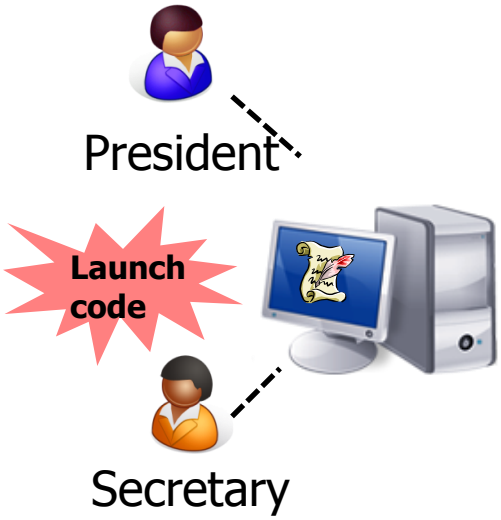


Detecting Covert Timing Channels with Time-deterministic Replay

Ang Chen[*] W. Brad Moore[+] Hanjun Xiao[*]
Andreas Haeberlen[*] Linh Thi Xuan Phan[*] Micah Sherr [+] Wenchao Zhou [+]

University of Pennsylvania[*] Georgetown University [+]

Motivation: Detecting covert timing channels



Motivation: Detecting covert timing channels

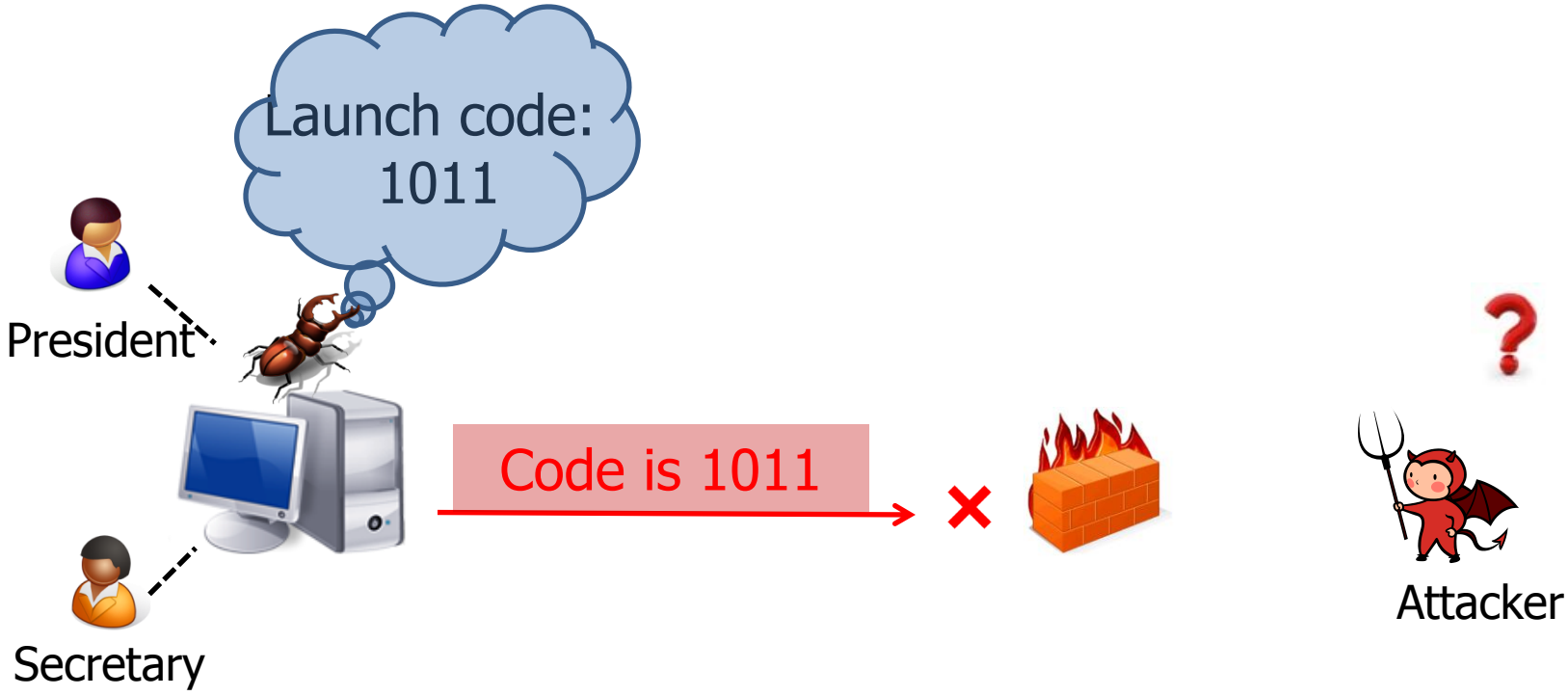


Motivation: Detecting covert timing channels

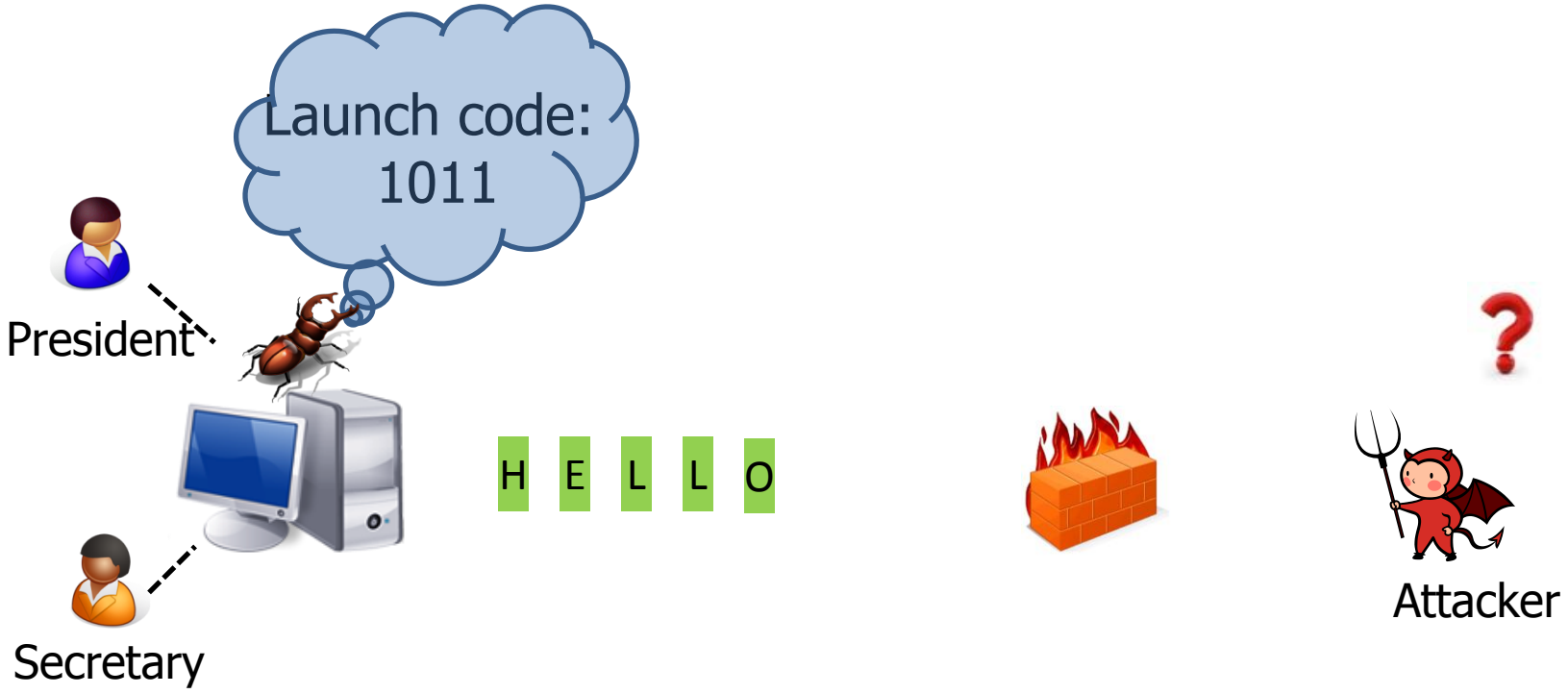




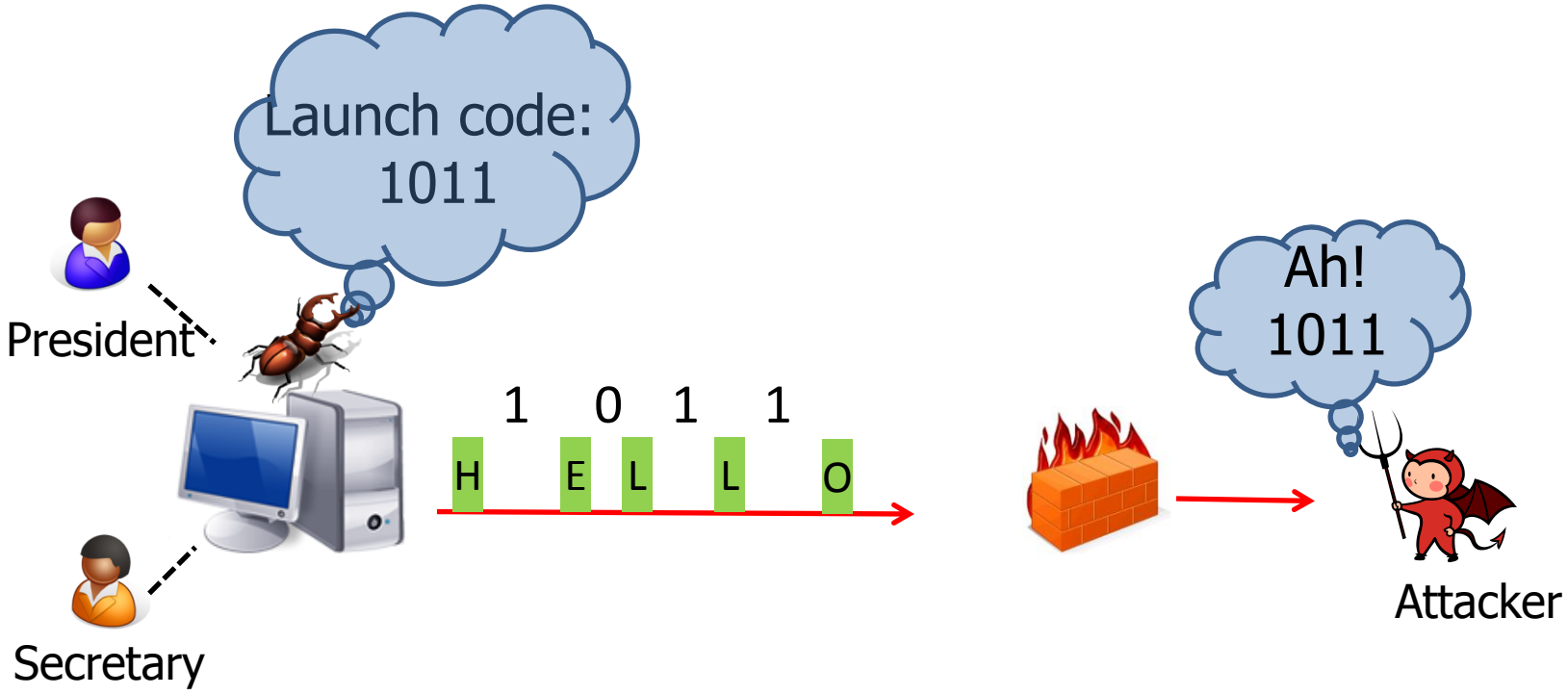
Motivation: Detecting covert timing channels



Motivation: Detecting covert timing channels



Motivation: Detecting covert timing channels



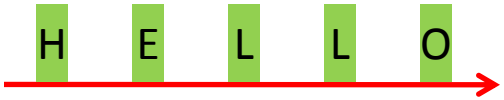
Covert timing channel: Leaks secrets by changing the timing of network outputs

State of the art: Statistics-based detection

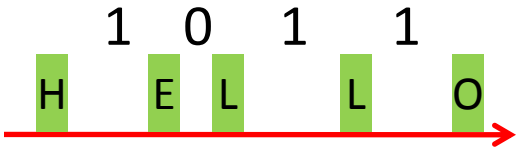
Current approaches look for **specific statistical deviations**

State of the art: Statistics-based detection

Normal traffic:

