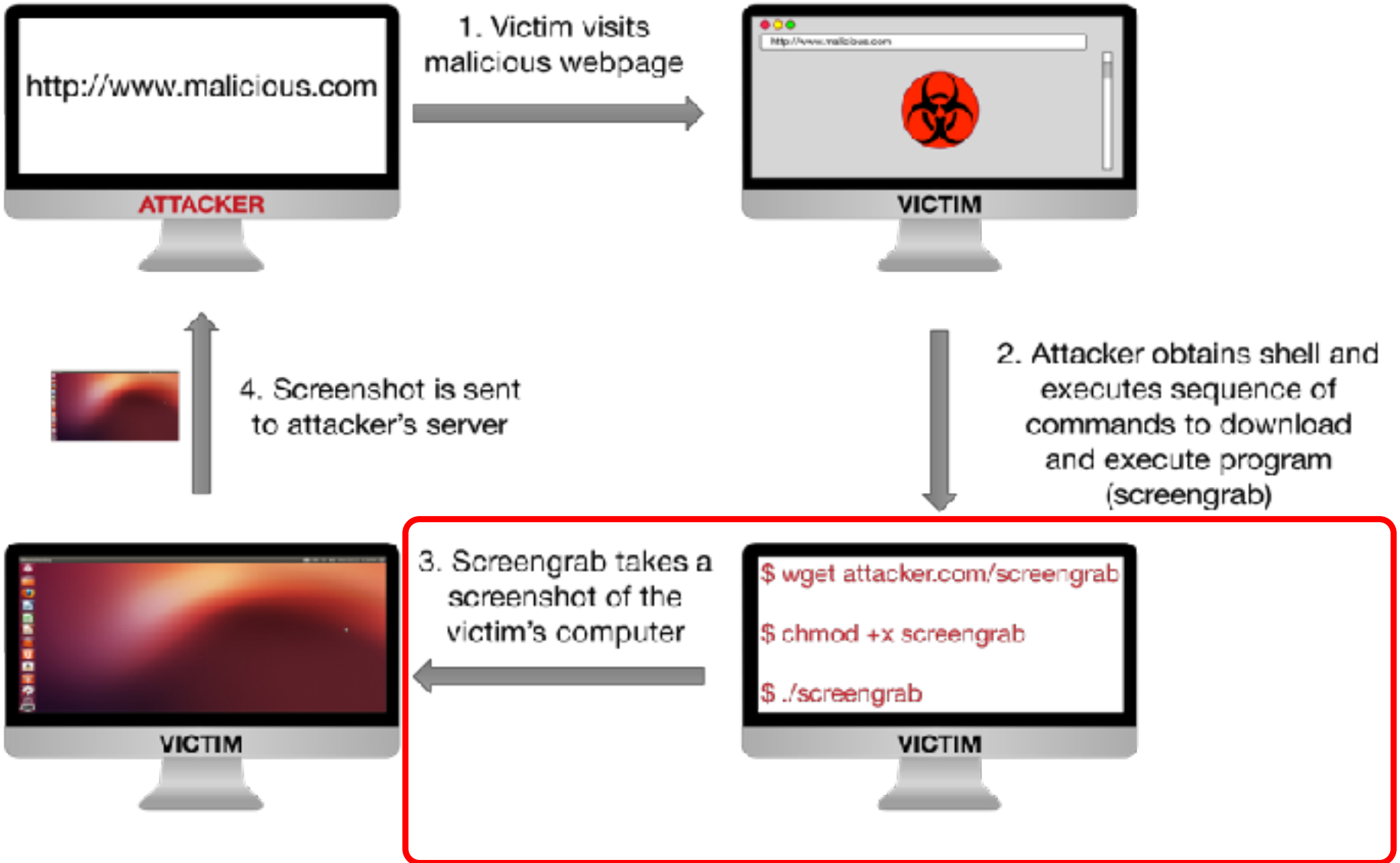
Case Study Overview



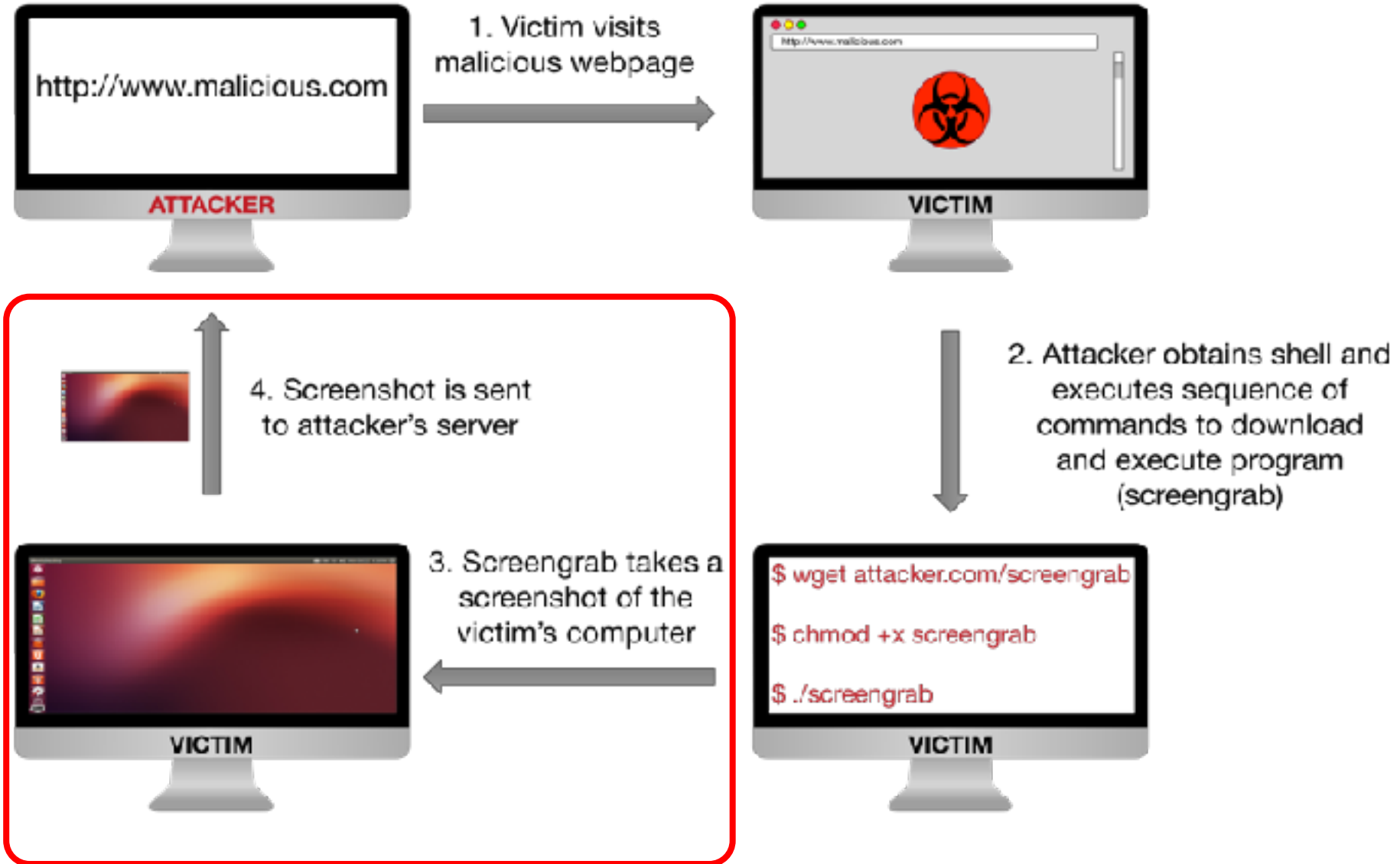
Case Study Overview



Case Study Overview

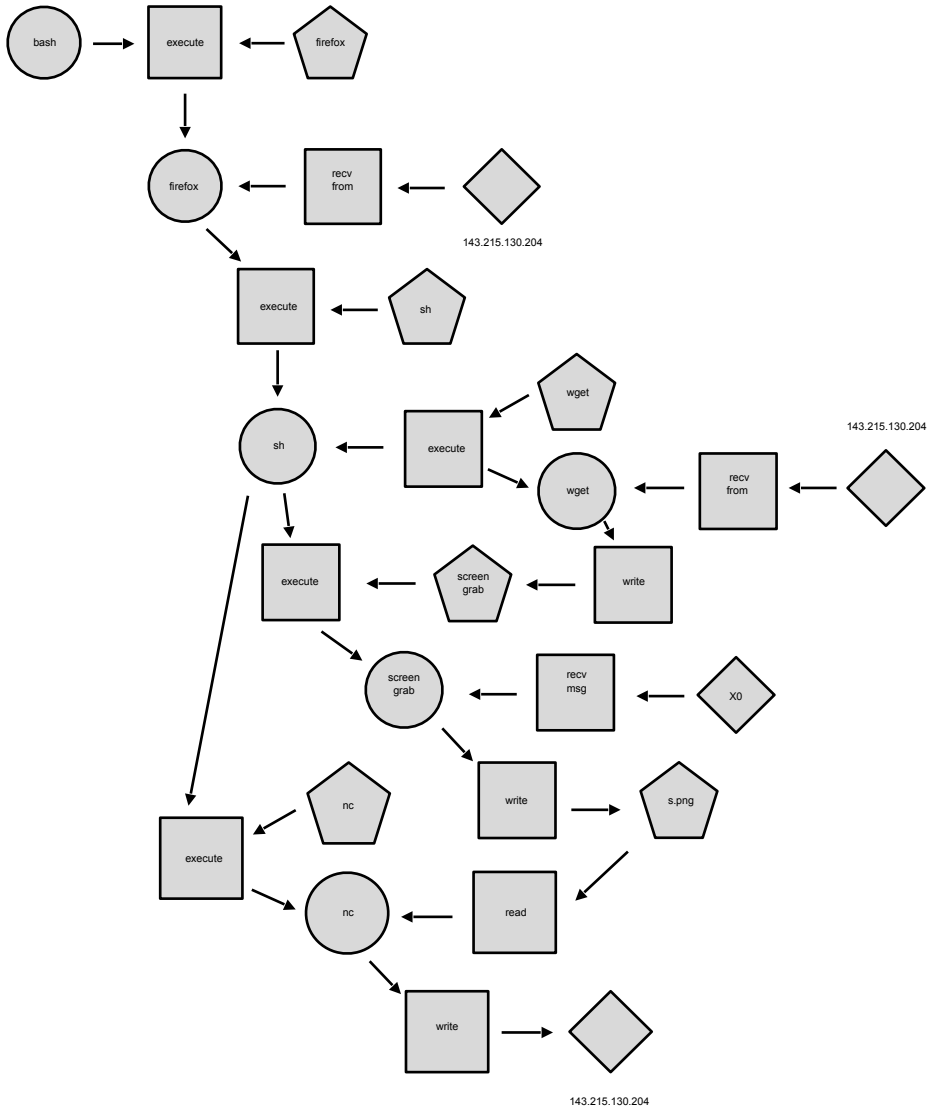


Case Study Overview

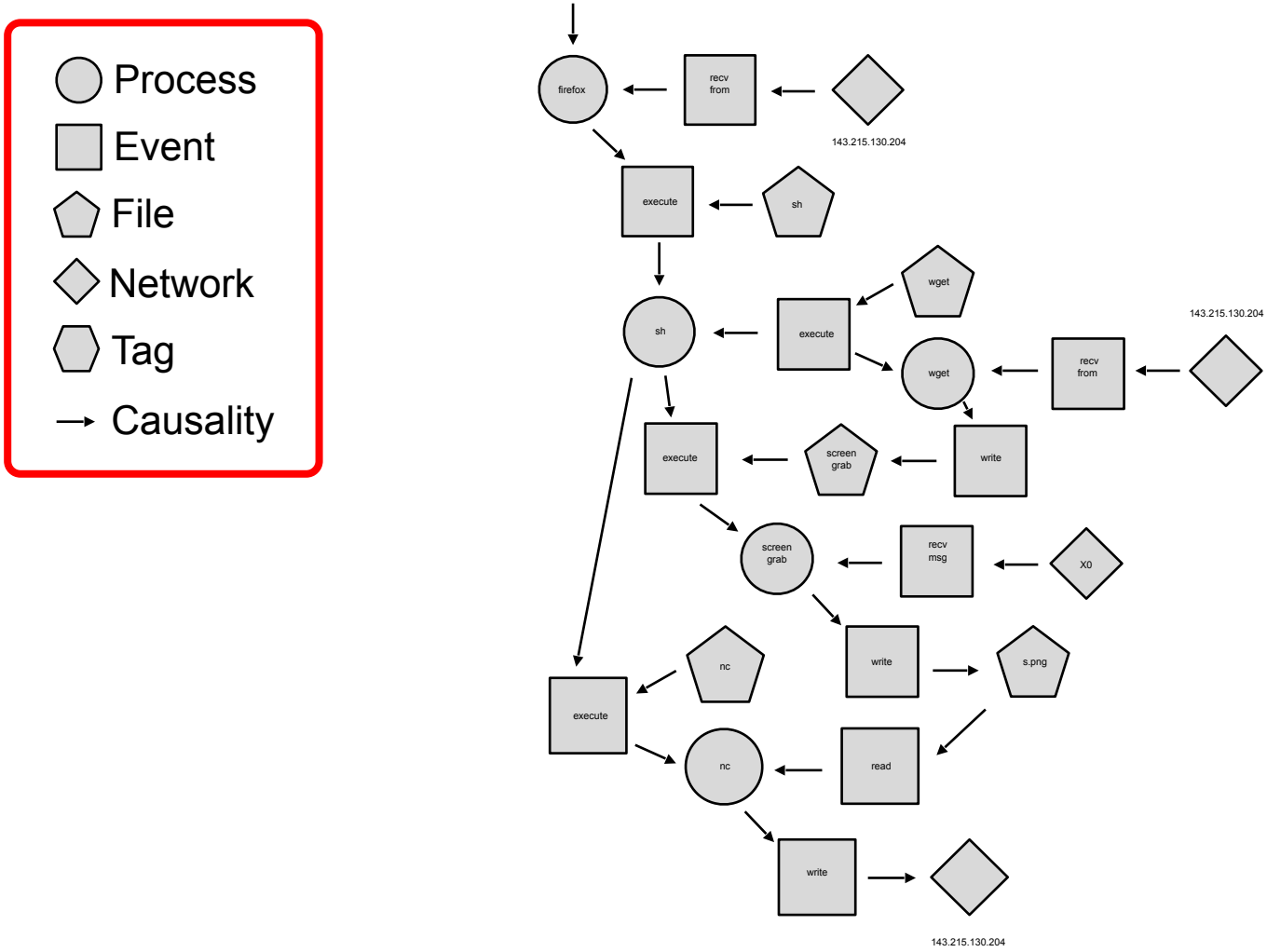


Case Study and AHG

- Process
- Event
- File
- Network
- Tag
- Causality

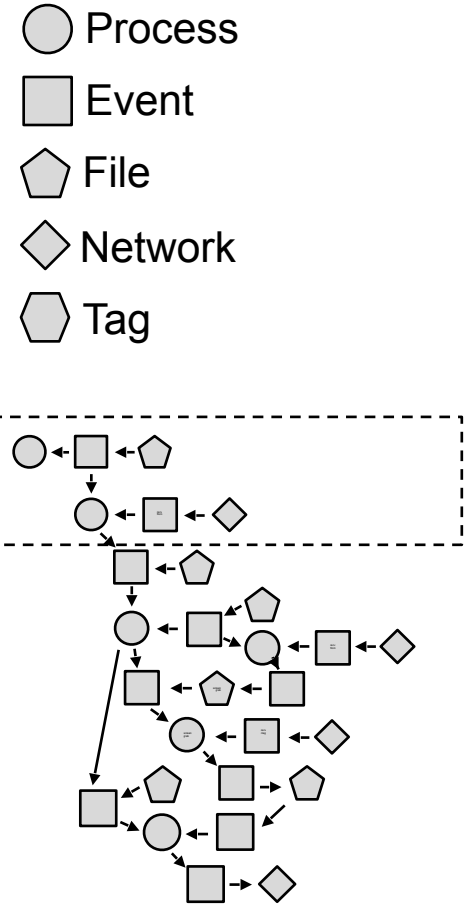


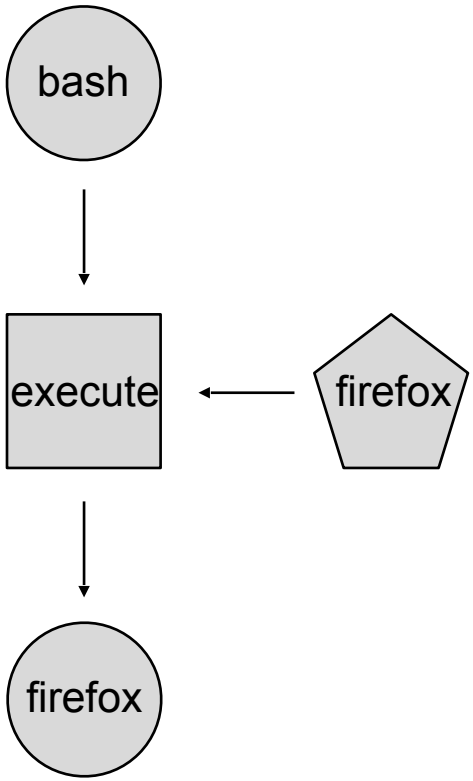
Case Study and AHG



Case Study and AHG Step 1

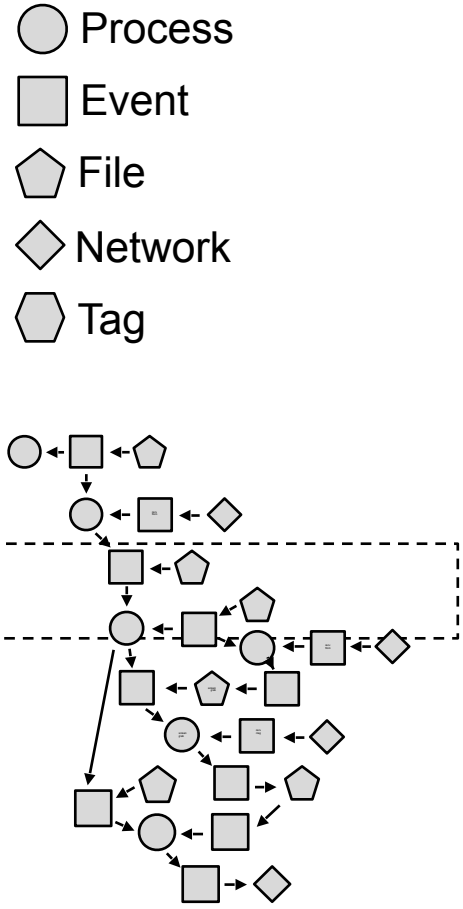
1) Victim starts Firefox

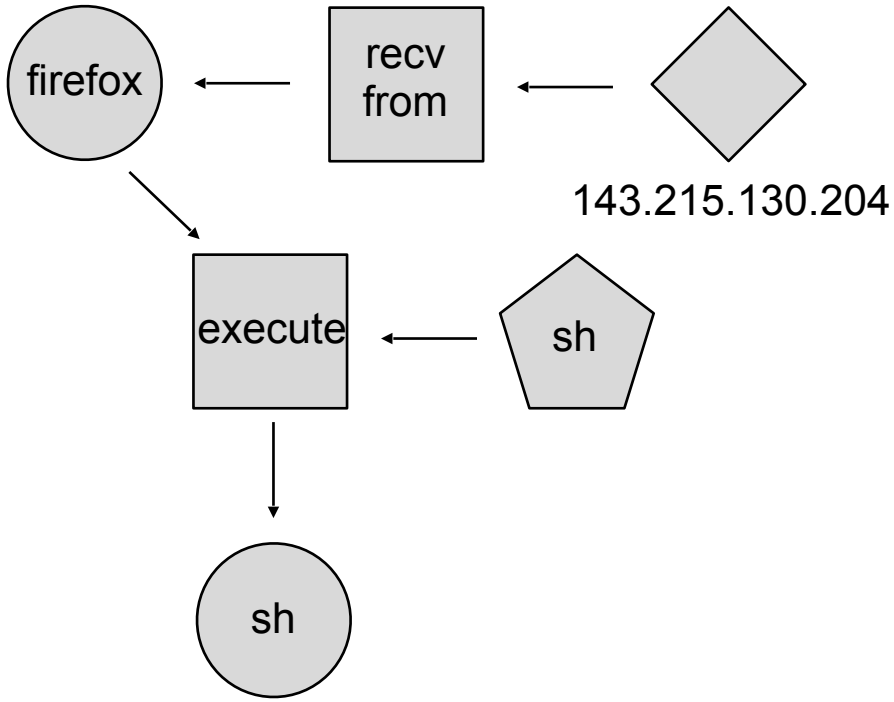




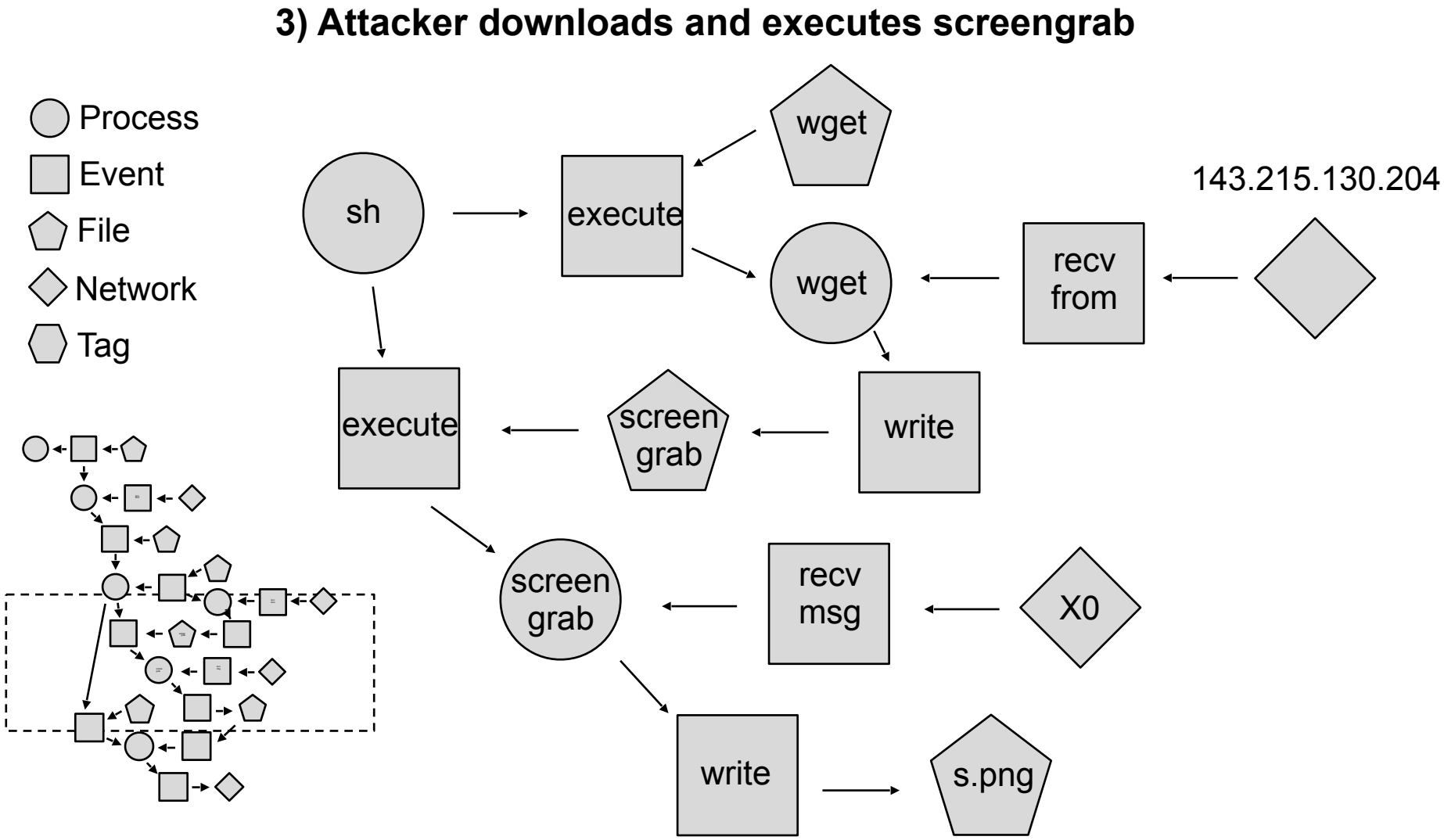
Case Study and AHG Step 2

2) Victim visits malicious.com (143.215.130.204) that runs shell process



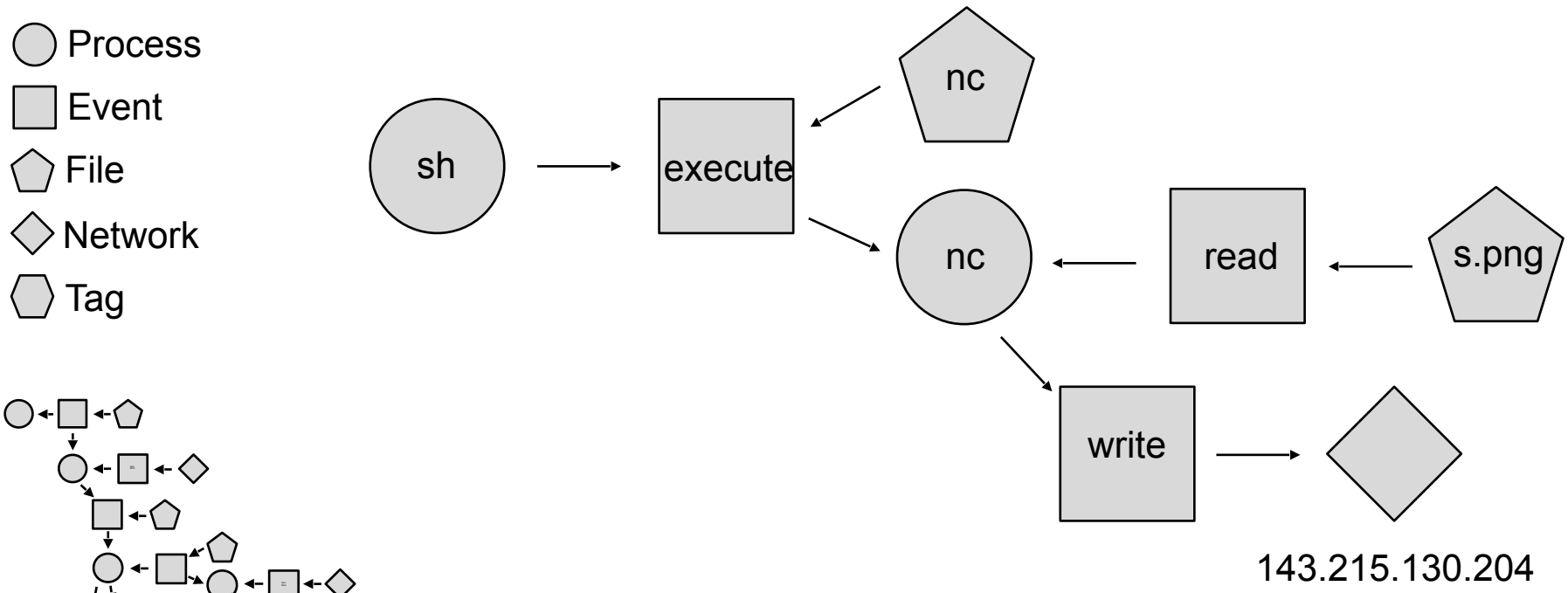


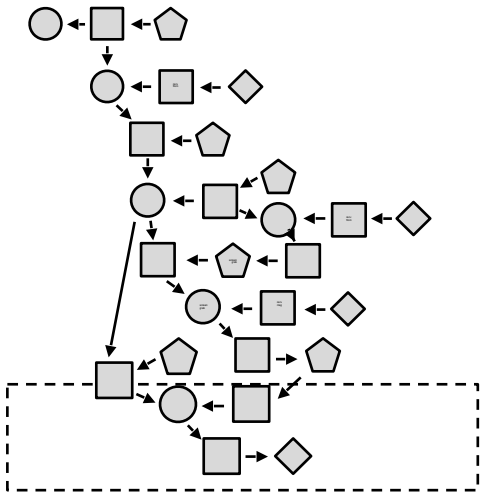
Case Study and AHG Step 3



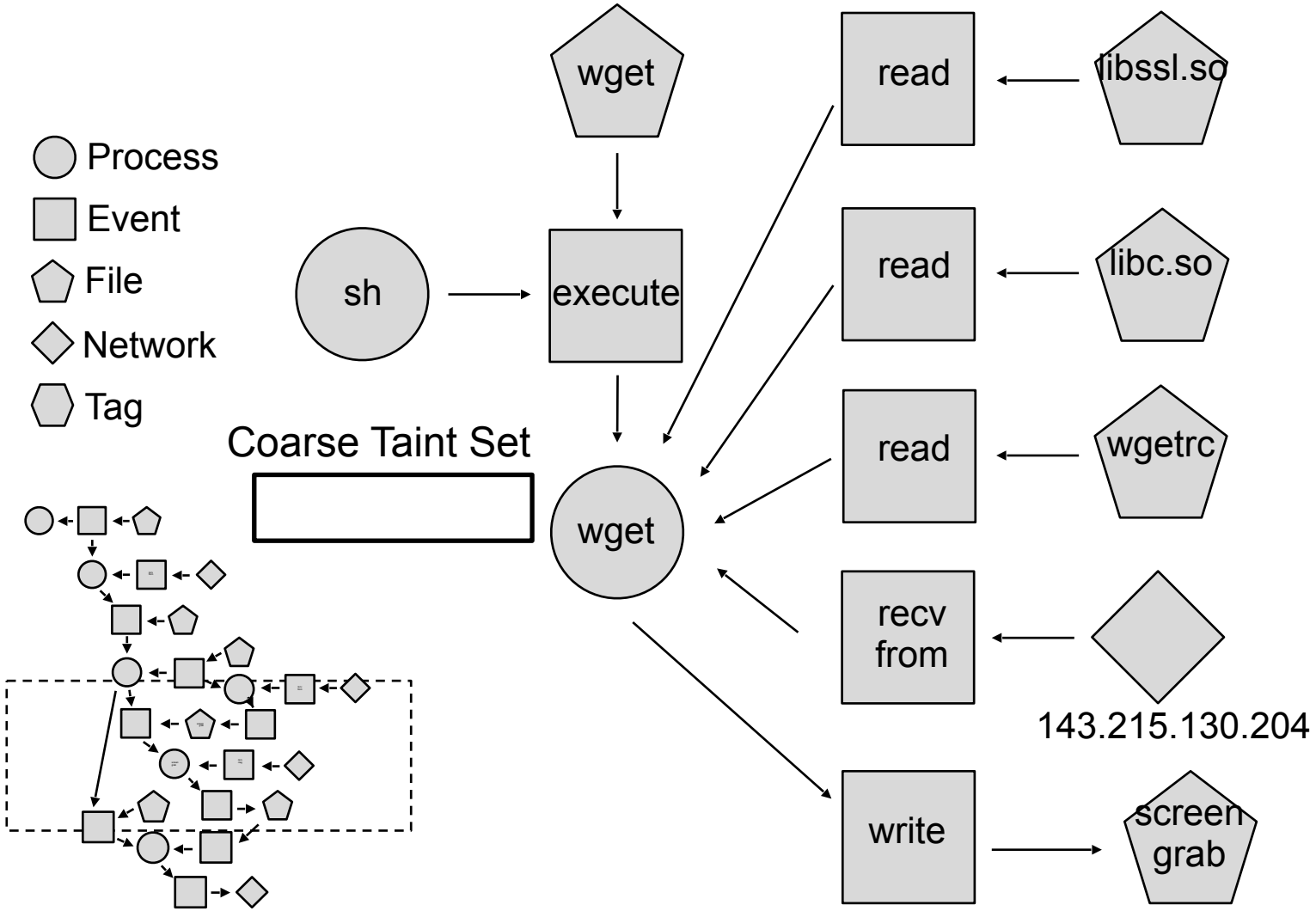
Case Study and AHG Step 4

4) Screenshot is sent to attacker's server

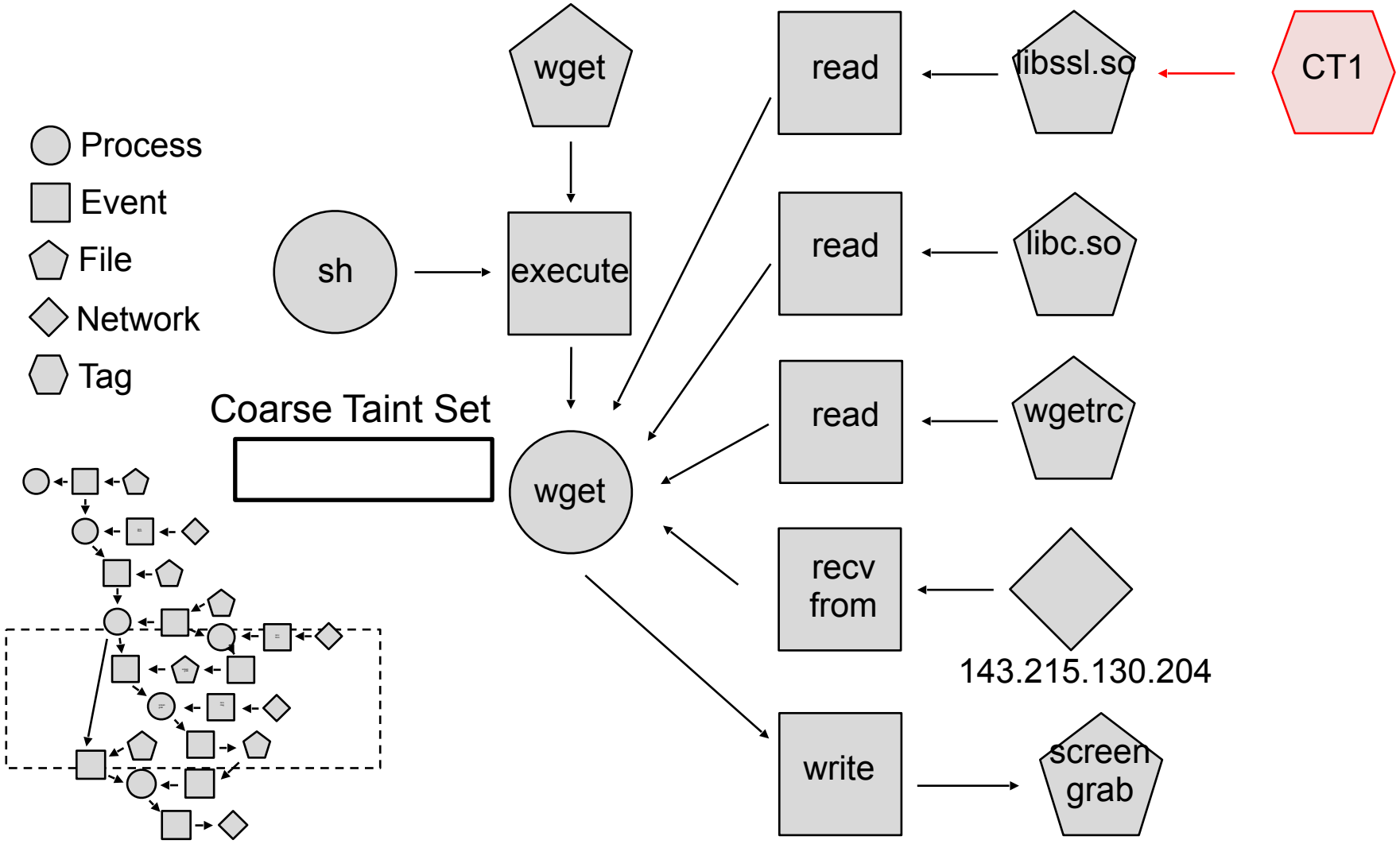




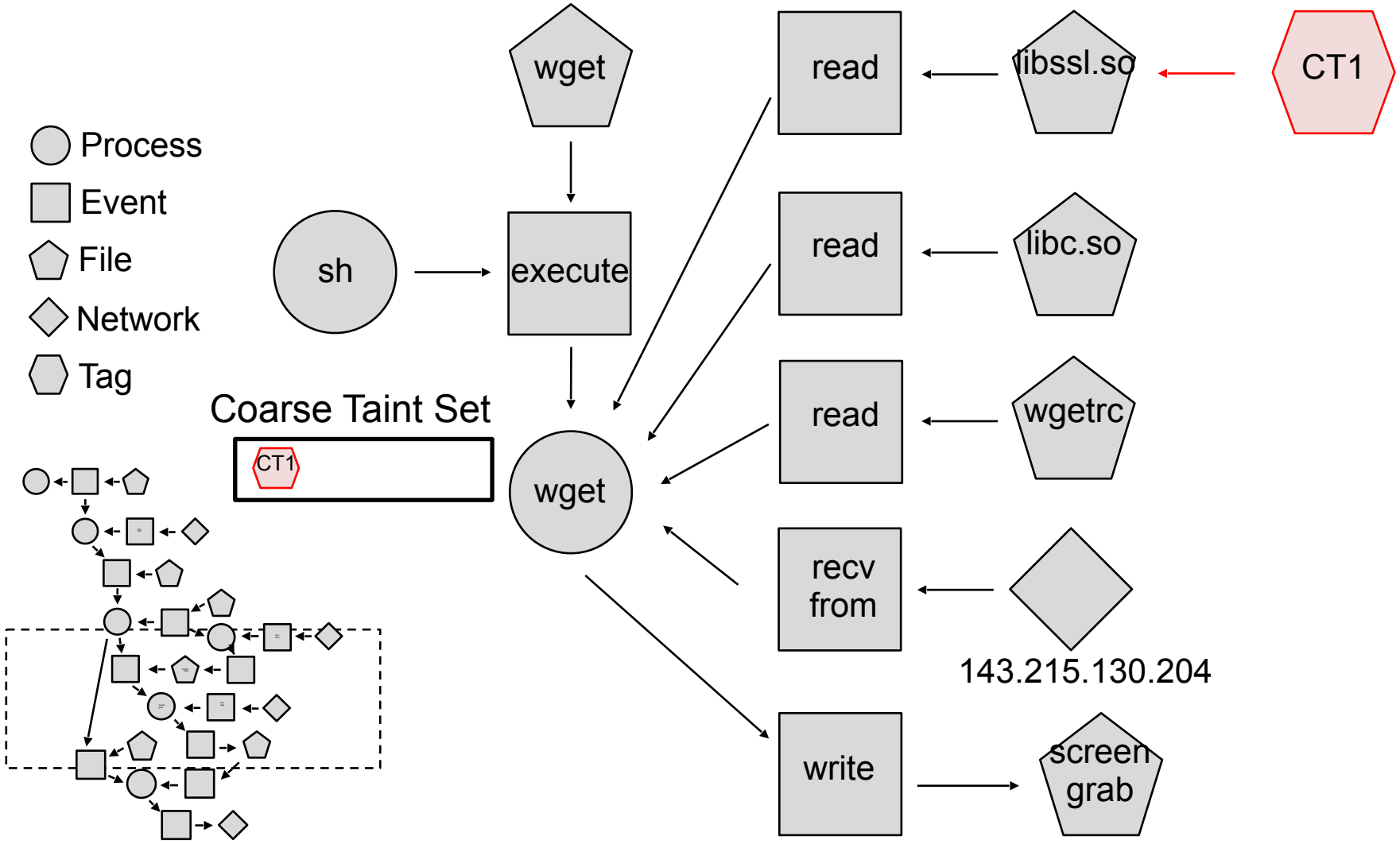
Case Study and Coarse-grained Taint Analysis.



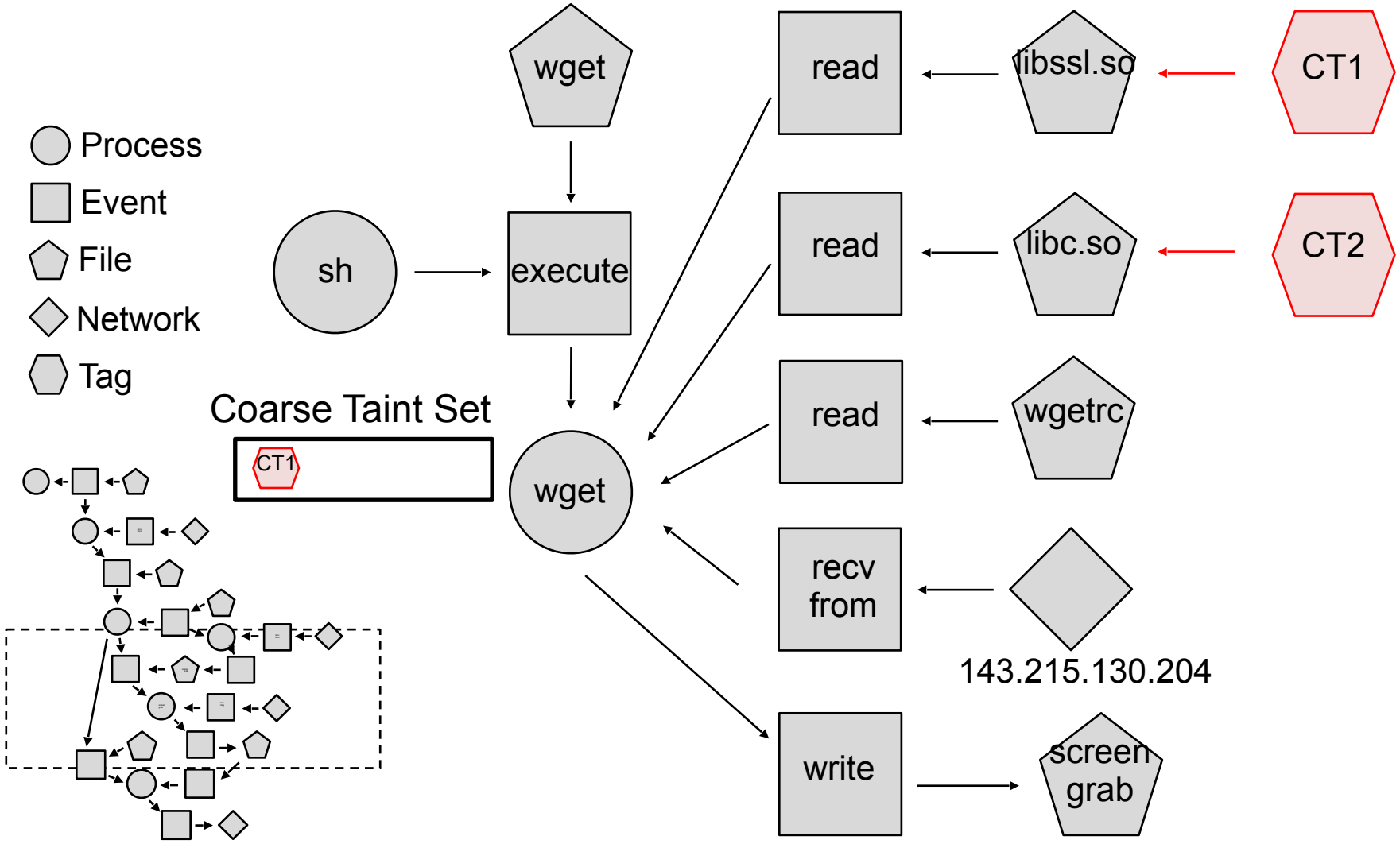
Case Study and Coarse-grained Taint Analysis.



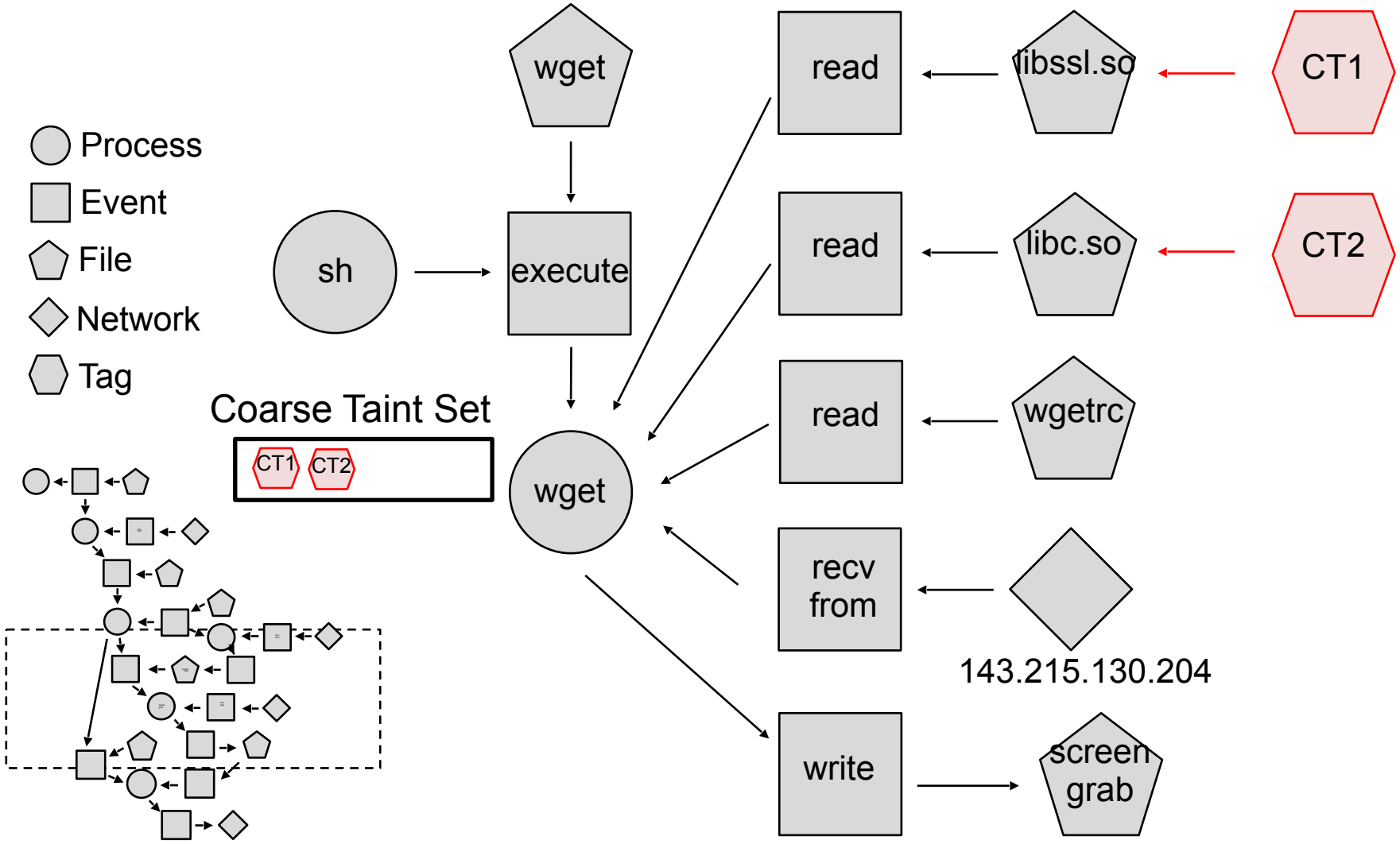
Case Study and Coarse-grained Taint Analysis.



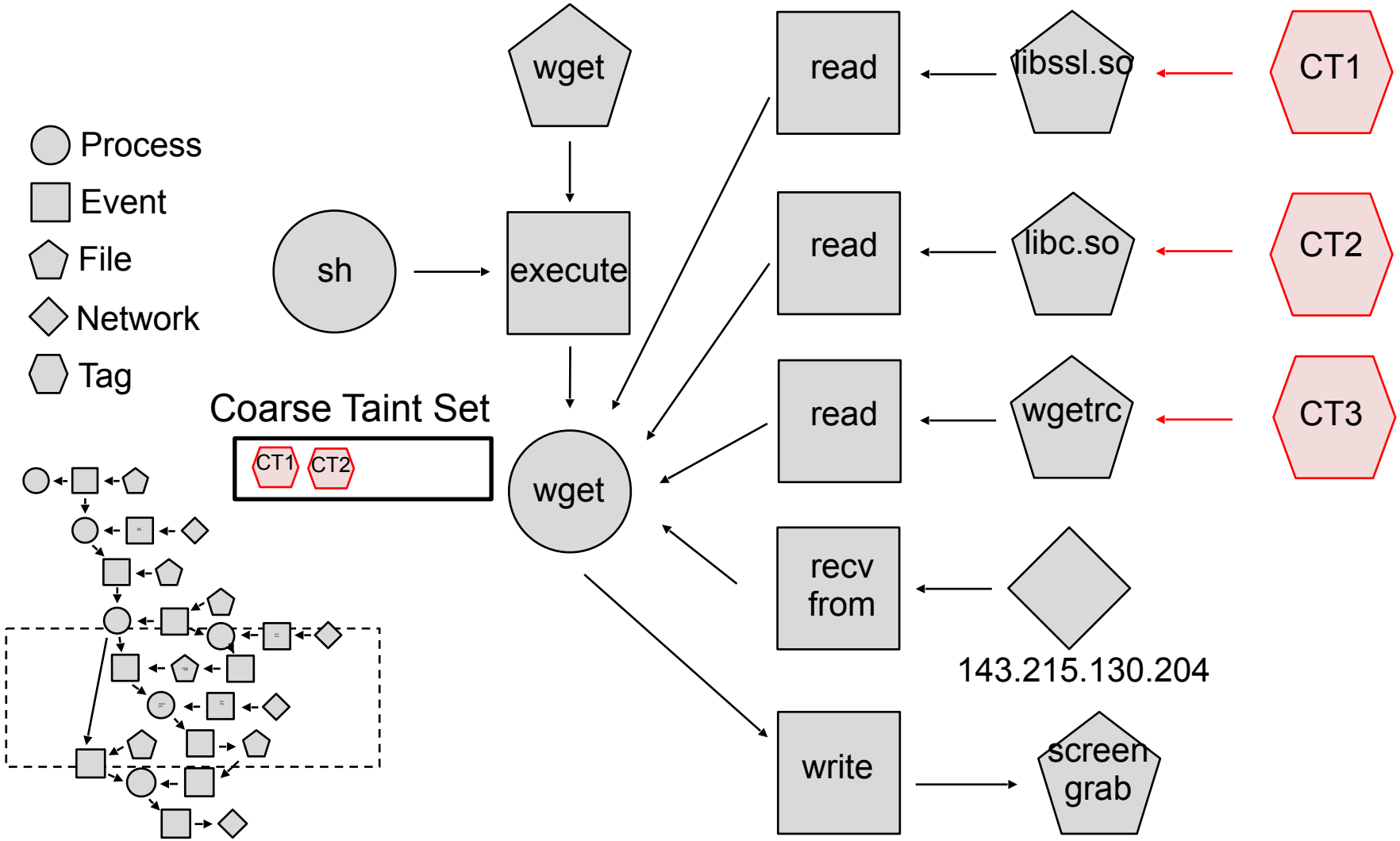
Case Study and Coarse-grained Taint Analysis.



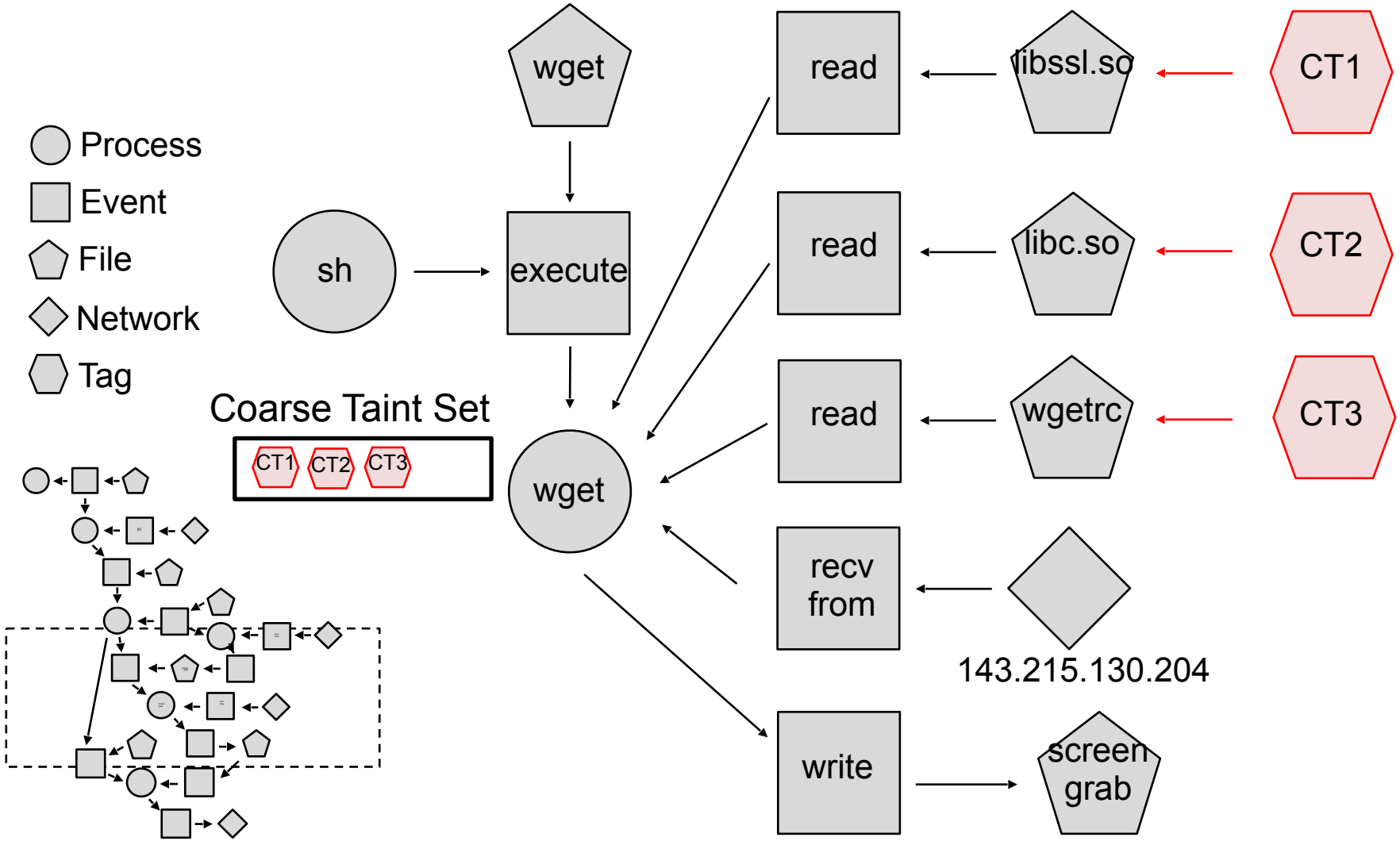
Case Study and Coarse-grained Taint Analysis.



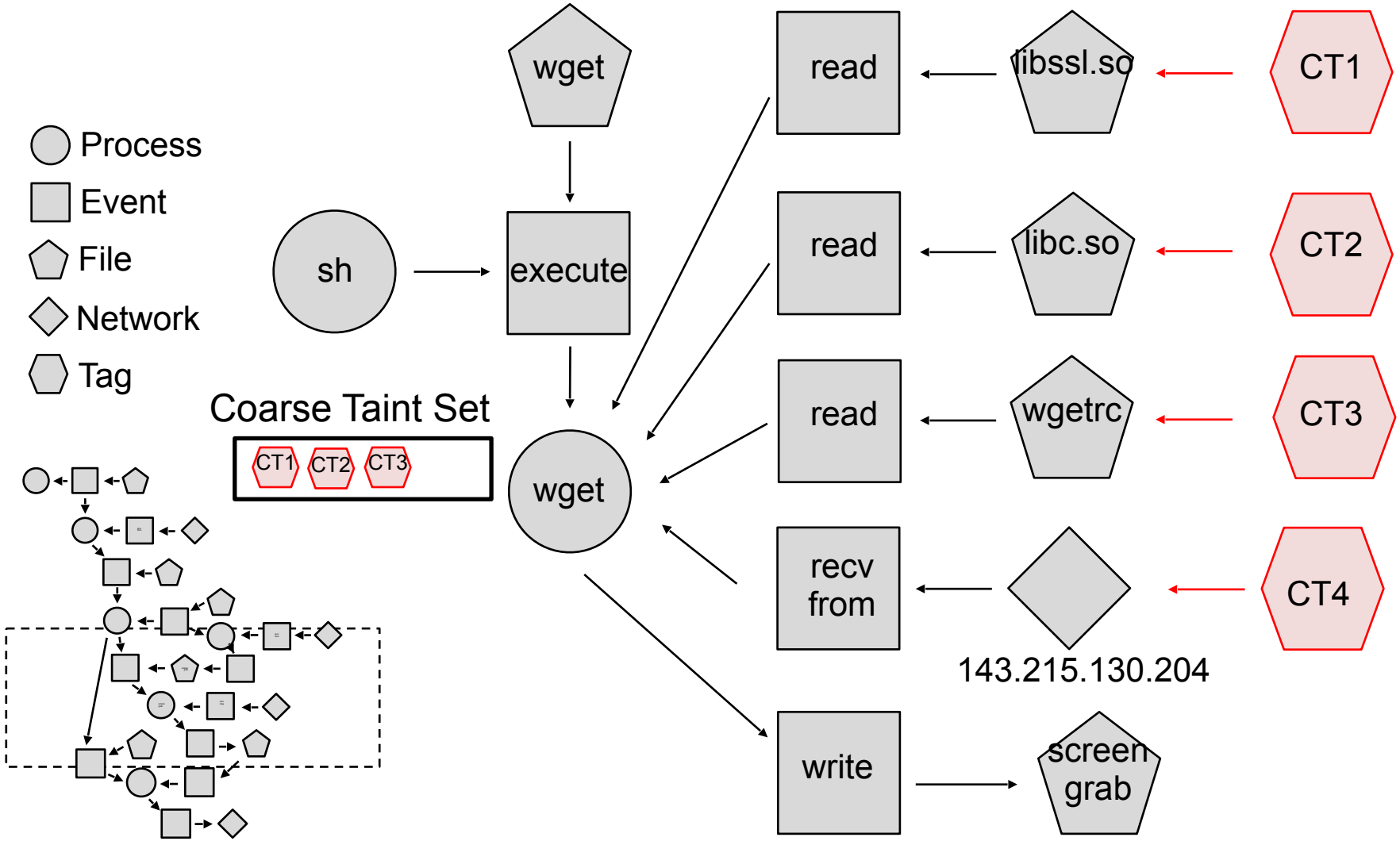
Case Study and Coarse-grained Taint Analysis.



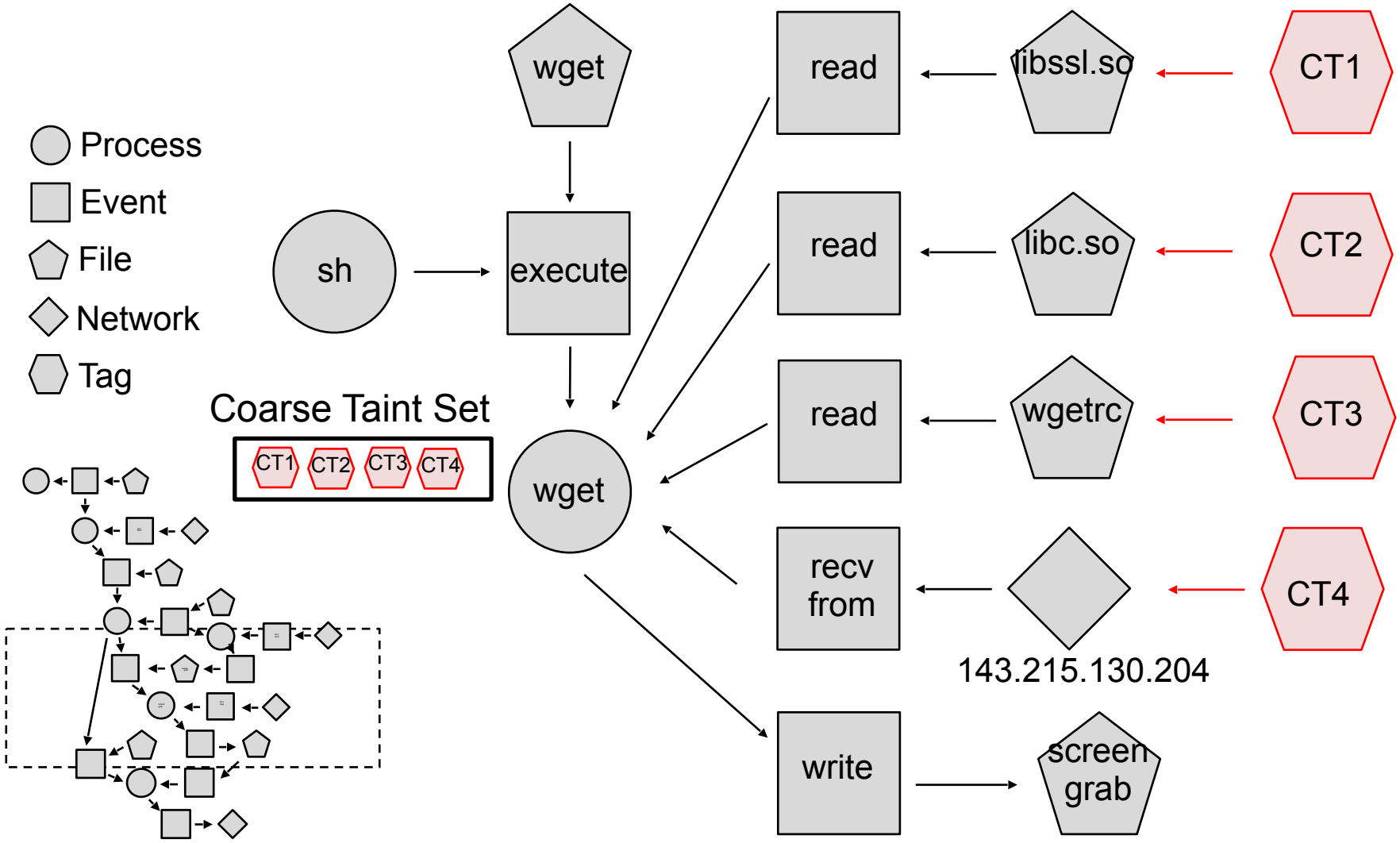
Case Study and Coarse-grained Taint Analysis.



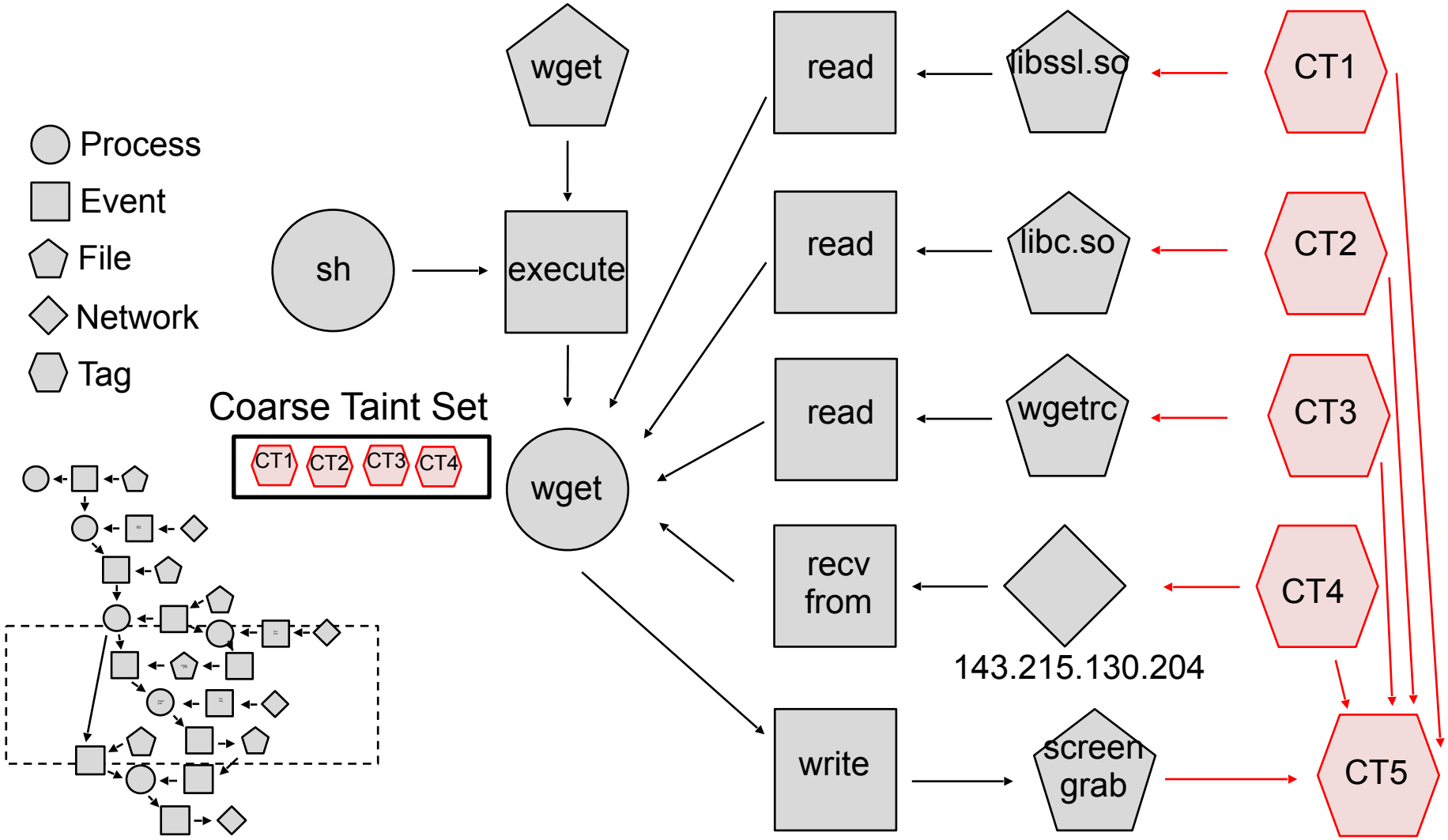
Case Study and Coarse-grained Taint Analysis.



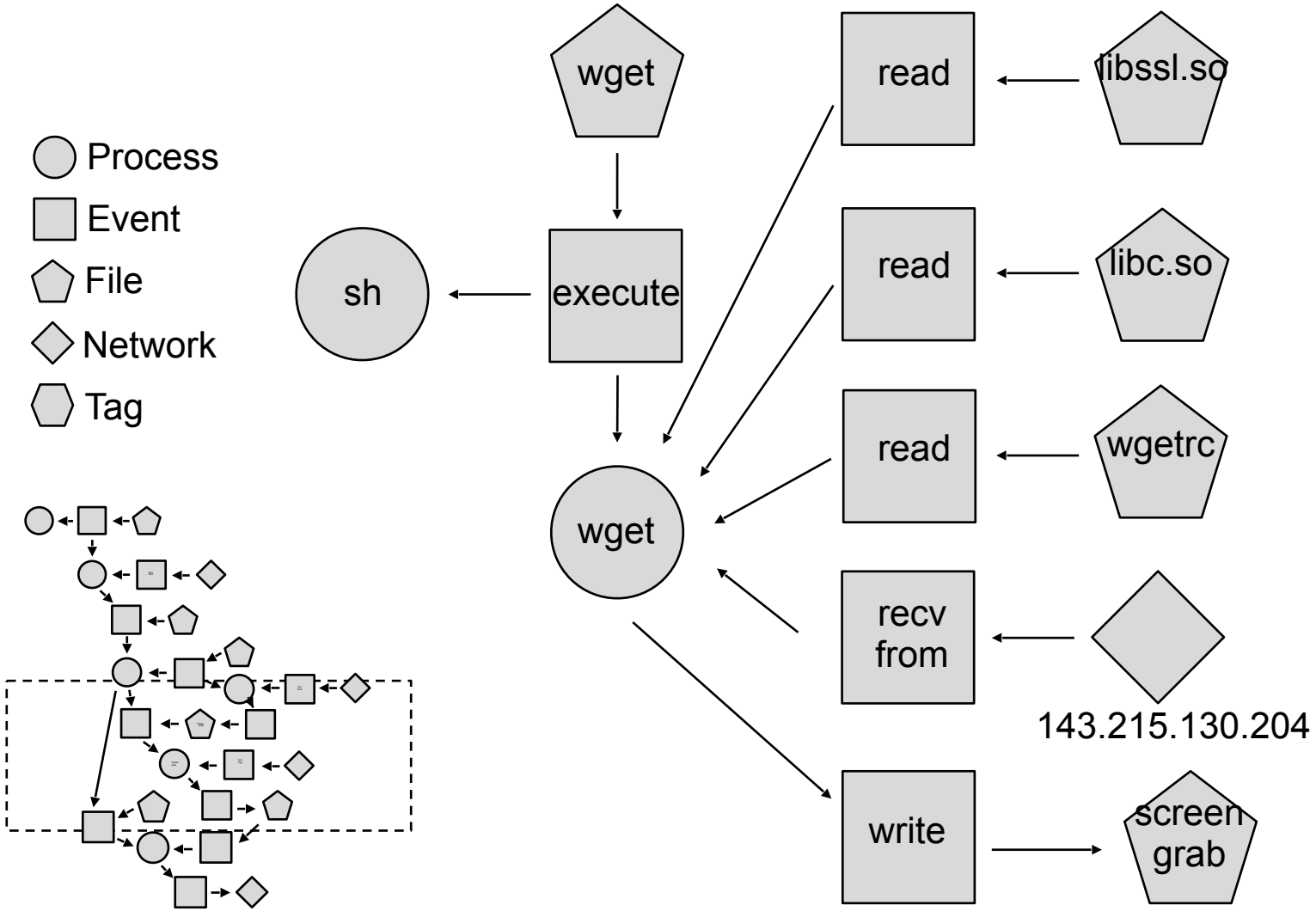
Case Study and Coarse-grained Taint Analysis.



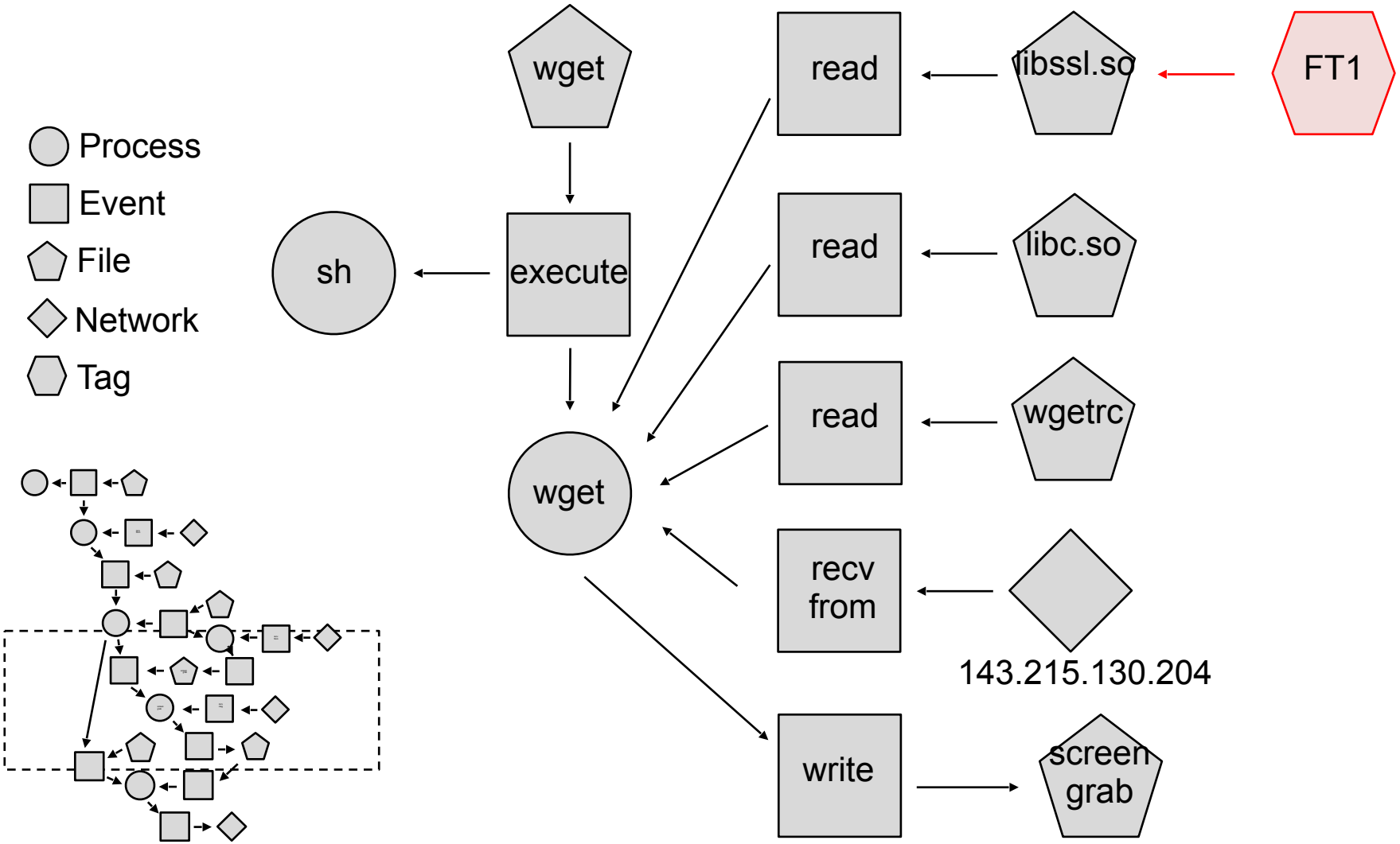
Case Study and Coarse-grained Taint Analysis.



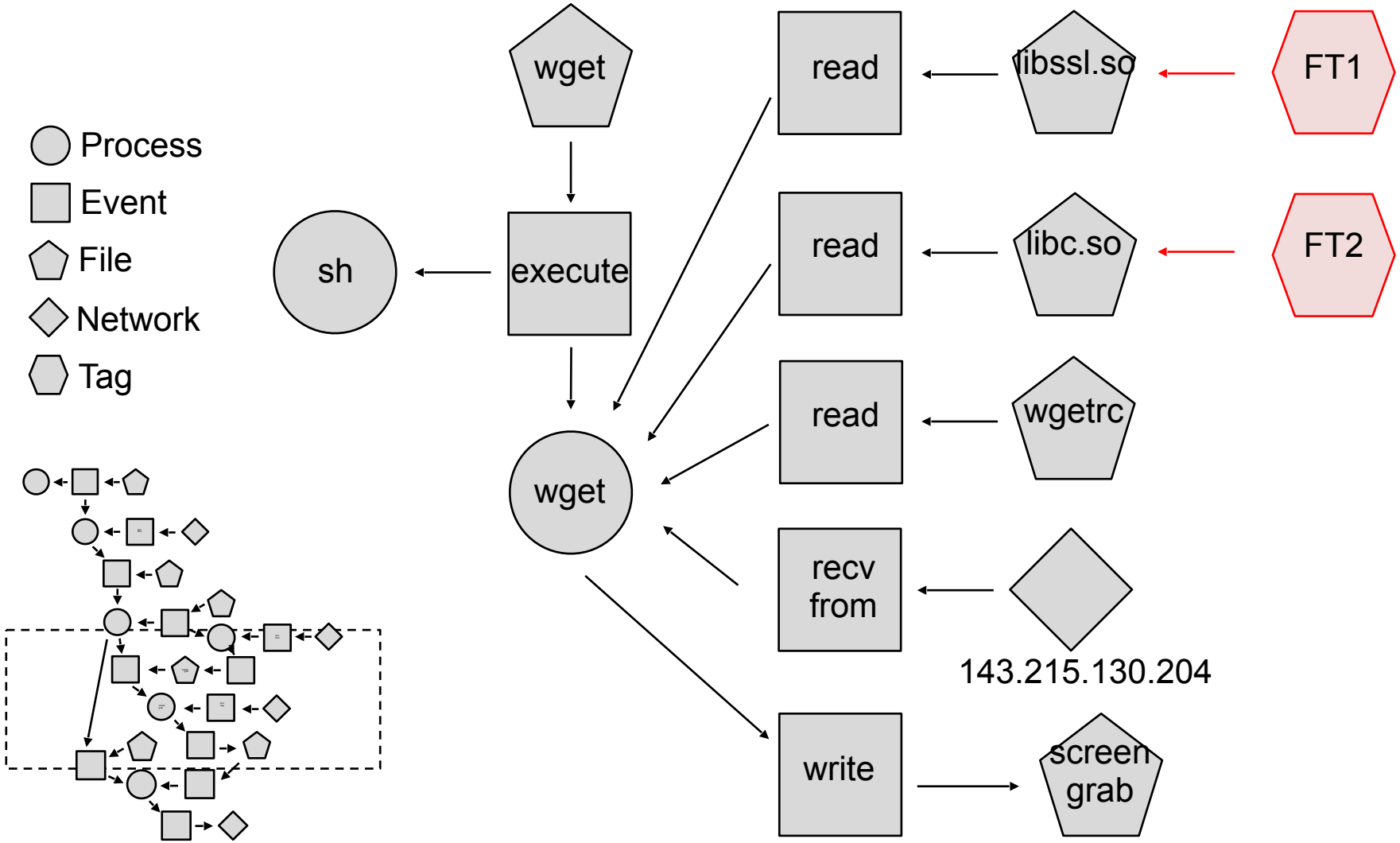
Case Study and Fine-grained Taint Analysis



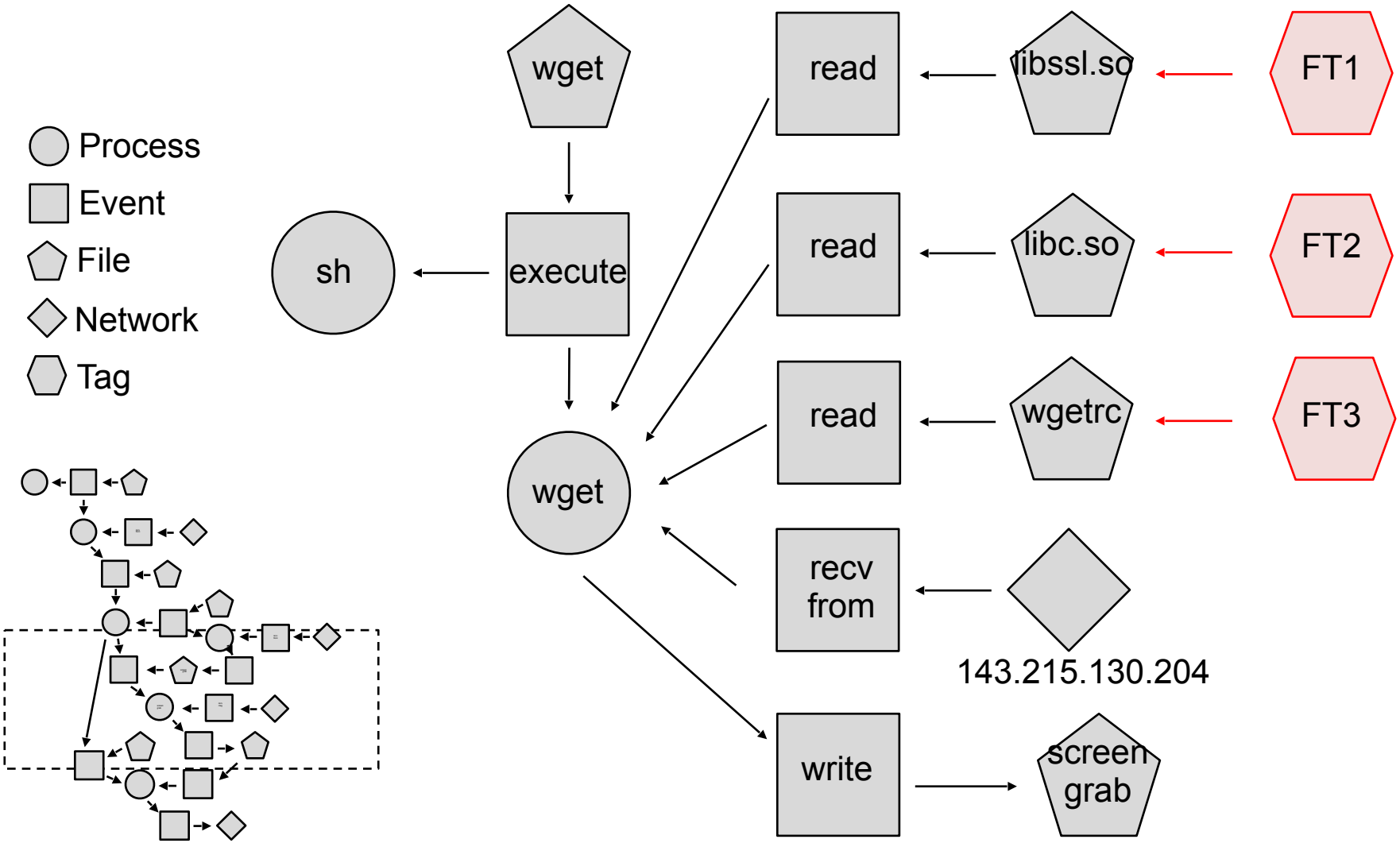
Case Study and Fine-grained Taint Analysis



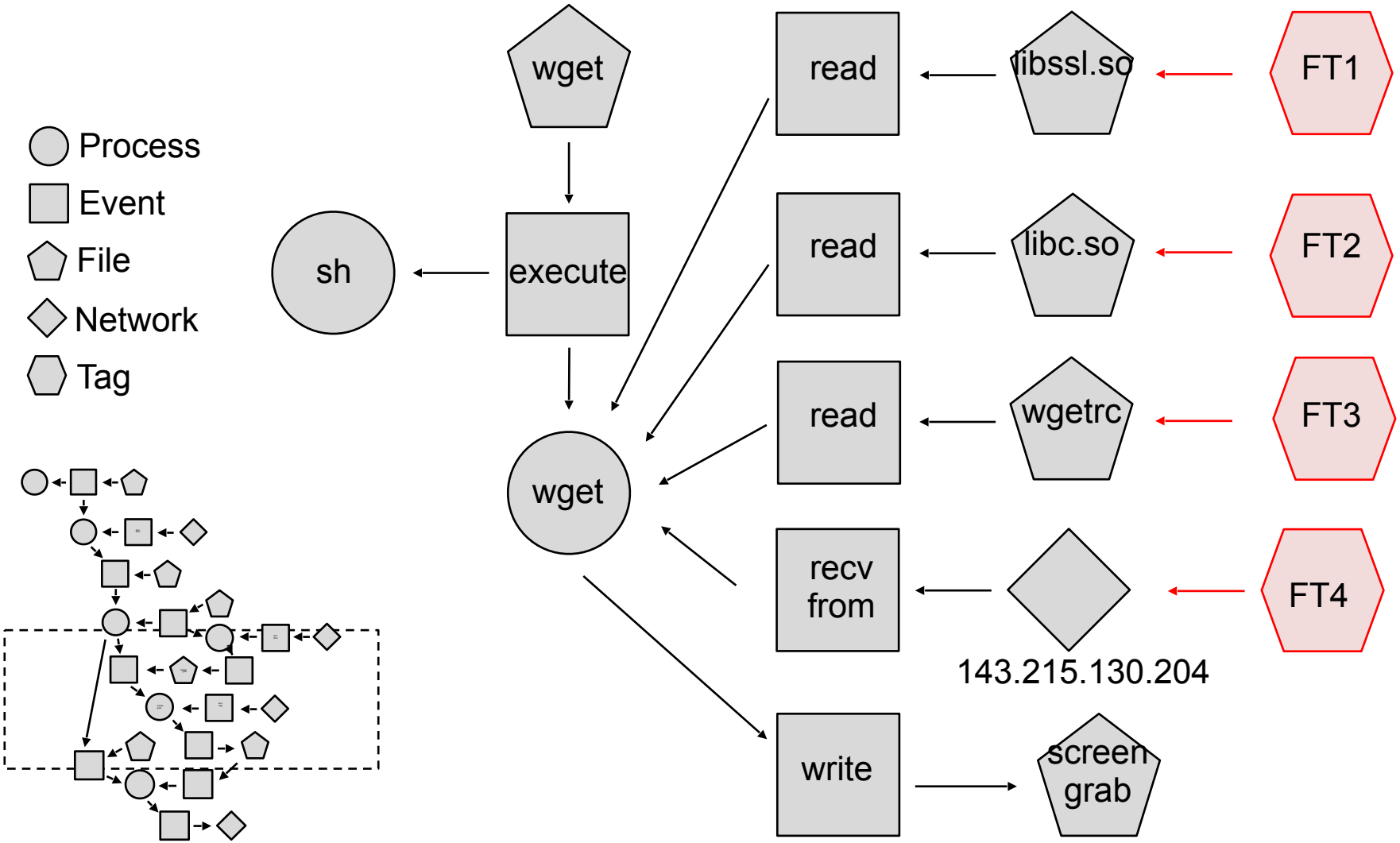
Case Study and Fine-grained Taint Analysis



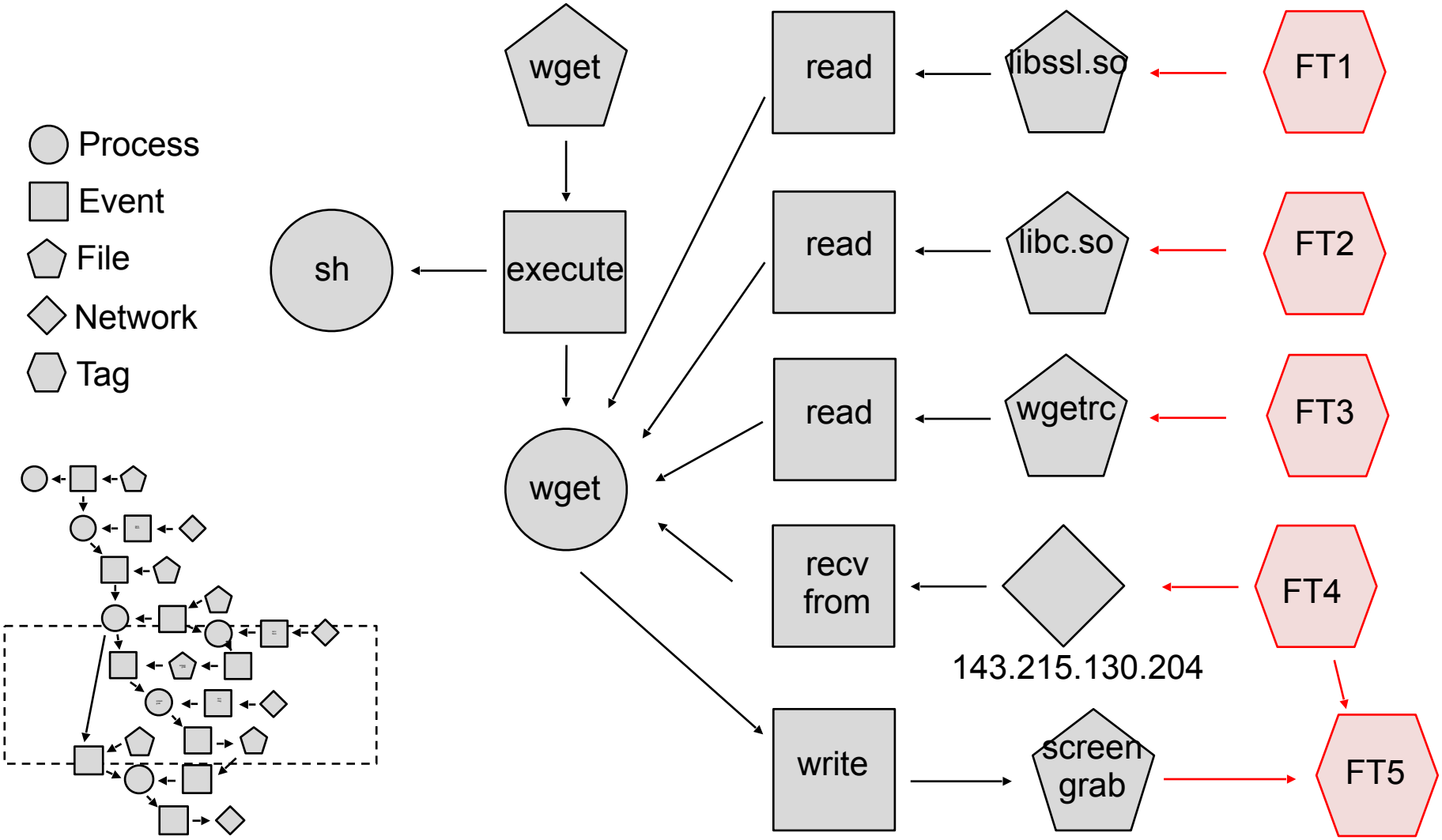
Case Study and Fine-grained Taint Analysis



Case Study and Fine-grained Taint Analysis



Case Study and Fine-grained Taint Analysis



THEIA-Panda Overheads

TIME	Bare Exec Time	KVM Exec Time	QEMU Exec Time	Record Exec Time	Replay Exec Time
Bare Exec Time					
KVM Exec Time	2.09 x				
QEMU Exec Time	6.19 x	2.96 x			
Record Exec Time	7.75 x	3.71 x	1.25 x		
Replay Exec Time	13.82 x	6.62 x	2.23 x	1.78 x	

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution
- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data

THEIA-Panda Overheads

TIME	Bare Exec Time	KVM Exec Time	QEMU Exec Time	Record Exec Time	Replay Exec Time
Bare Exec Time					
KVM Exec Time	2.09 x				
QEMU Exec Time	6.19 x	2.96 x			
Record Exec Time	7.75 x	3.71 x	1.25 x		
Replay Exec Time	13.82 x	6.62 x	2.23 x	1.78 x	

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution
- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data

THEIA-Panda Overheads

TIME	Bare Exec Time	KVM Exec Time	QEMU Exec Time	Record Exec Time	Replay Exec Time
Bare Exec Time					
KVM Exec Time	2.09 x				
QEMU Exec Time	6.19 x	2.96 x			
Record Exec Time	7.75 x	3.71 x	1.25 x		
Replay Exec Time	13.82 x	6.62 x	2.23 x	1.78 x	

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution
- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data

THEIA-Panda Overheads

TIME	Bare Exec Time	KVM Exec Time	QEMU Exec Time	Record Exec Time	Replay Exec Time
Bare Exec Time					
KVM Exec Time	2.09 x				
QEMU Exec Time	6.19 x	2.96 x			
Record Exec Time	7.75 x	3.71 x	1.25 x		
Replay Exec Time	13.82 x	6.62 x	2.23 x	1.78 x	

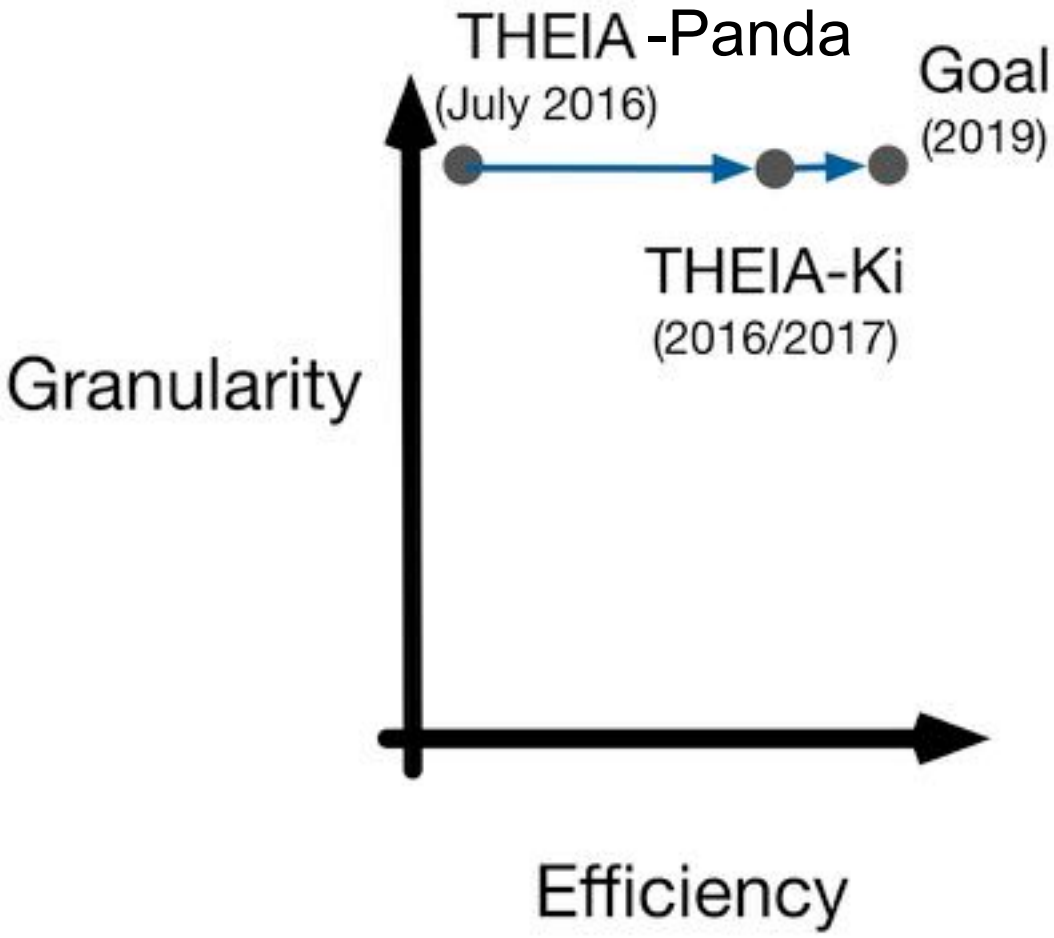
- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution
- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data

THEIA-Panda Overheads

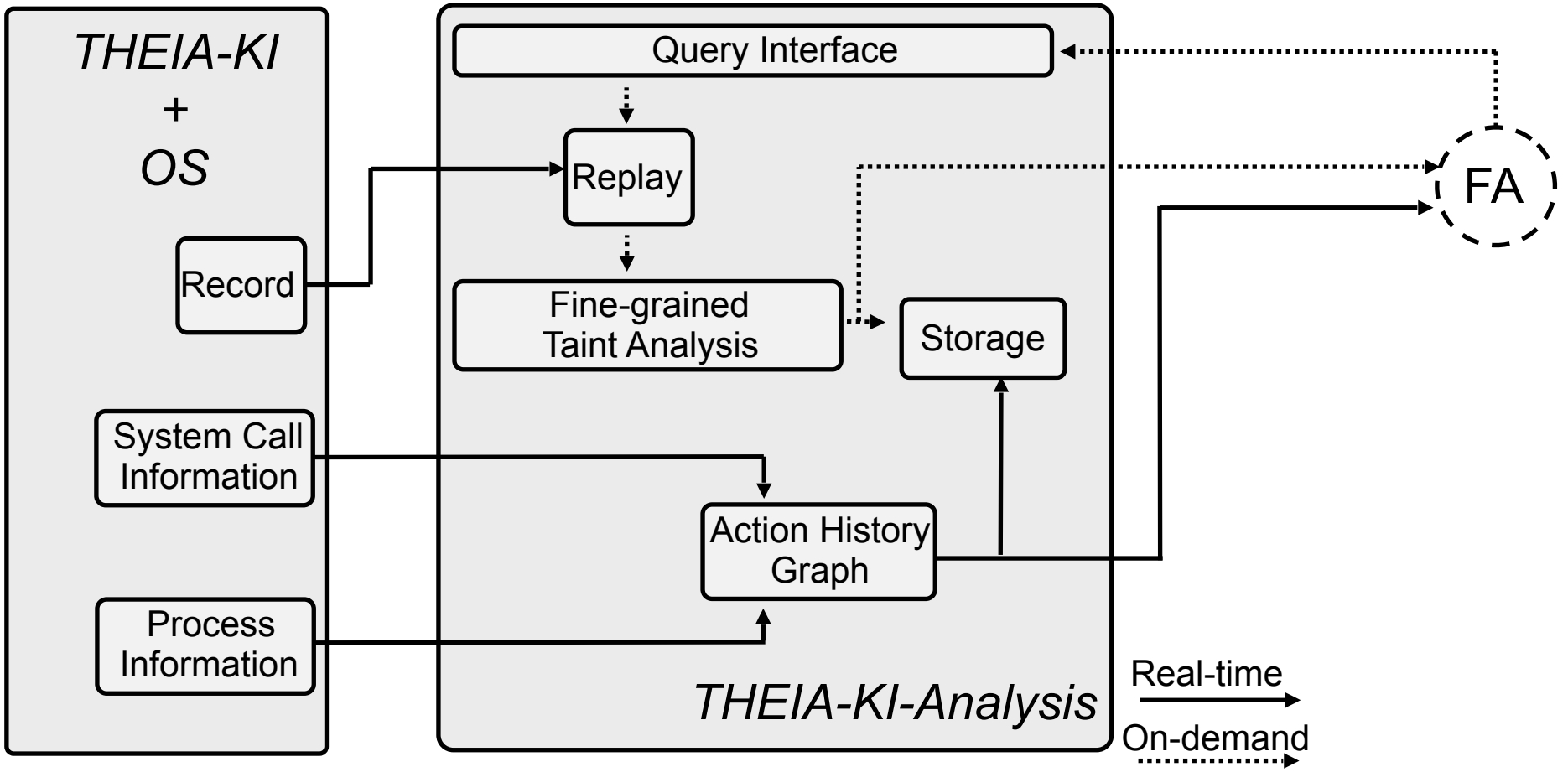
TIME	Bare Exec Time	KVM Exec Time	QEMU Exec Time	Record Exec Time	Replay Exec Time
Bare Exec Time					
KVM Exec Time	2.09 x				
QEMU Exec Time	6.19 x	2.96 x			
Record Exec Time	7.75 x	3.71 x	1.25 x		
Replay Exec Time	13.82 x	6.62 x	2.23 x	1.78 x	

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution
- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data

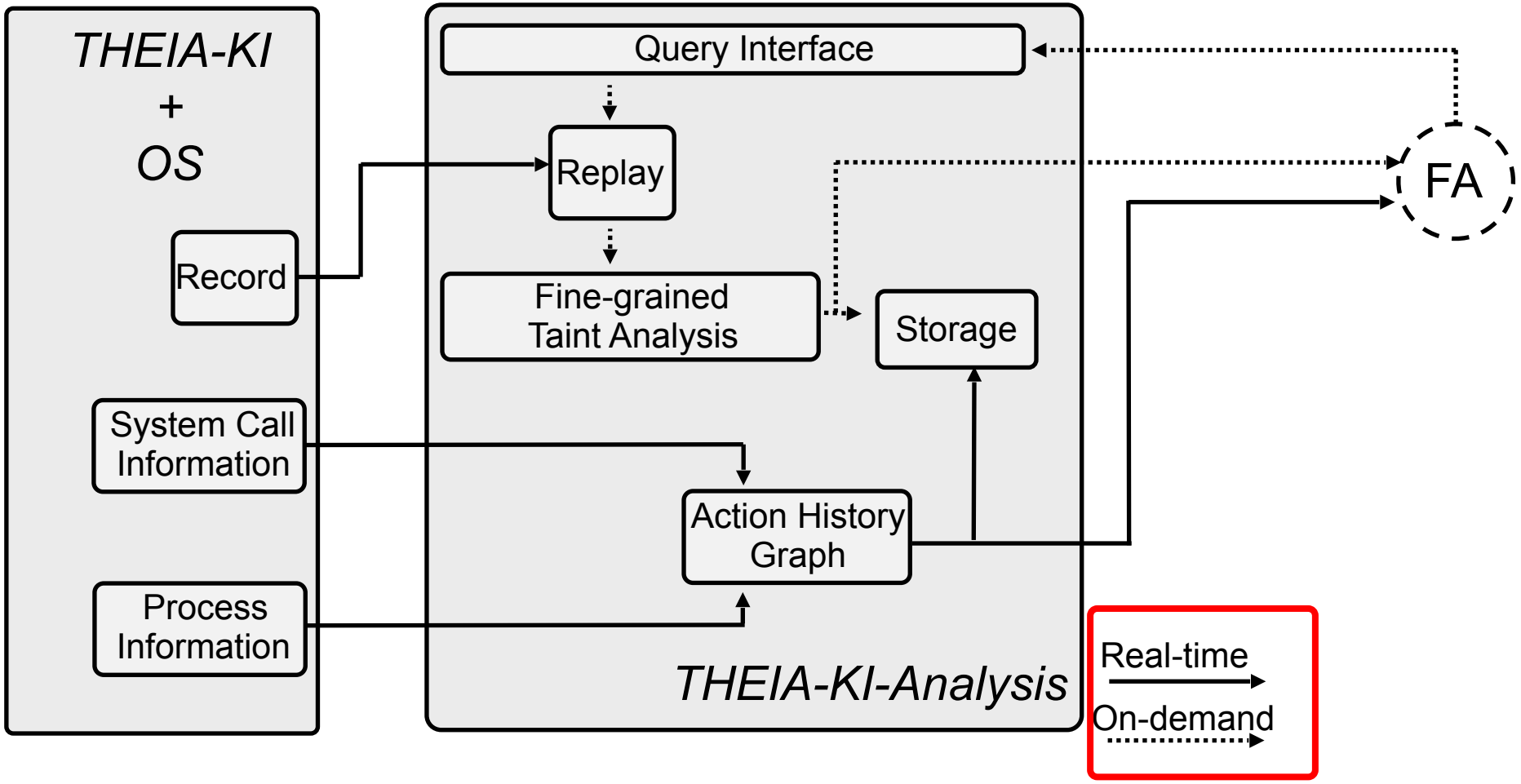
THEIA-Panda Observations



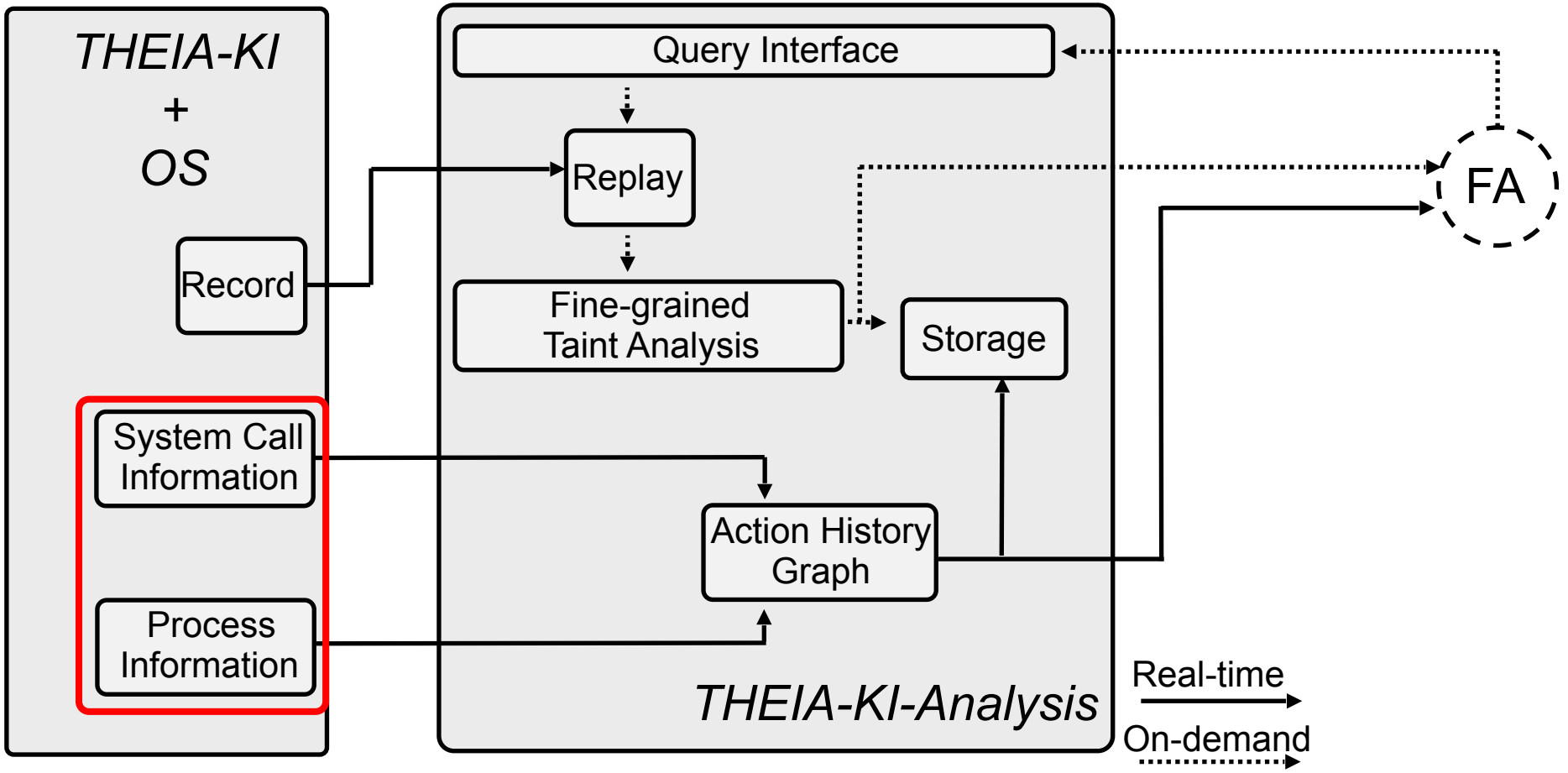
THEIA-KI Overview



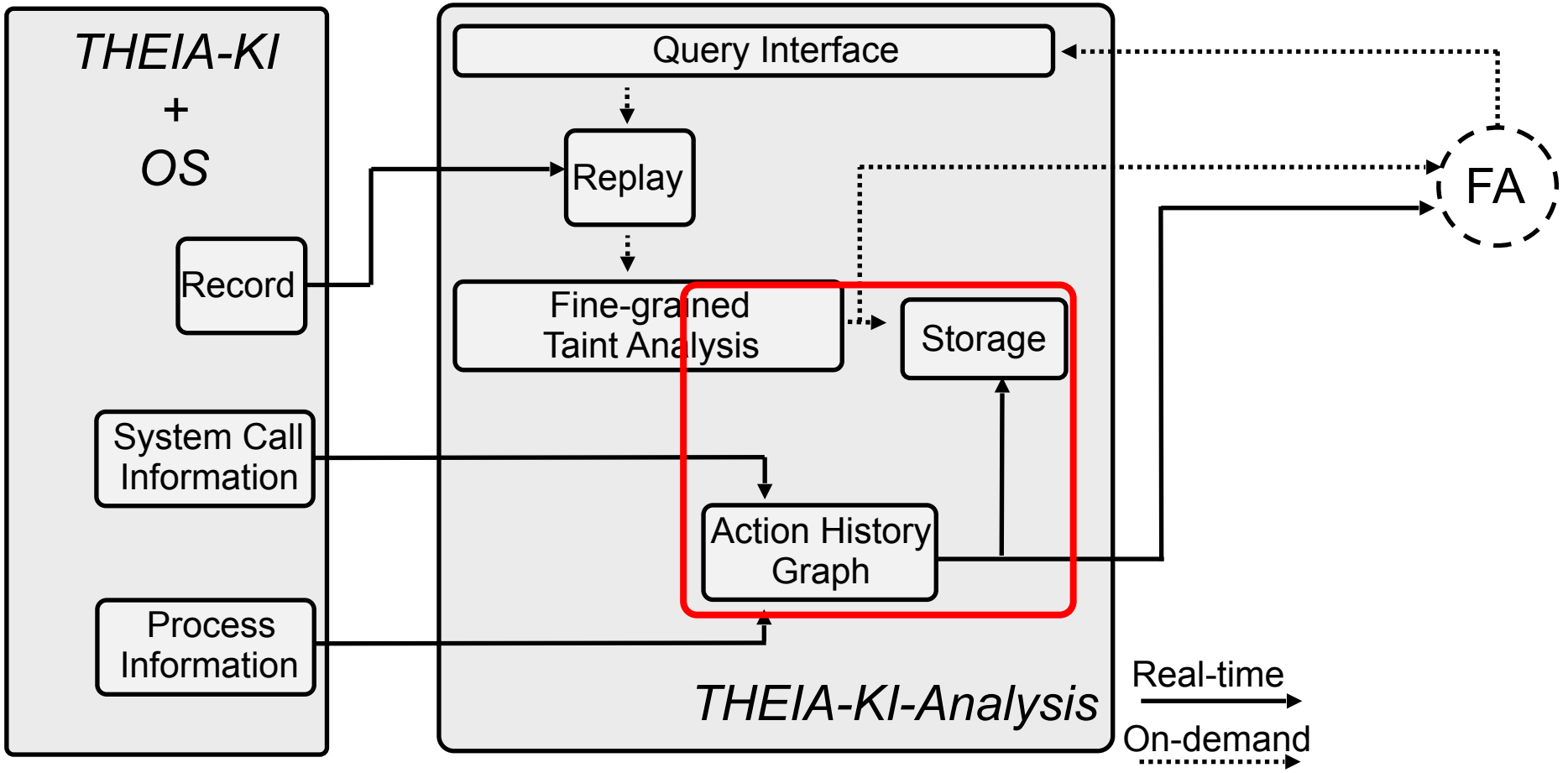
THEIA-KI Overview



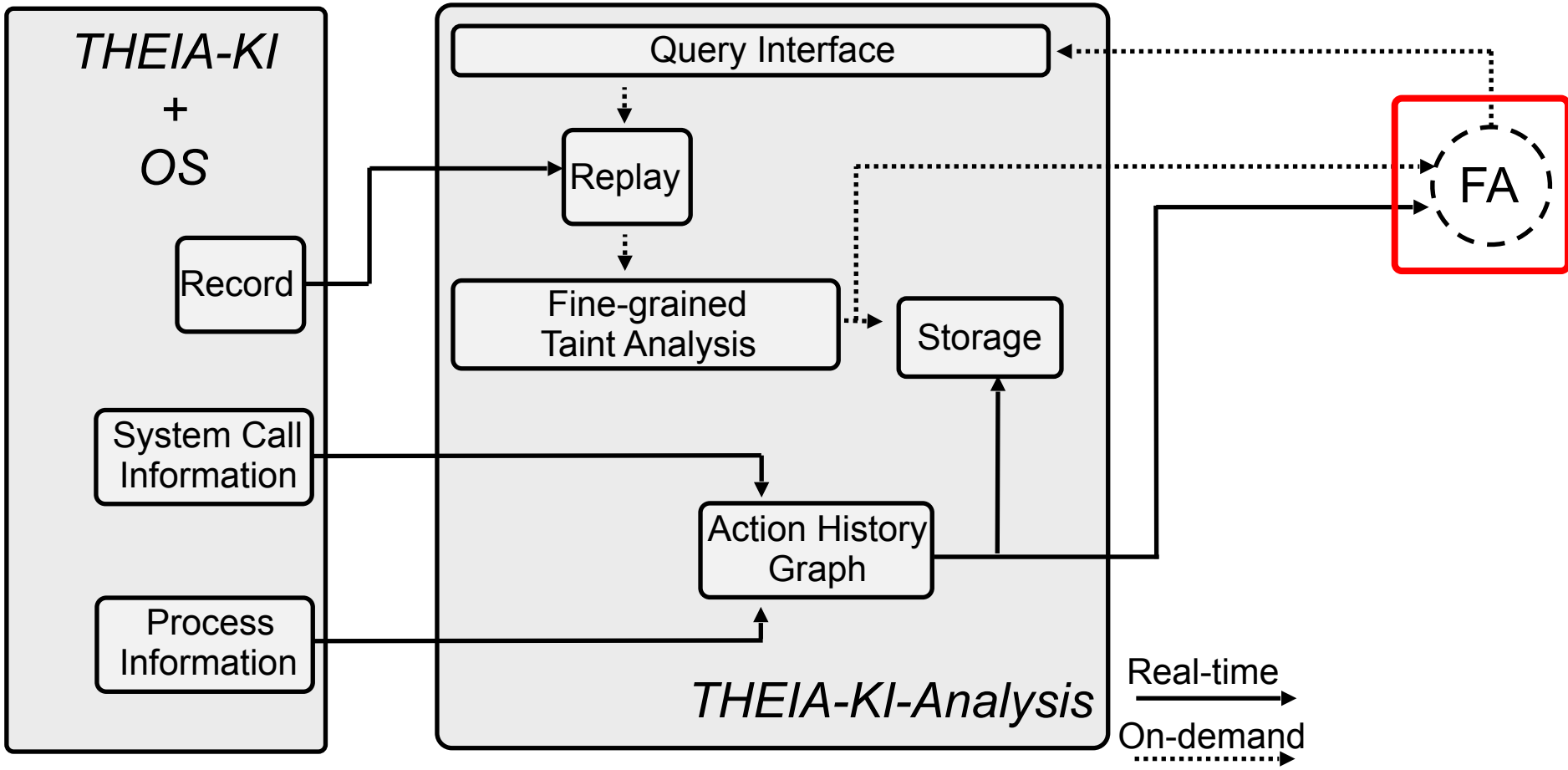
THEIA-KI Overview



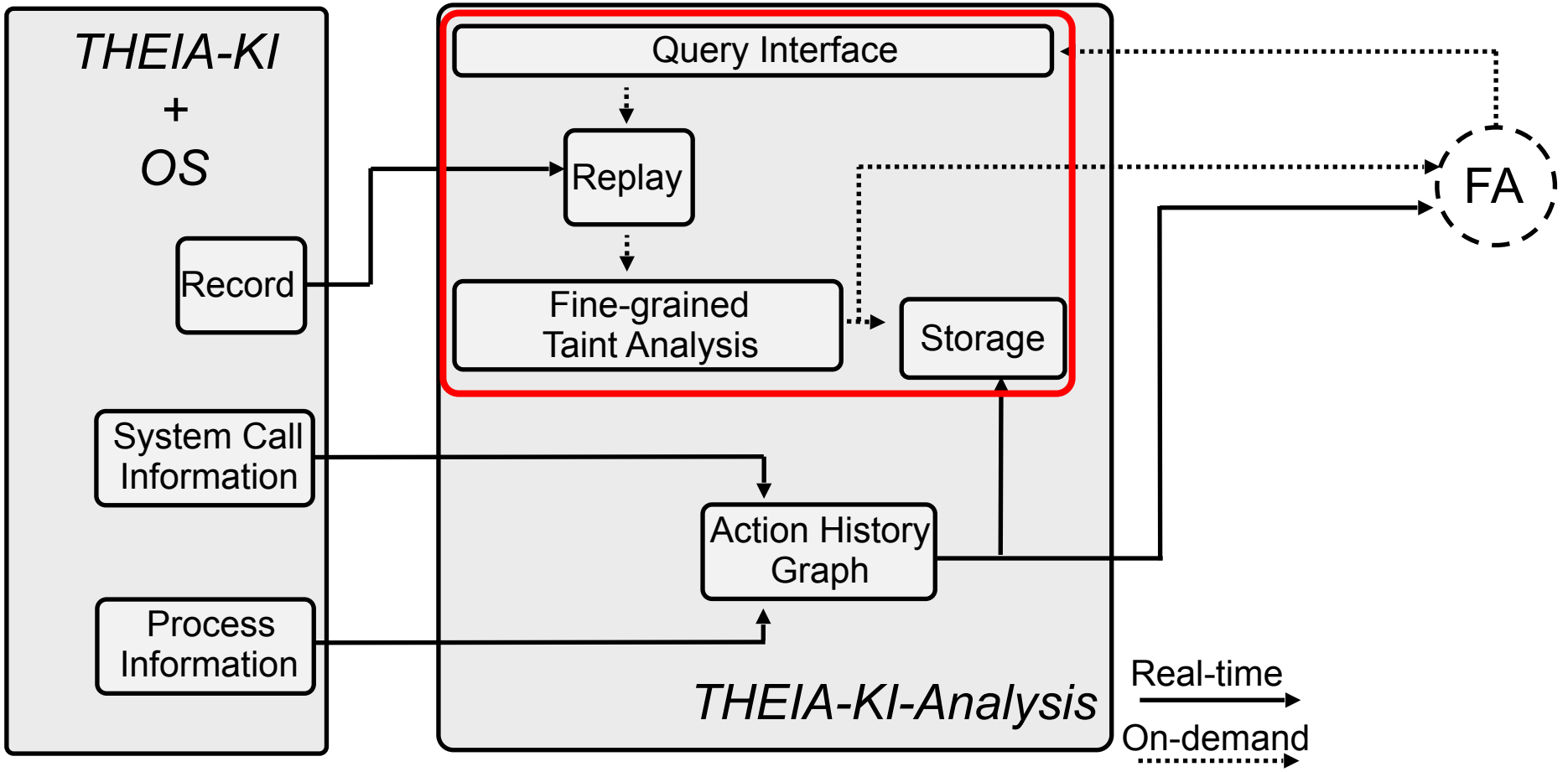
THEIA-KI Overview



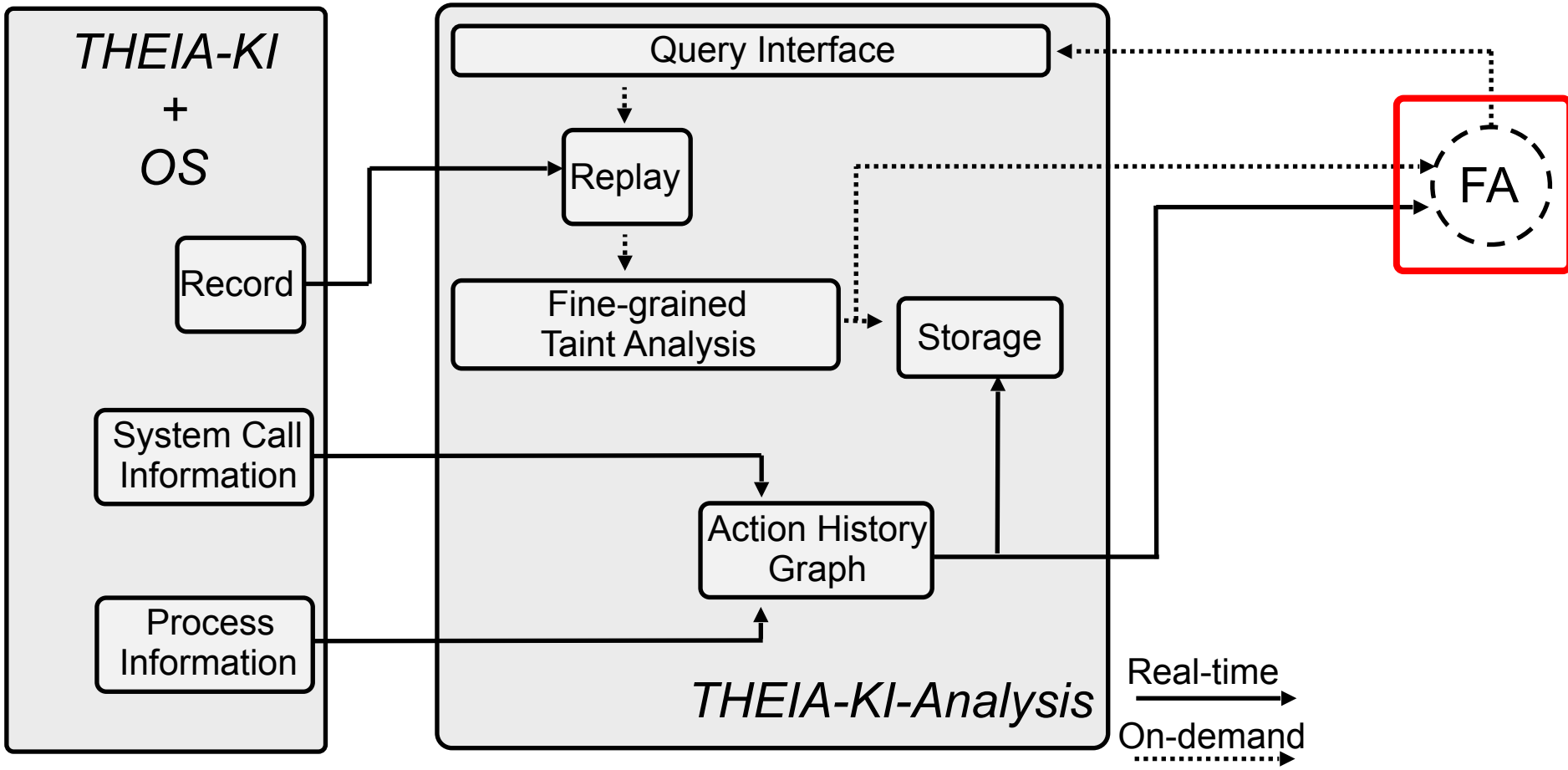
THEIA-KI Overview



THEIA-KI Overview



THEIA-KI Overview



THEIA-KI

- **Key features:**
  - Record/replay
    - Kernel-based instrumentation
  - Instruction level replay of the user space
    - On top of Intel PIN
  - Coarse-grained causality
    - From system instrumentation and logging
  - Fine-grained causality
    - From dynamic taint tracking
- **Threat model:**
  - Kernel is trusted

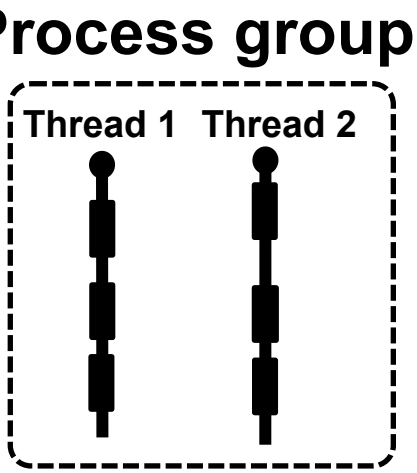
THEIA-KI

- **Key features:**
  - Record/replay
    - Kernel-based instrumentation
  - Instruction level replay of the user space
    - On top of Intel PIN
  - Coarse-grained causality
    - From system instrumentation and logging
  - Fine-grained causality
    - From dynamic taint tracking
- **Threat model:**
  - Kernel is trusted

THEIA-KI

- **Key features:**
  - Record/replay
    - Kernel-based instrumentation
  - Instruction level replay of the user space
    - On top of Intel PIN
  - Coarse-grained causality
    - From system instrumentation and logging
  - Fine-grained causality
    - From dynamic taint tracking
- **Threat model:**
  - Kernel is trusted

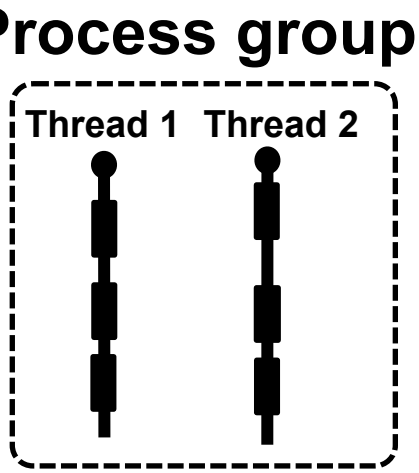
Record and Replay



- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread
- **Implementation:**
  - Arnold* with 32-bit Linux kernel

*David Devecsery, Michael Chow, Xianzheng Dou, Peter M Chen, Jason Flinn. **Eidetic Systems**. Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI), October 2014.

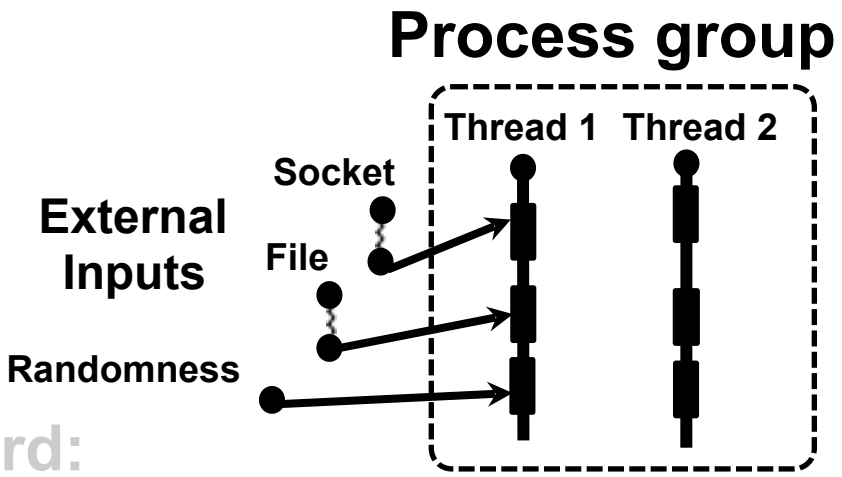
Record and Replay



- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread
- **Implementation:**
  - Arnold* with 32-bit Linux kernel

*David Devecsery, Michael Chow, Xianzheng Dou, Peter M Chen, Jason Flinn. **Eidetic Systems**. Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI), October 2014.

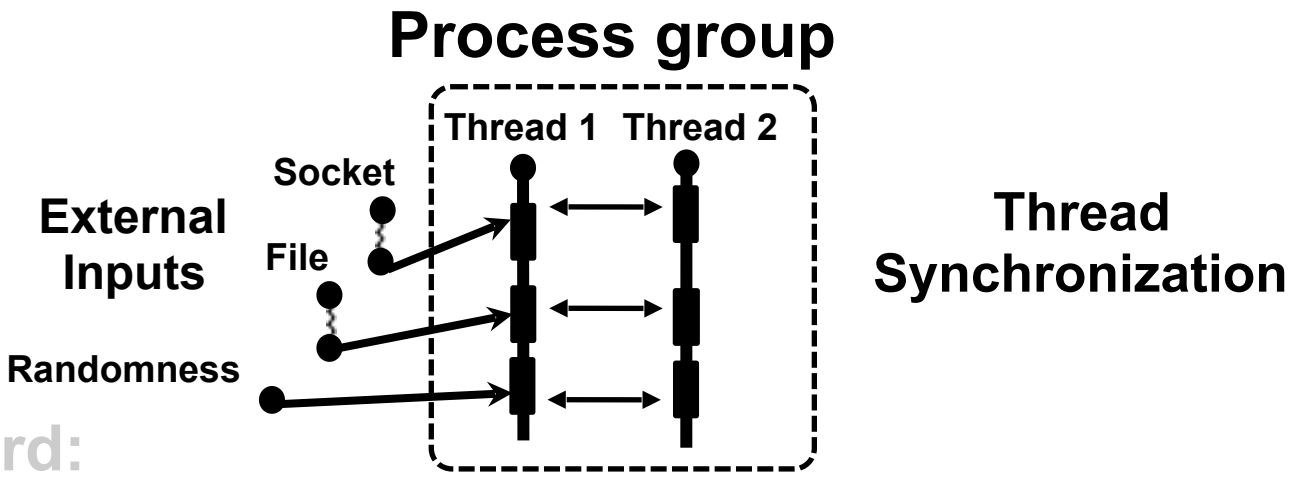
Record and Replay



- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread
- **Implementation:**
  - Arnold* with 32-bit Linux kernel

*David Devecsery, Michael Chow, Xianzheng Dou, Peter M Chen, Jason Flinn. **Eidetic Systems**. Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI), October 2014.

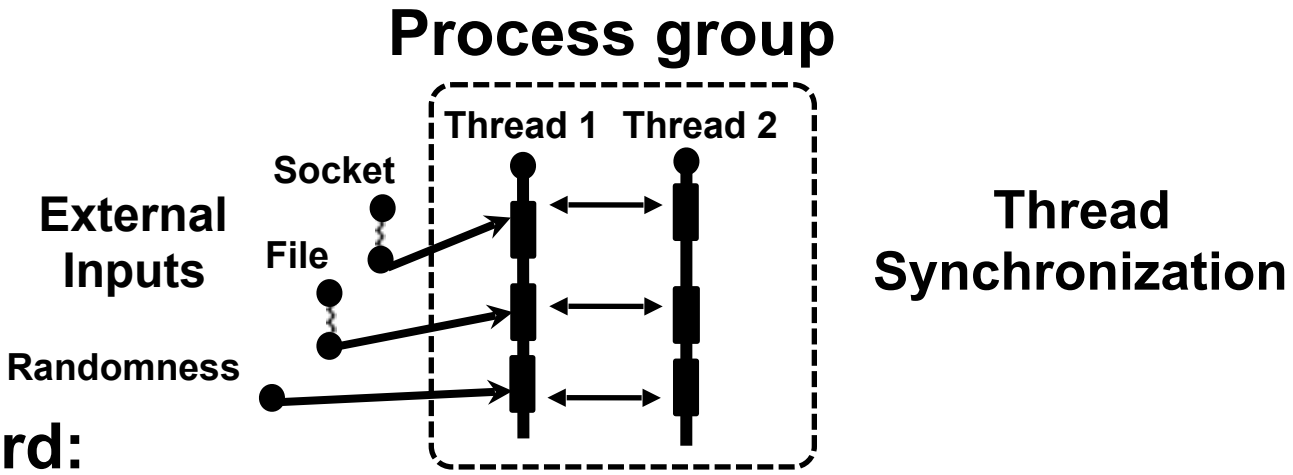
Record and Replay



- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread
- **Implementation:**
  - Arnold* with 32-bit Linux kernel

*David Devecsery, Michael Chow, Xianzheng Dou, Peter M Chen, Jason Flinn. **Eidetic Systems**. Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI), October 2014.

Record and Replay



- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread
- **Implementation:**
  - Arnold* with 32-bit Linux kernel

*David Devecsery, Michael Chow, Xianzheng Dou, Peter M Chen, Jason Flinn. **Eidetic Systems**. Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI), October 2014.

Kernel Instrumentation Implementation Example

```
unsigned long arch_align_stack(unsigned long sp
{
        /* Begin REPLAY */
        if (!(current->personality & ADDR_NO_RANDOMIZE) &&
        randomize_va_space){
                unsigned int rand = get_random_int();
        if (current->record_thrd) {
        record_randomness(rand);
                } else if (current->replay_thrd){
                rand = replay_randomness();
}
                sp -= rand % 8192;
        }
        /* End REPLAY */
        return sp & ~0xf;
}
```

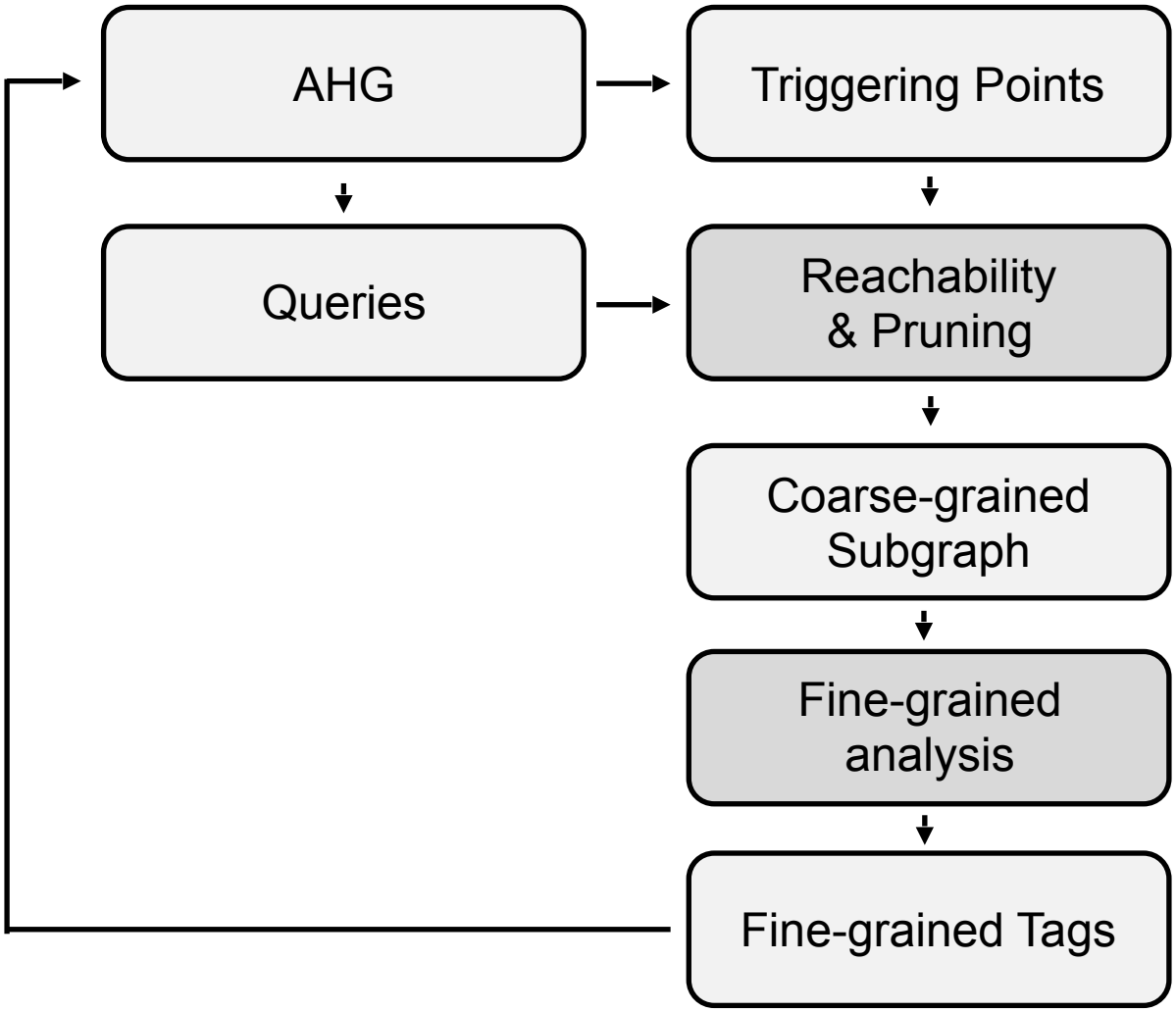
Kernel Instrumentation Implementation Example

```
unsigned long arch_align_stack(unsigned long sp
{
        /* Begin REPLAY */
        if (!(current->personality & ADDR_NO_RANDOMIZE) &&
        randomize_va_space){
                unsigned int rand = get_random_int();
        if (current->record_thrd) {
        record_randomness(rand);
                } else if (current->replay_thrd){
                rand = replay_randomness();
}
                sp -= rand % 8192;
        }
        /* End REPLAY */
        return sp & ~0xf;
}
```

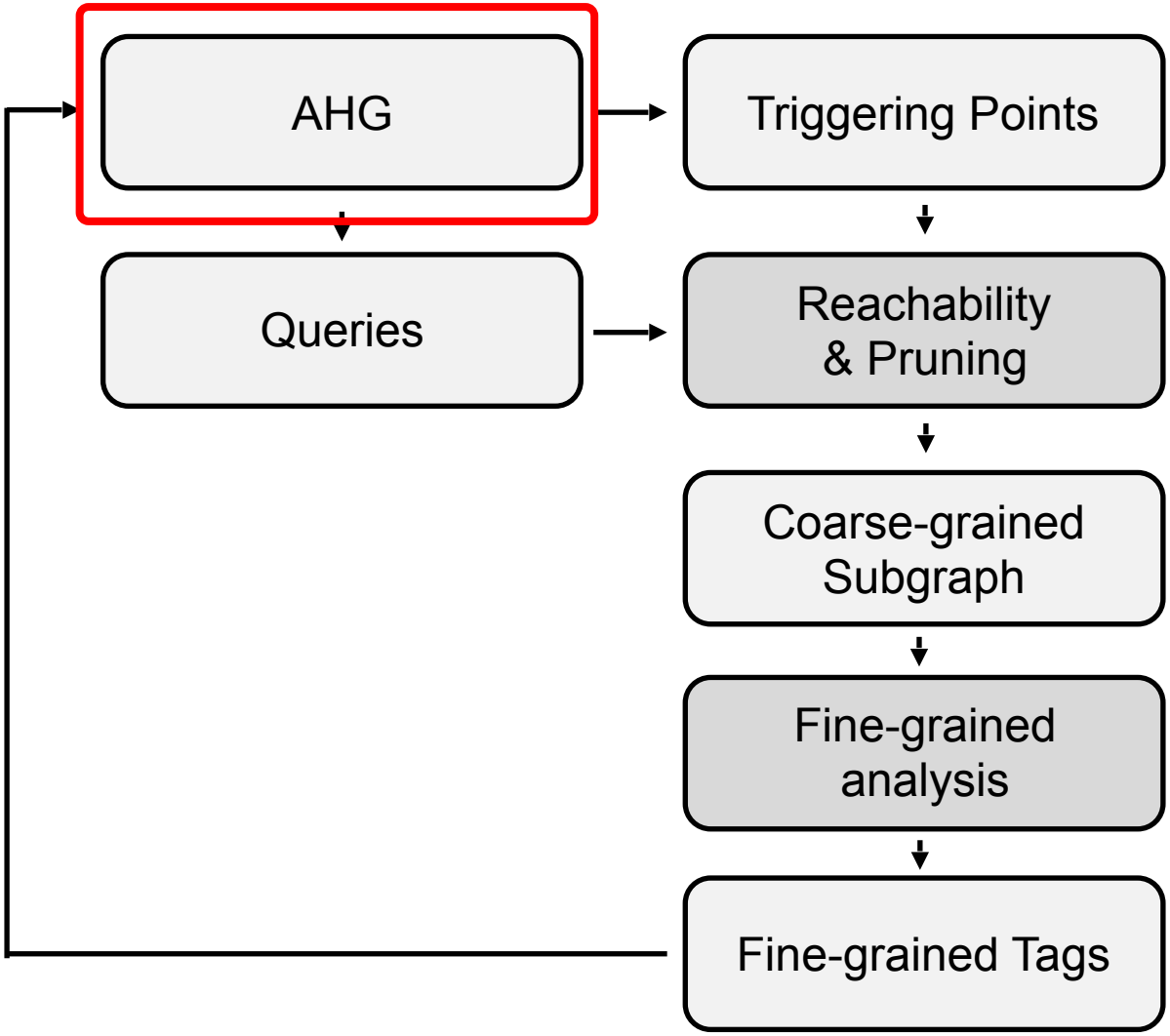
Kernel Instrumentation Implementation Example

```
unsigned long arch_align_stack(unsigned long sp
{
        /* Begin REPLAY */
        if (!(current->personality & ADDR_NO_RANDOMIZE) &&
        randomize_va_space){
                unsigned int rand = get_random_int();
        if (current->record_thrd) {
        record_randomness(rand);
                } else if (current->replay_thrd){
                rand = replay_randomness();
}
                sp -= rand % 8192;
        }
        /* End REPLAY */
        return sp & ~0xf;
}
```

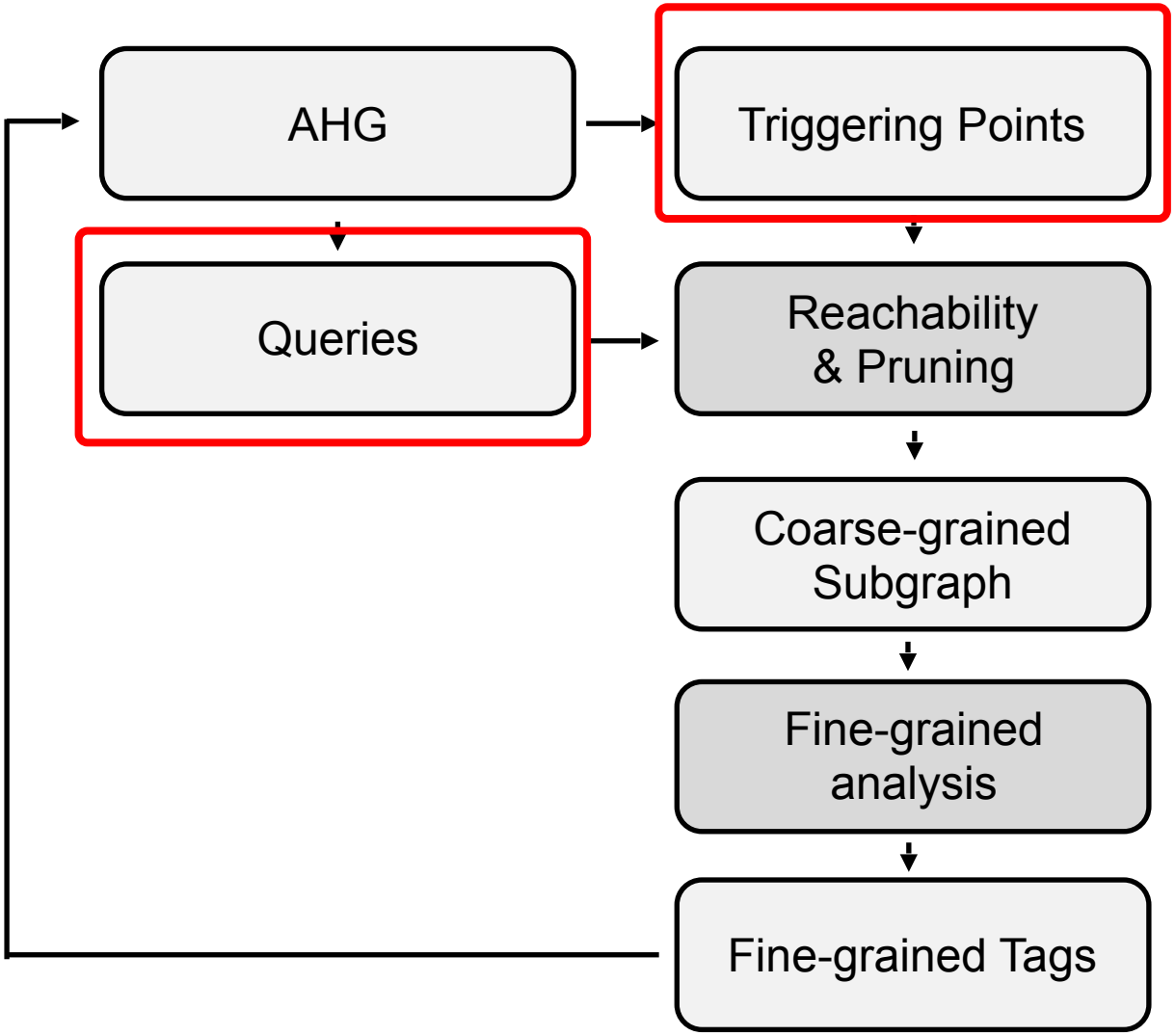
Query System Workflow



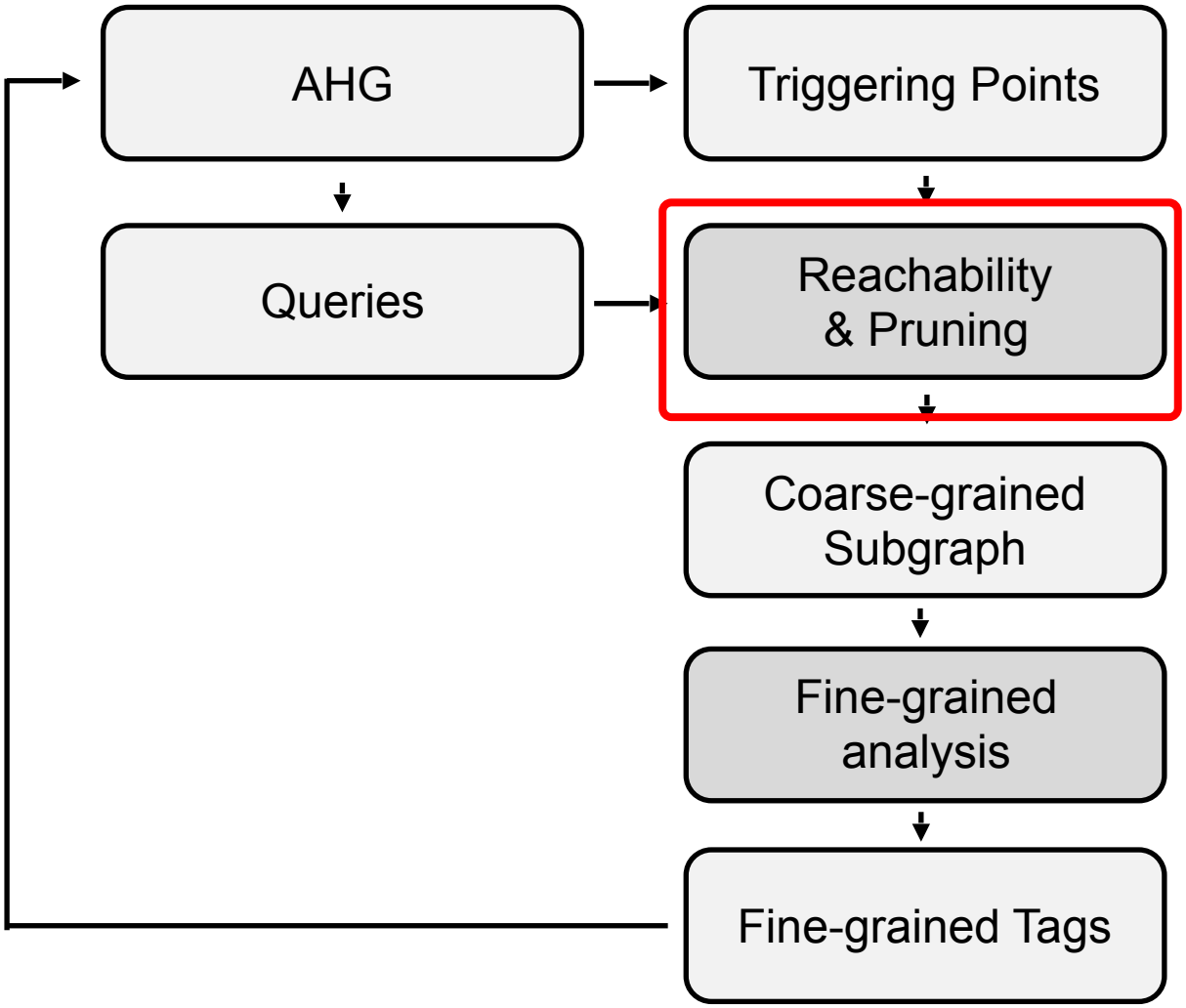
Query System Workflow



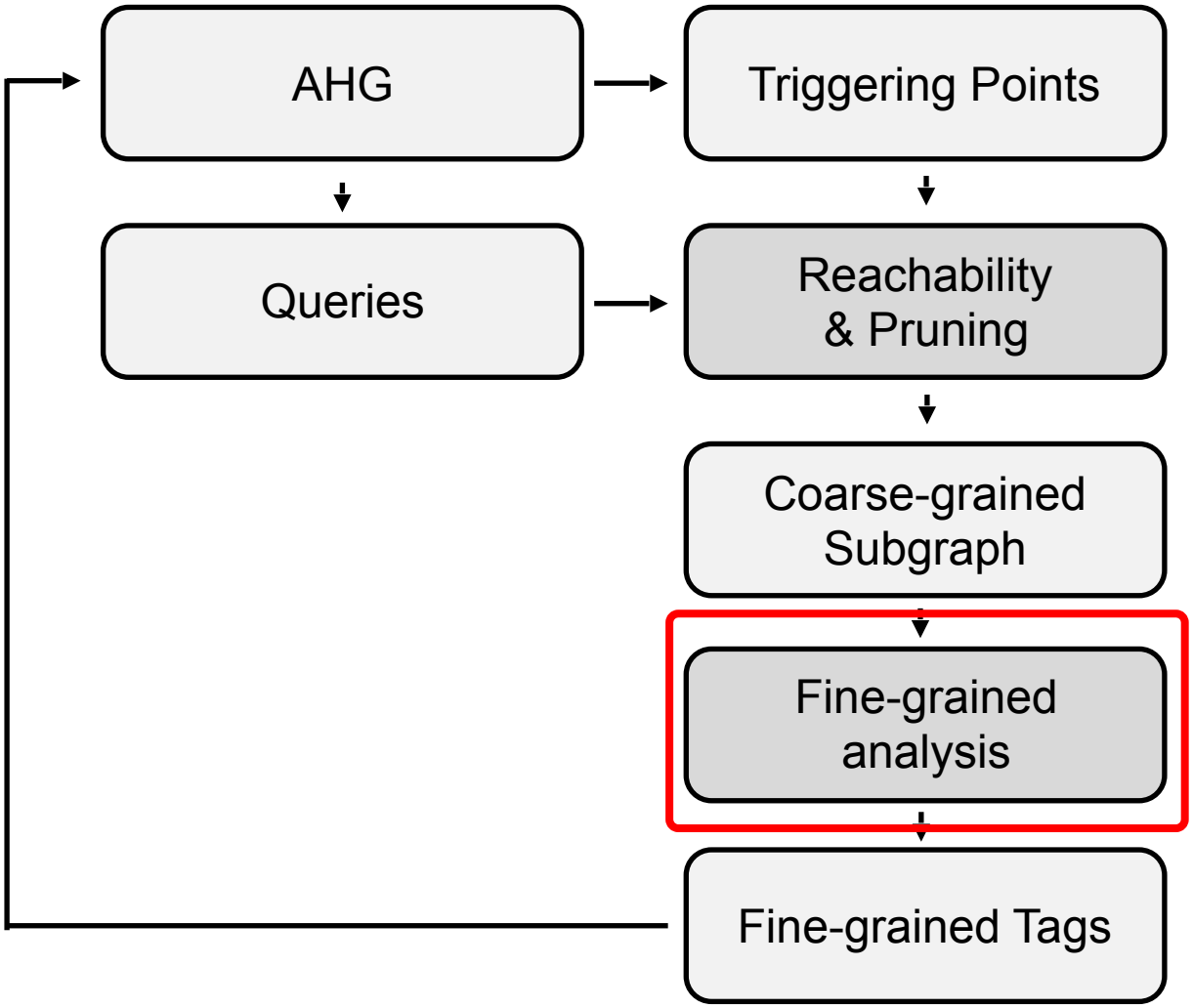
Query System Workflow



Query System Workflow



Query System Workflow



Triggering Points and Queries

- **Triggering points:**
  - Pre-defined policies
    - Process writes to /etc/passwd
- **Queries:**
  - From automated forensic analysis systems
  - Human based analysis
- **Analysis types:**
  - Backward:
    - Where does this object come from?
  - Forward:
    - What is the impact of this object on the system?
  - Point-to-point:
    - Are these two objects related?

Triggering Points and Queries

- **Triggering points:**
  - Pre-defined policies
    - Process writes to /etc/passwd
- **Queries:**
  - From automated forensic analysis systems
  - Human based analysis
- **Analysis types:**
  - Backward:
    - Where does this object come from?
  - Forward:
    - What is the impact of this object on the system?
  - Point-to-point:
    - Are these two objects related?

Triggering Points and Queries

- **Triggering points:**
  - Pre-defined policies
    - Process writes to /etc/passwd
- **Queries:**
  - From automated forensic analysis systems
  - Human based analysis
- **Analysis types:**
  - Backward:
    - Where does this object come from?
  - Forward:
    - What is the impact of this object on the system?
  - Point-to-point:
    - Are these two objects related?

Point-to-point Query Example

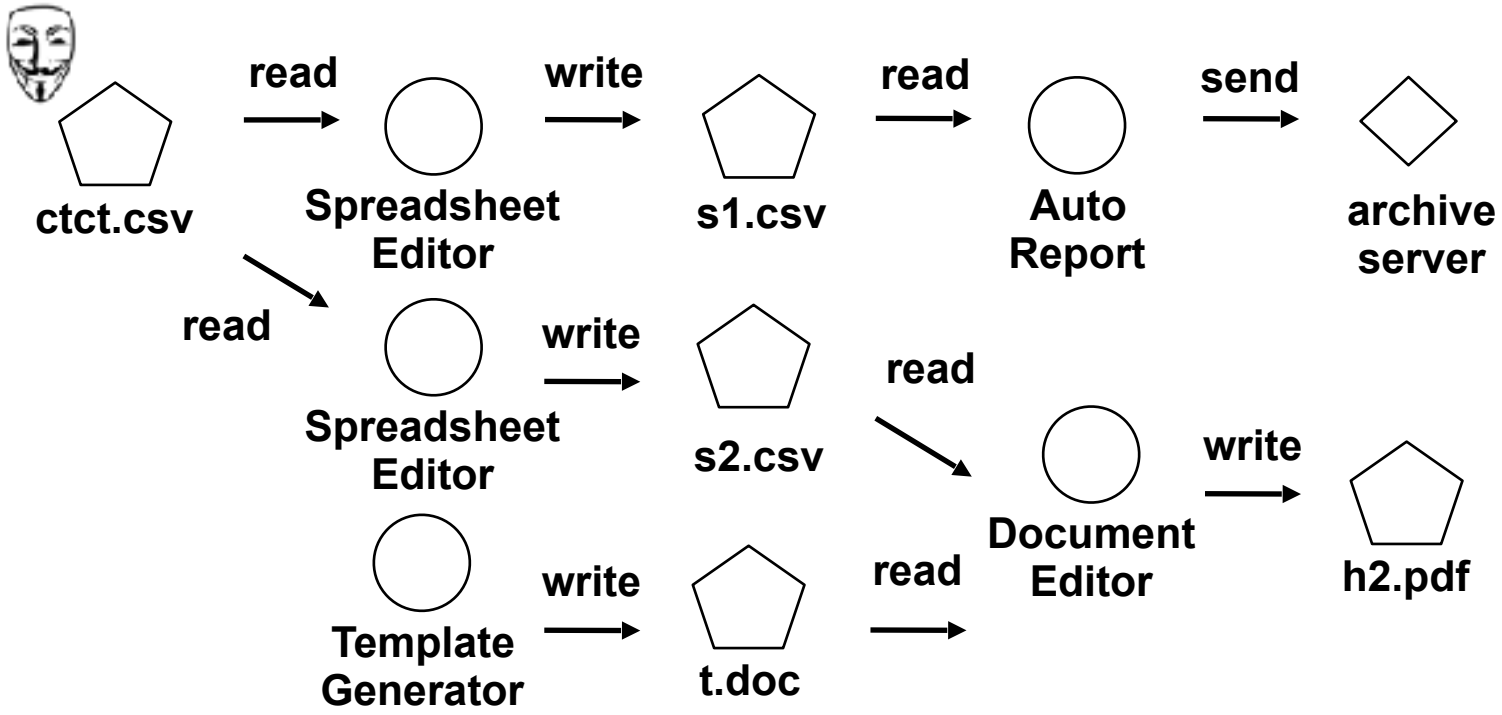
1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor

Point-to-point Query Example

1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor

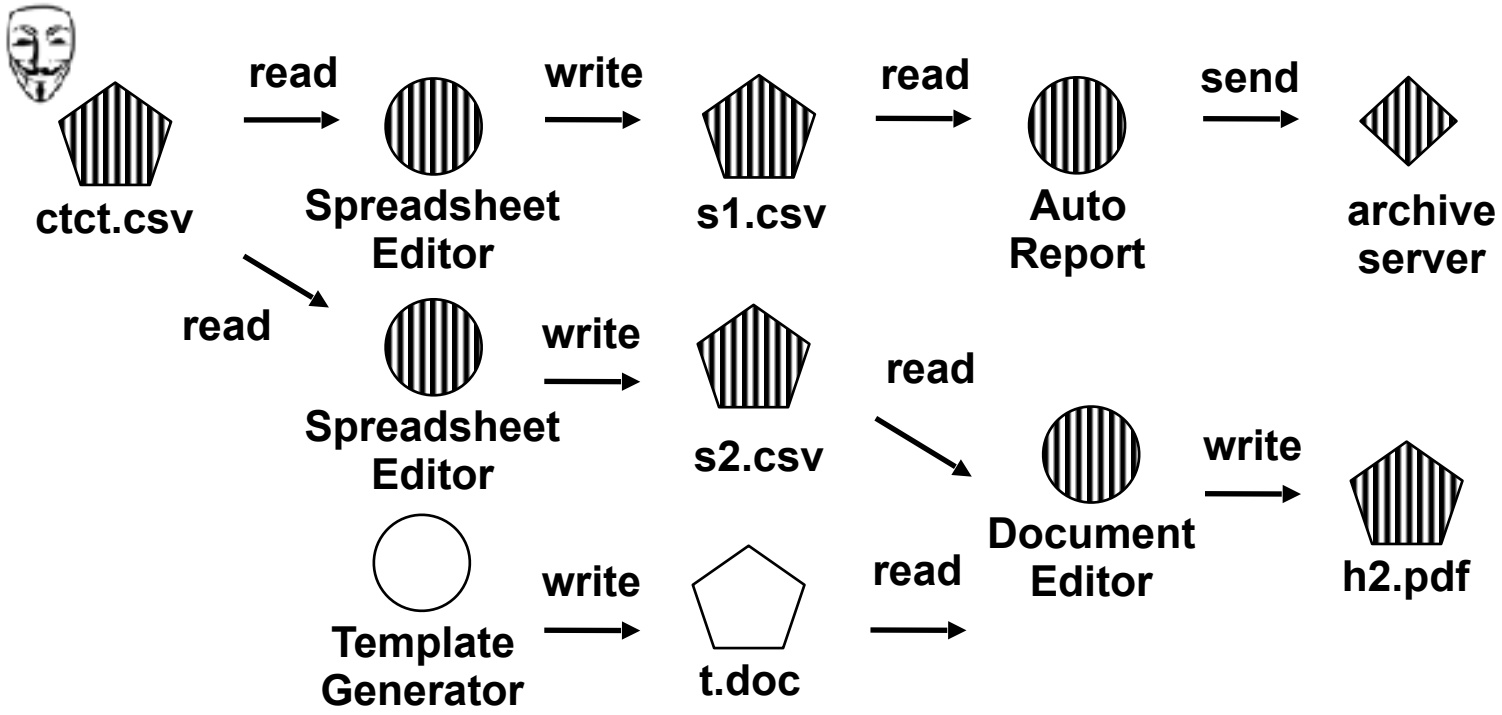
Point-to-point Query Example

1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor



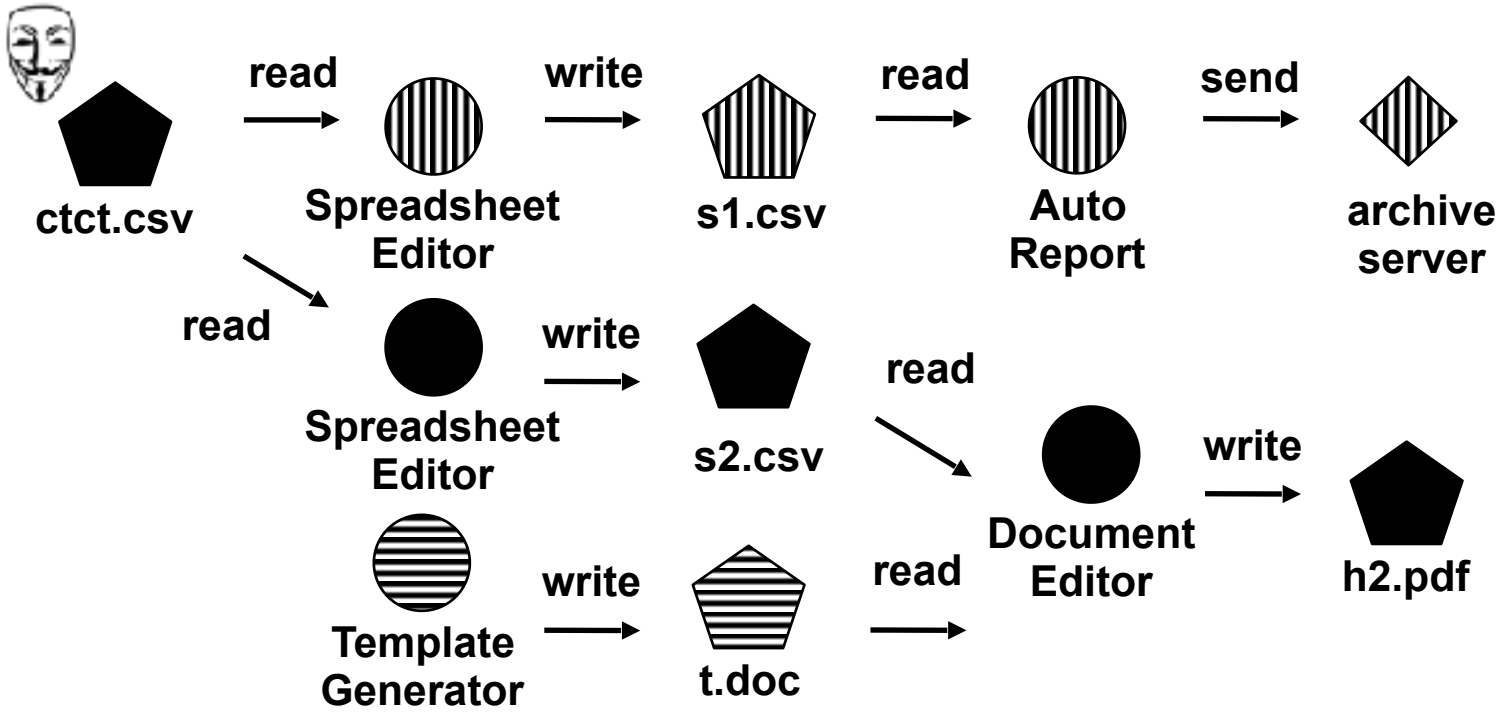
Forward Reachability

1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor



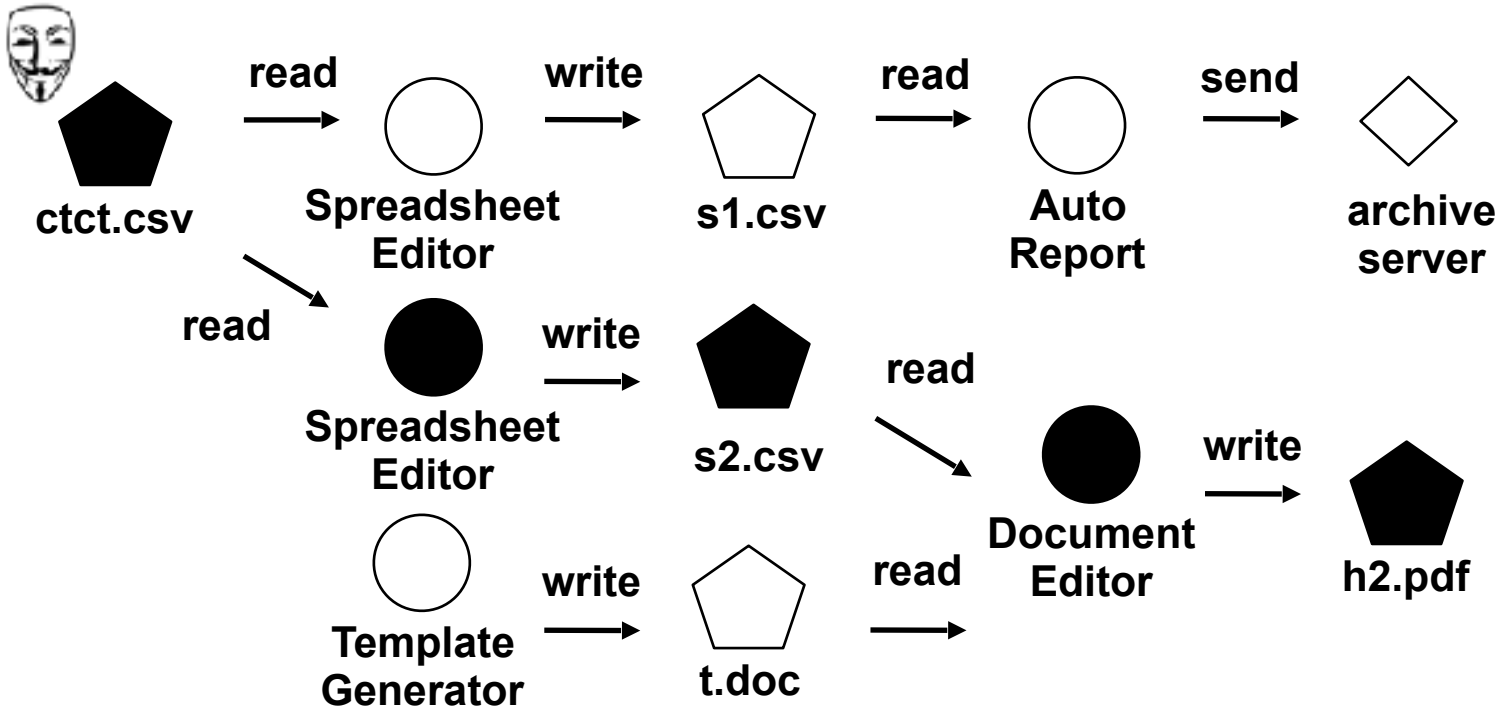
Backward Reachability

1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor

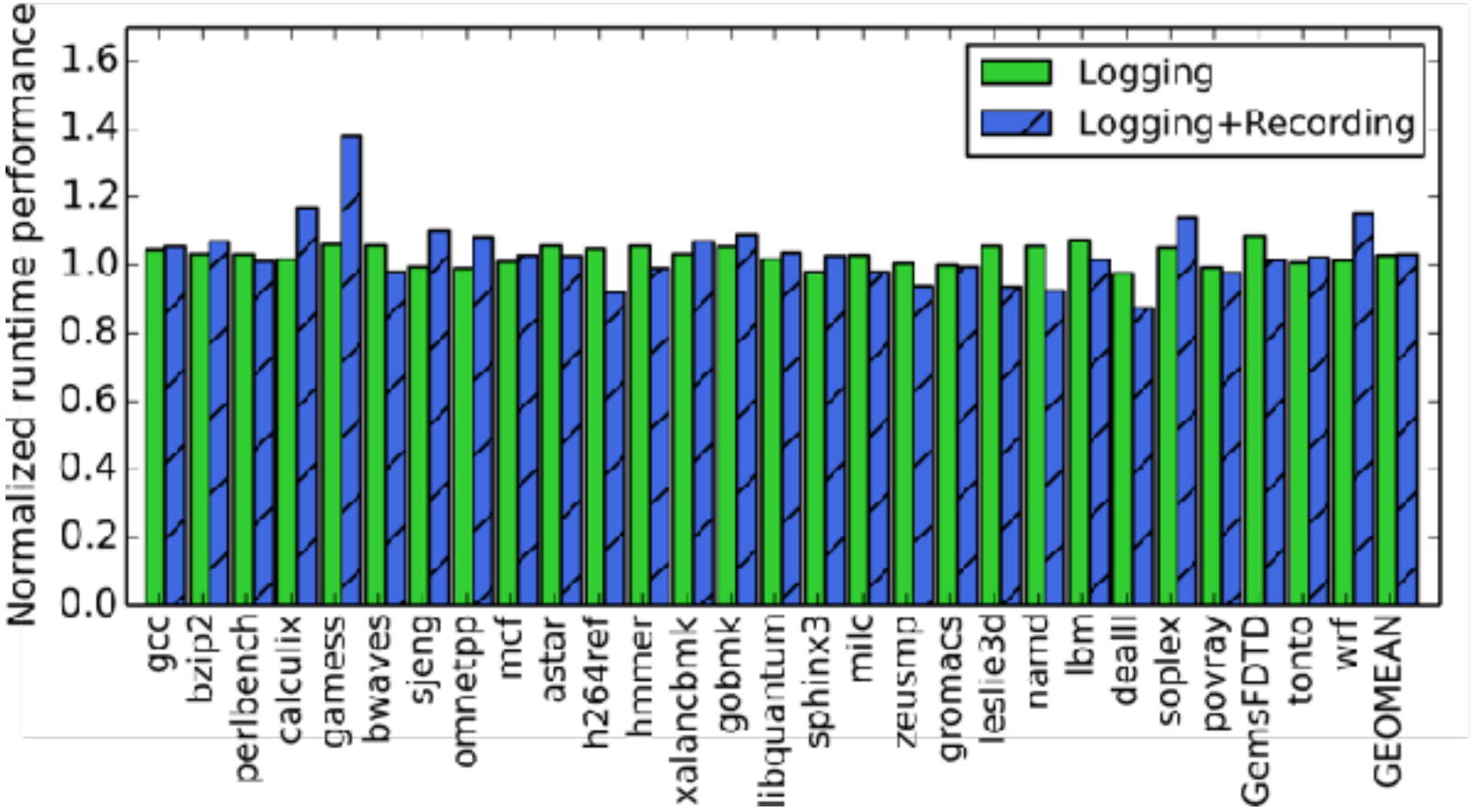


Reachability Result

1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. Auto report program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. Template generator creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor

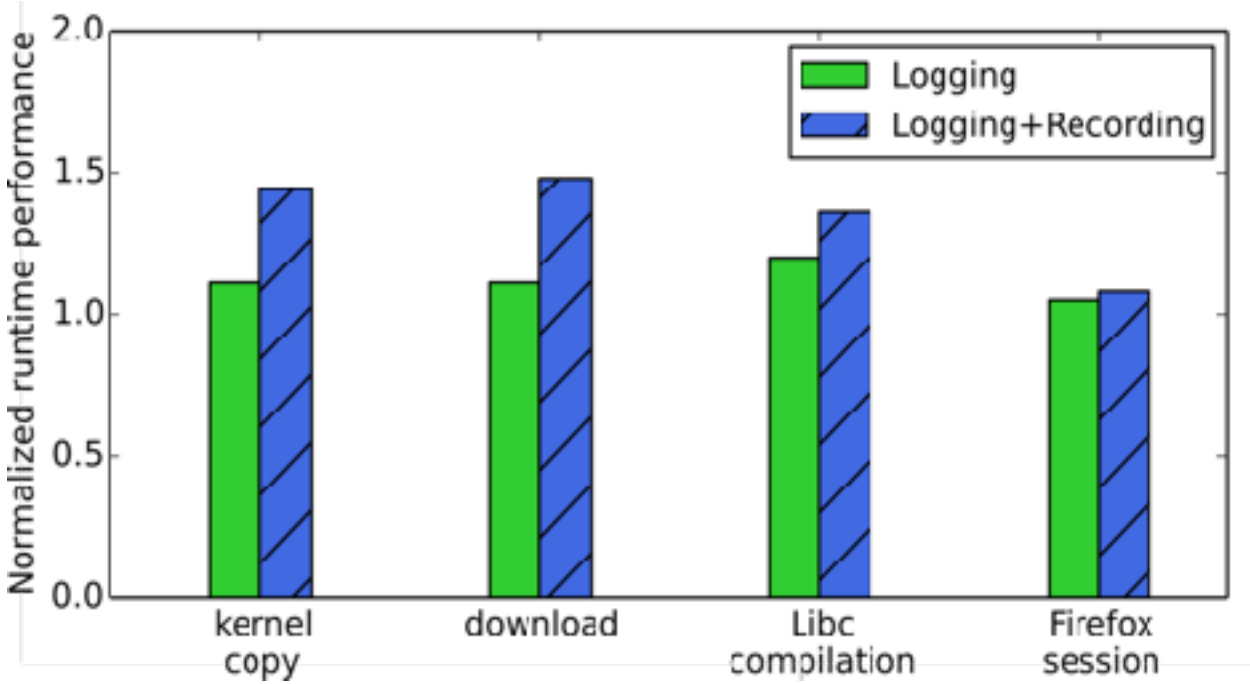


Runtime Overhead: SPEC CPU2006



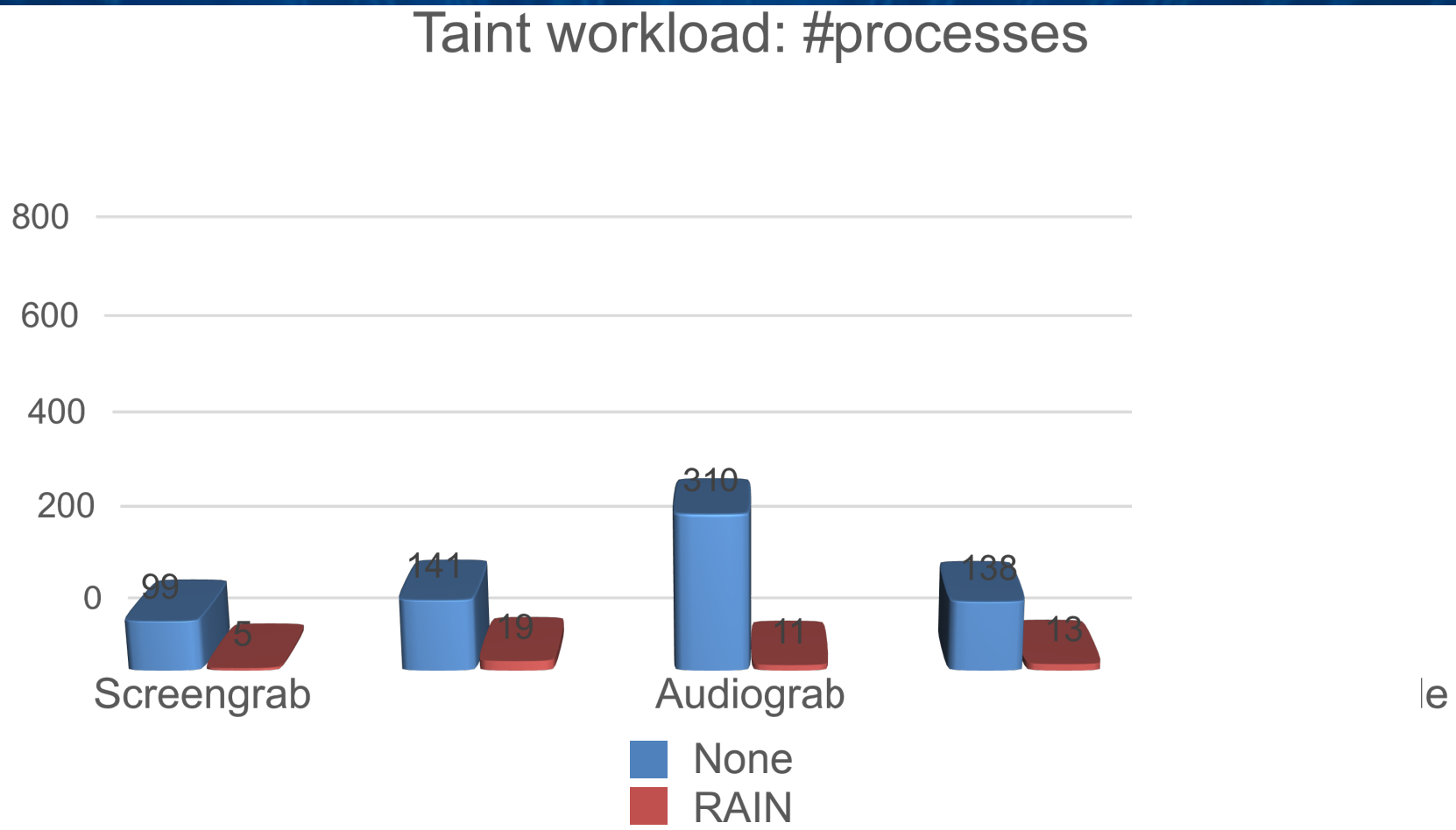
3.22%

Runtime Overhead: I/O Operations



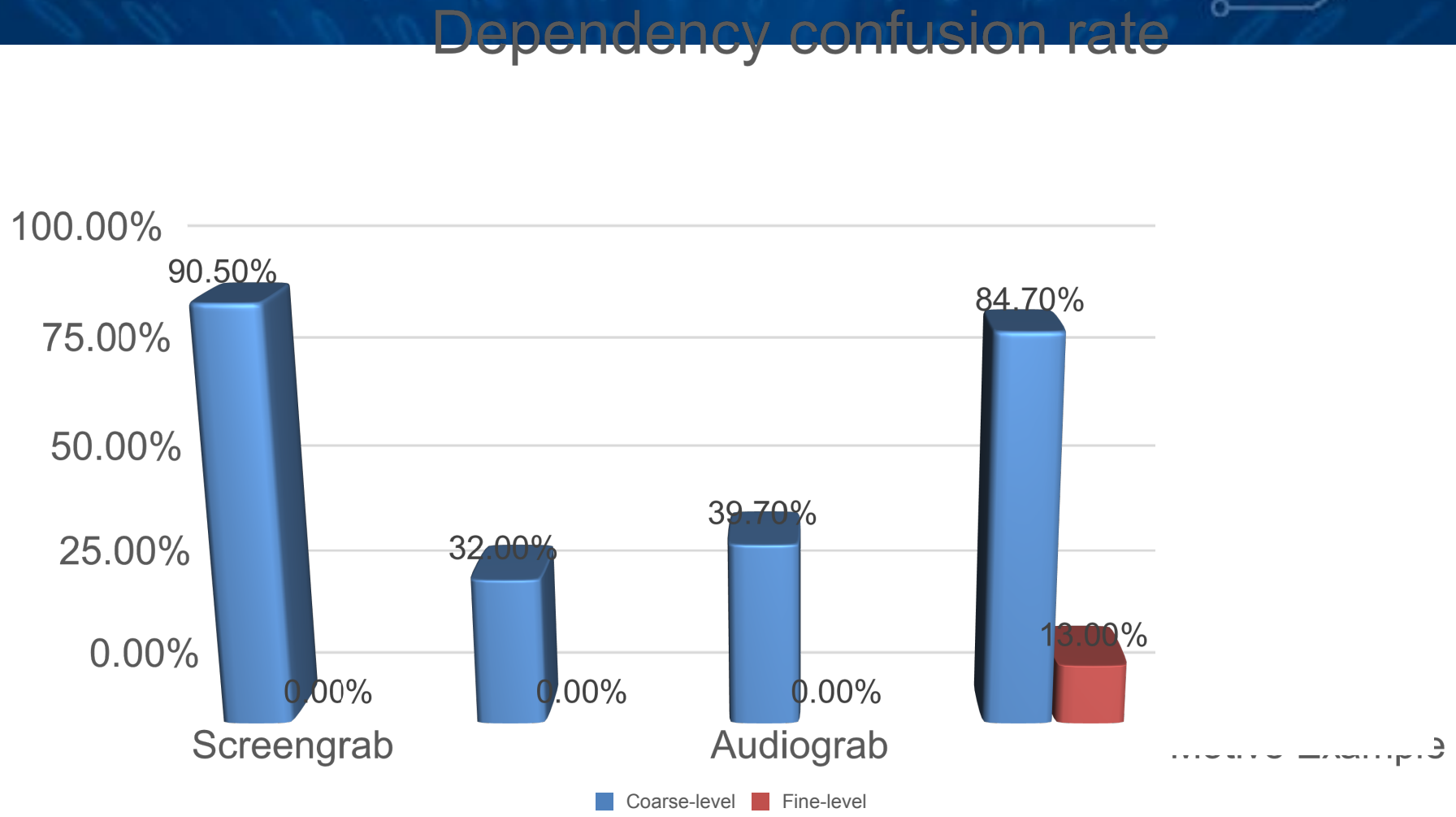
<50%

Pruning Efficiency



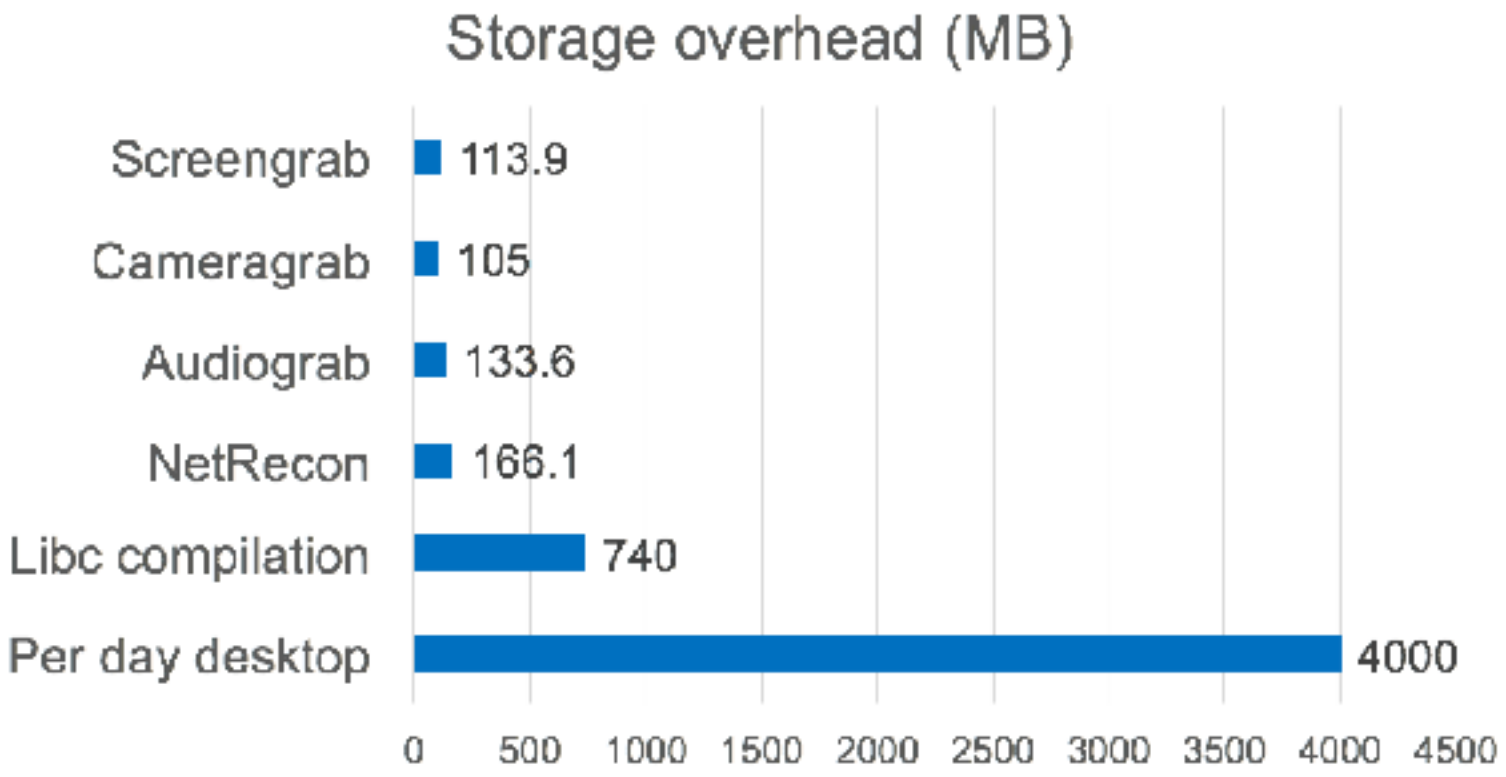
~94.2% reduction

Information Flow Tracking Accuracy



~94.2% reduction

Storage Cost



~4GB per day

Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Conclusion

Data Breaches

EQUIFAX

SONY

LinkedIn

YAHOO!

THEIA-Panda Overview

THEIA-KI Overview

Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Conclusion

Data Breaches

EQUIFAX

SONY

LinkedIn

YAHOO!

THEIA-Panda Overview

THEIA-KI Overview

Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Conclusion

Data Breaches

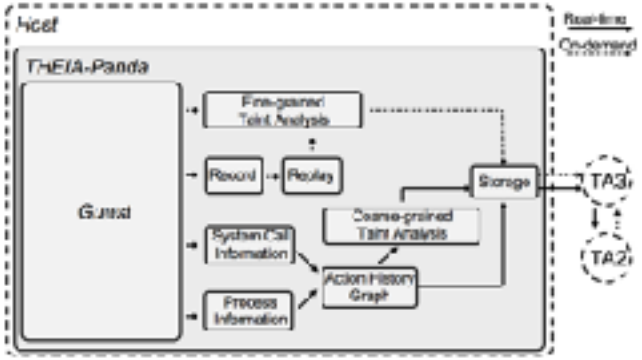
EQUIFAX

SONY

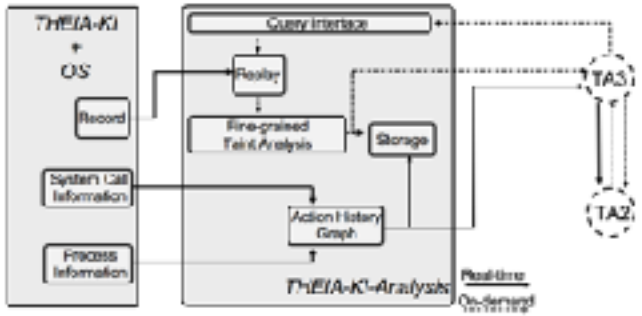
LinkedIn

YAHOO!

THEIA-Panda Overview



THEIA-KI Overview



Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

Conclusion

Data Breaches

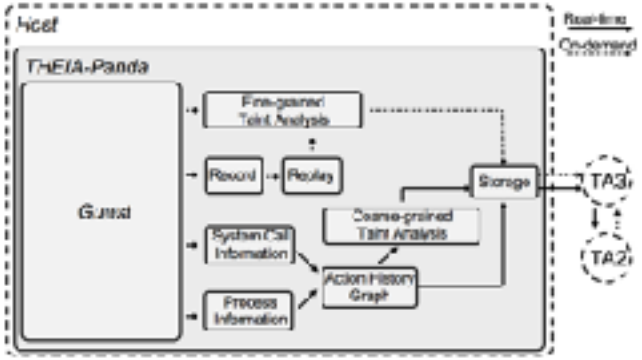
EQUIFAX

SONY

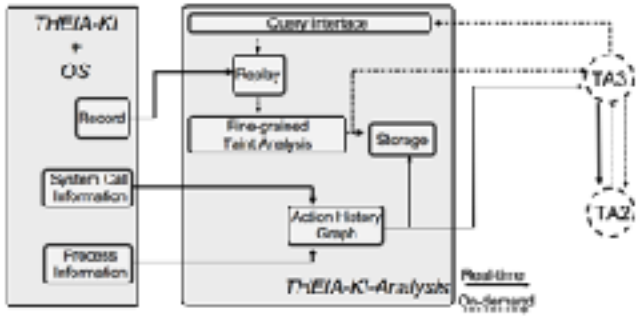
LinkedIn

YAHOO!

THEIA-Panda Overview



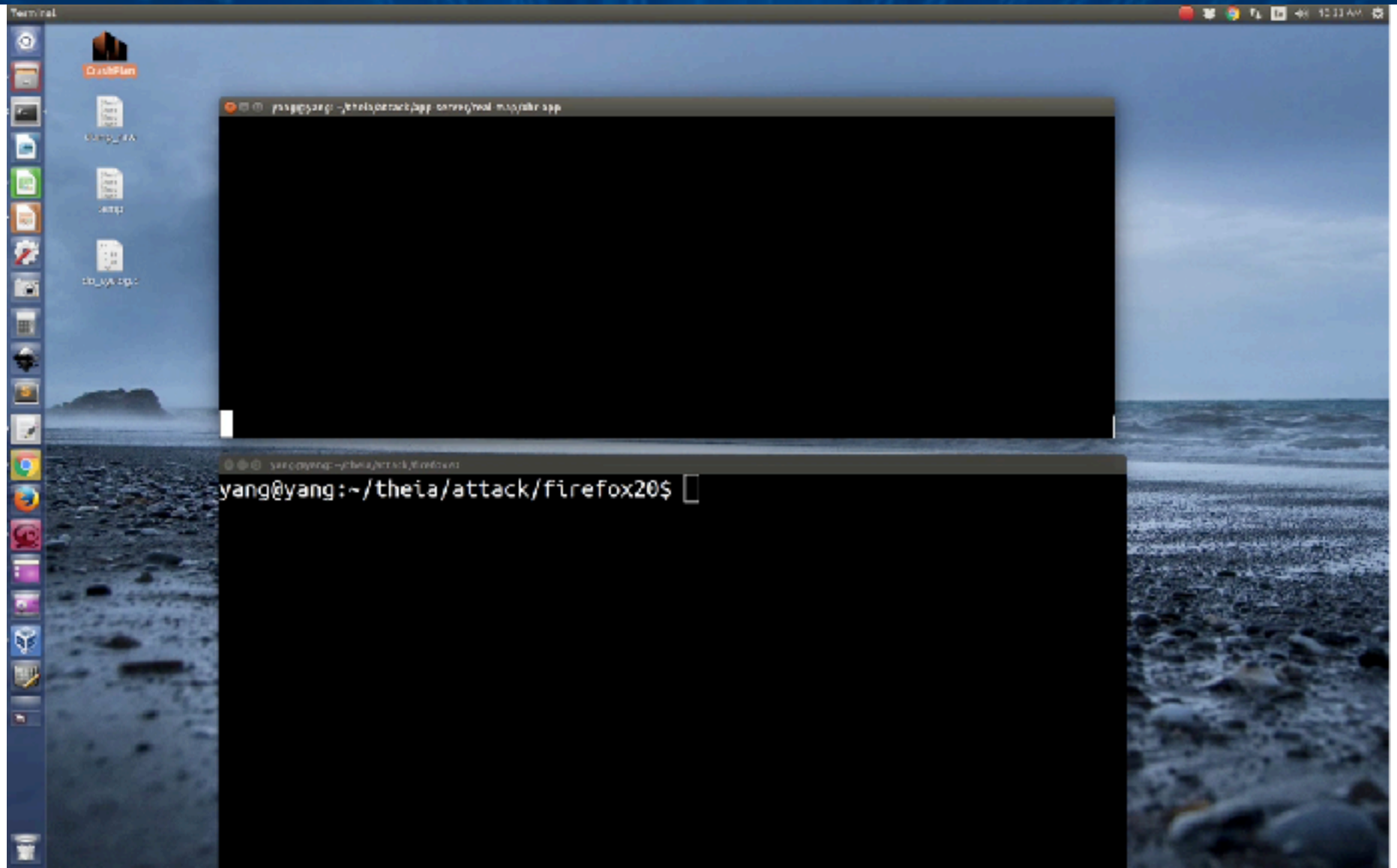
THEIA-KI Overview



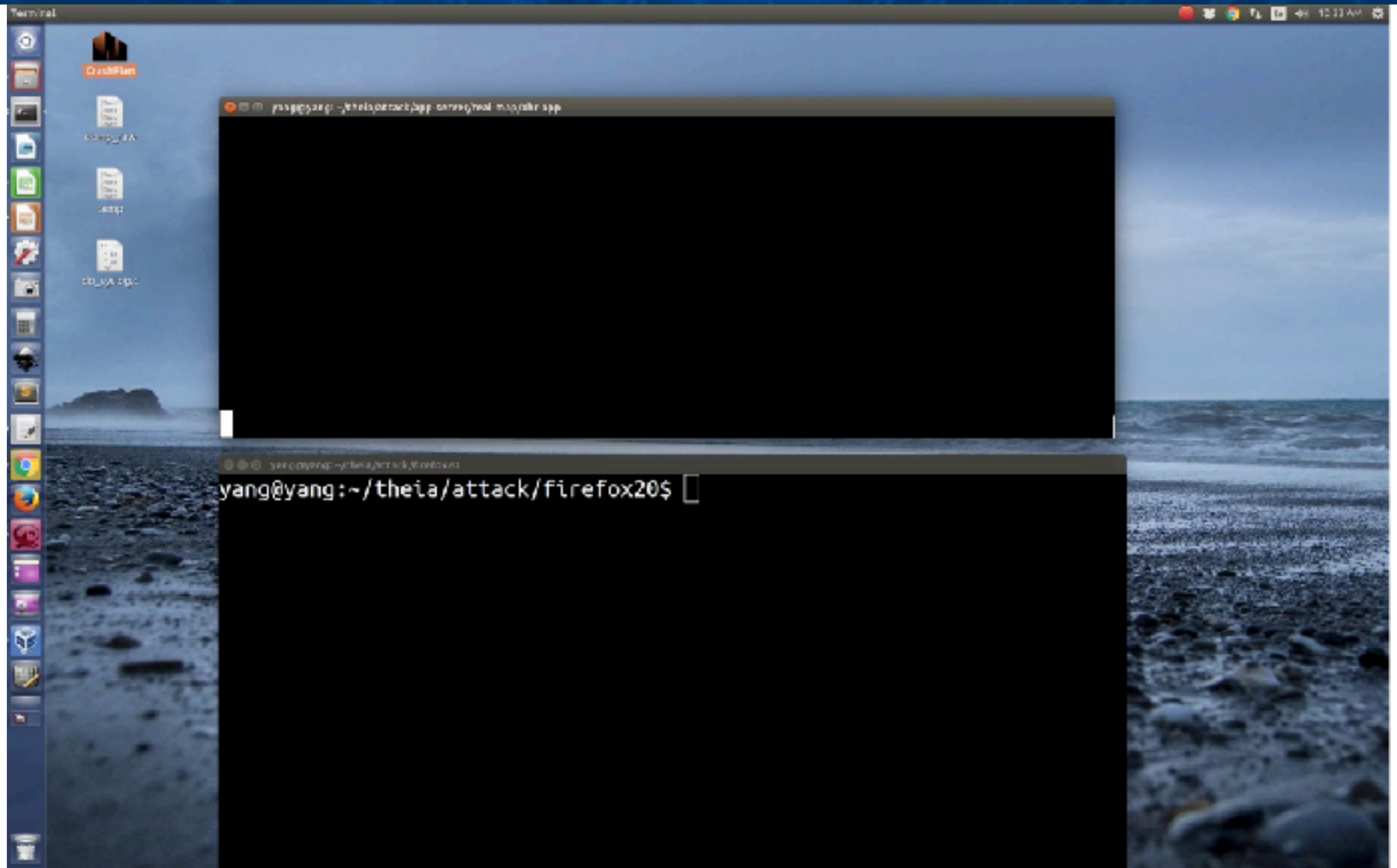
Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit

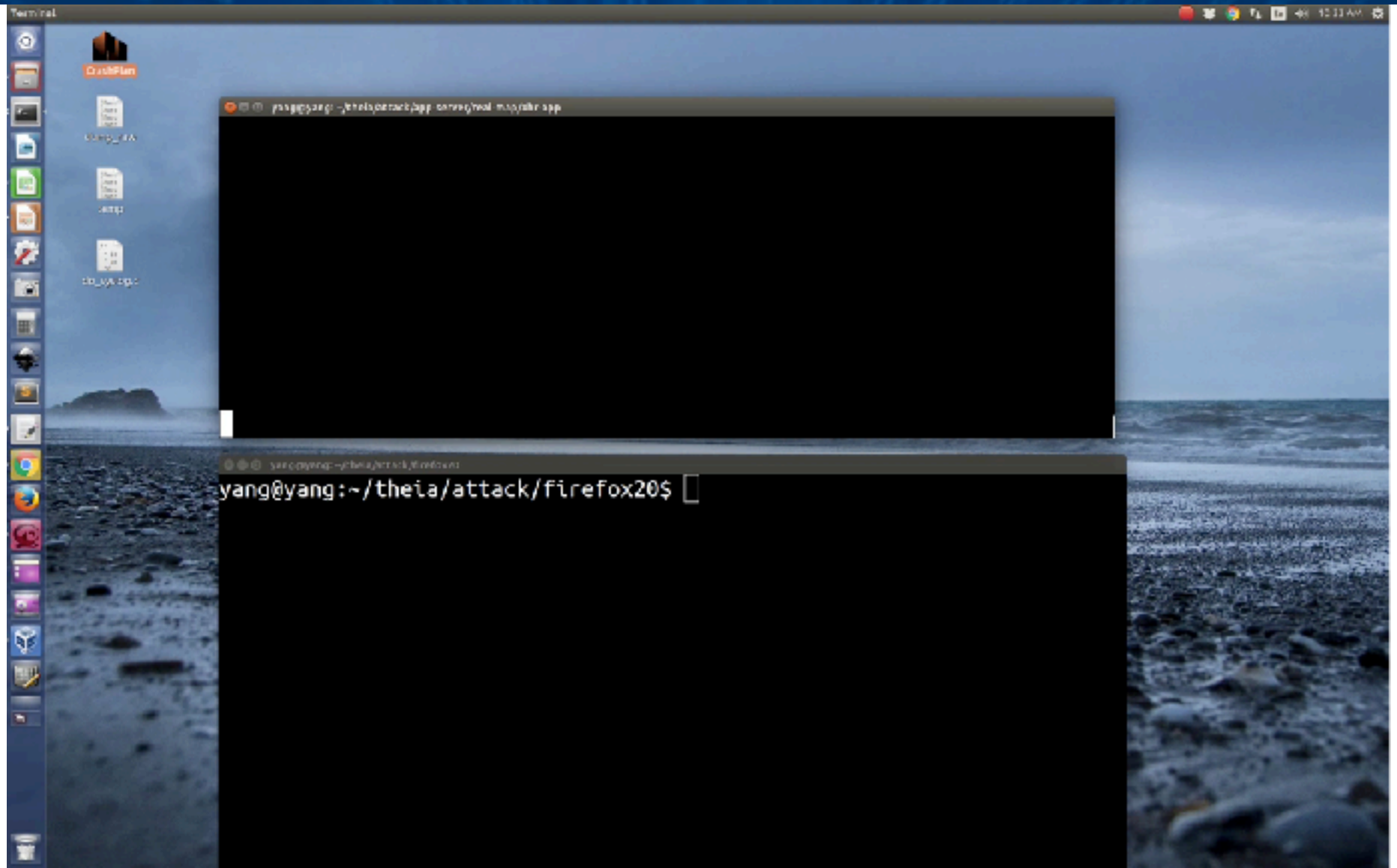
APT Demo



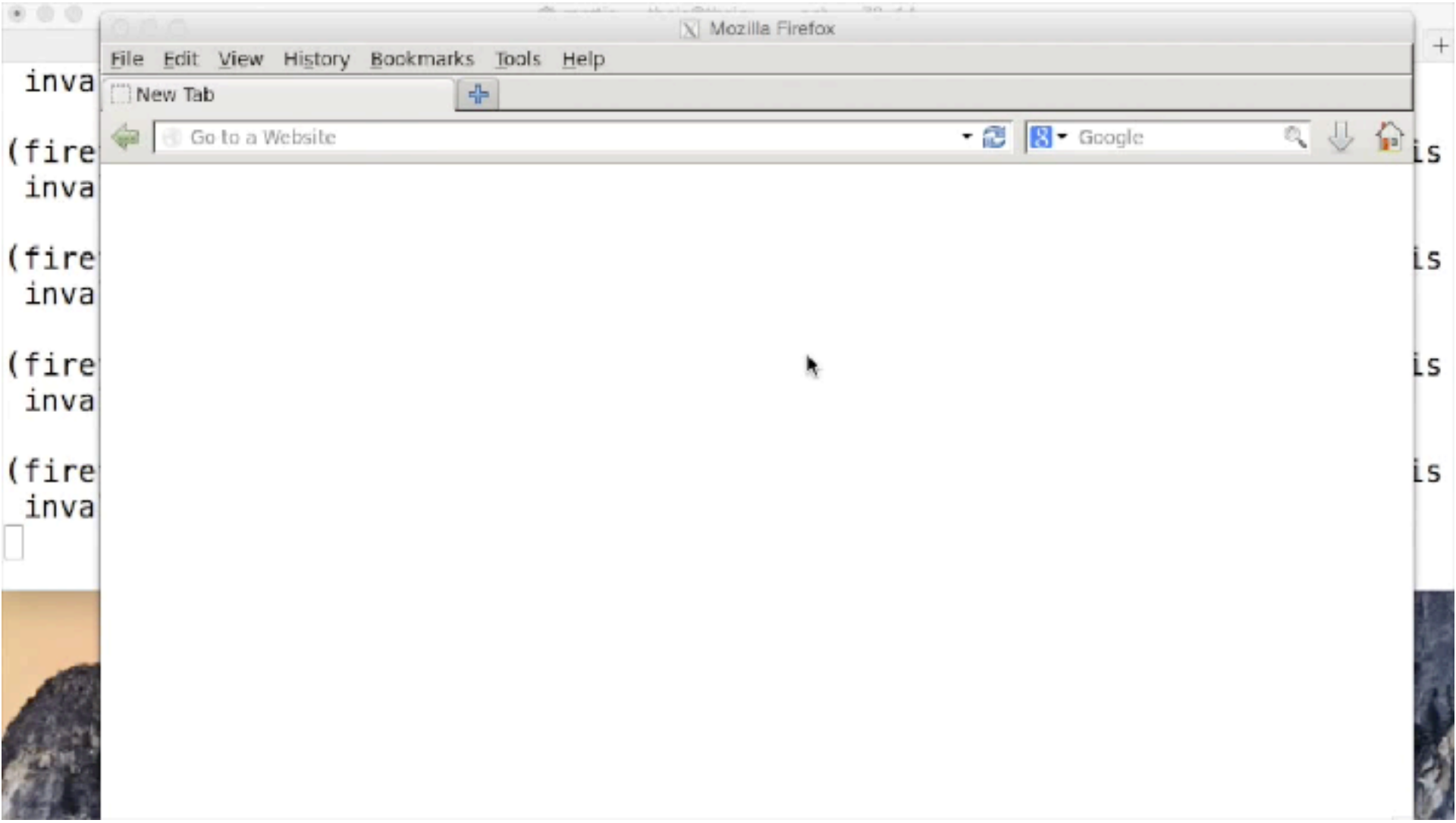
APT Demo



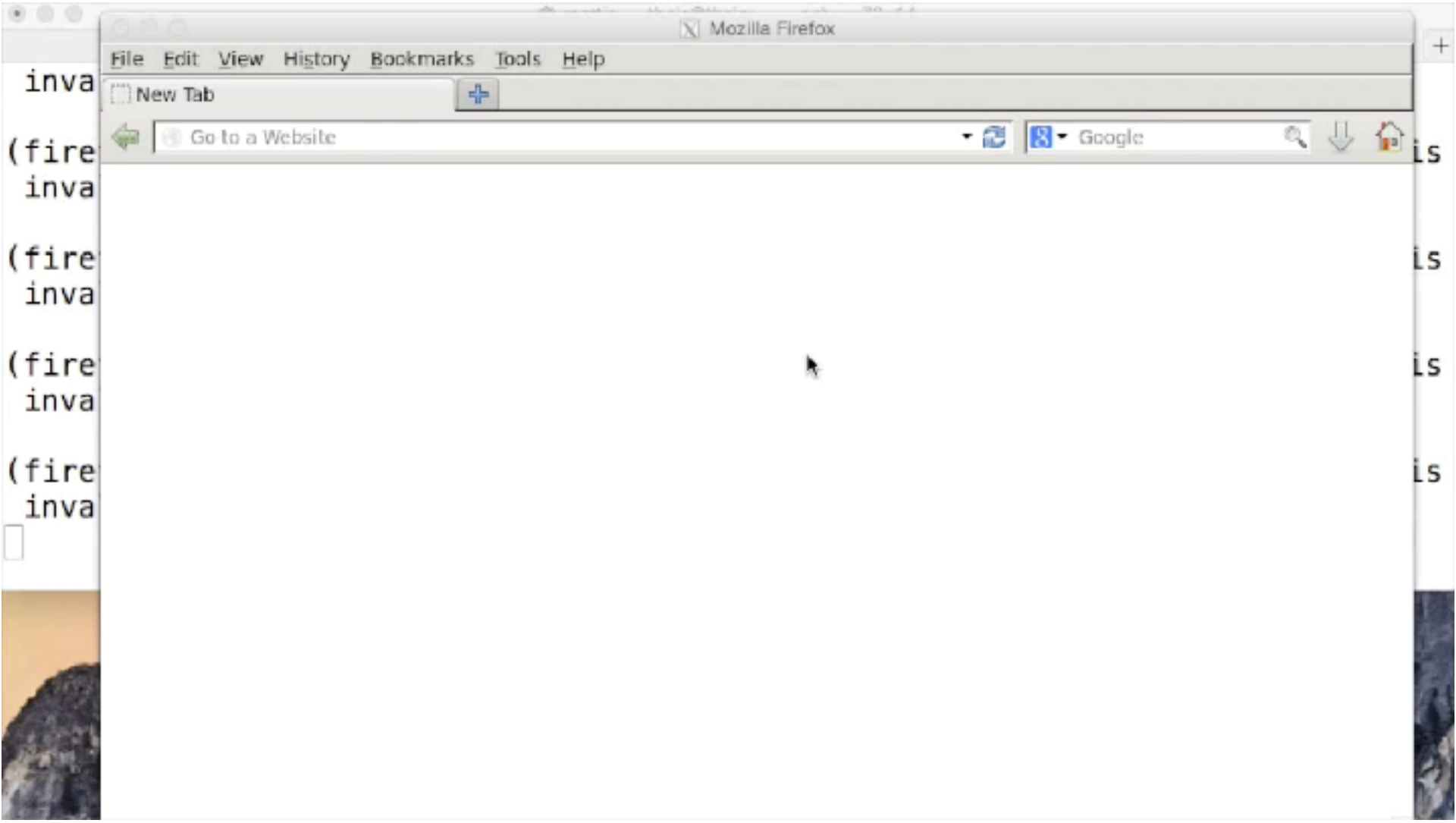
APT Demo



THEIA-Panda Demo



THEIA-Panda Demo



THEIA-Panda Demo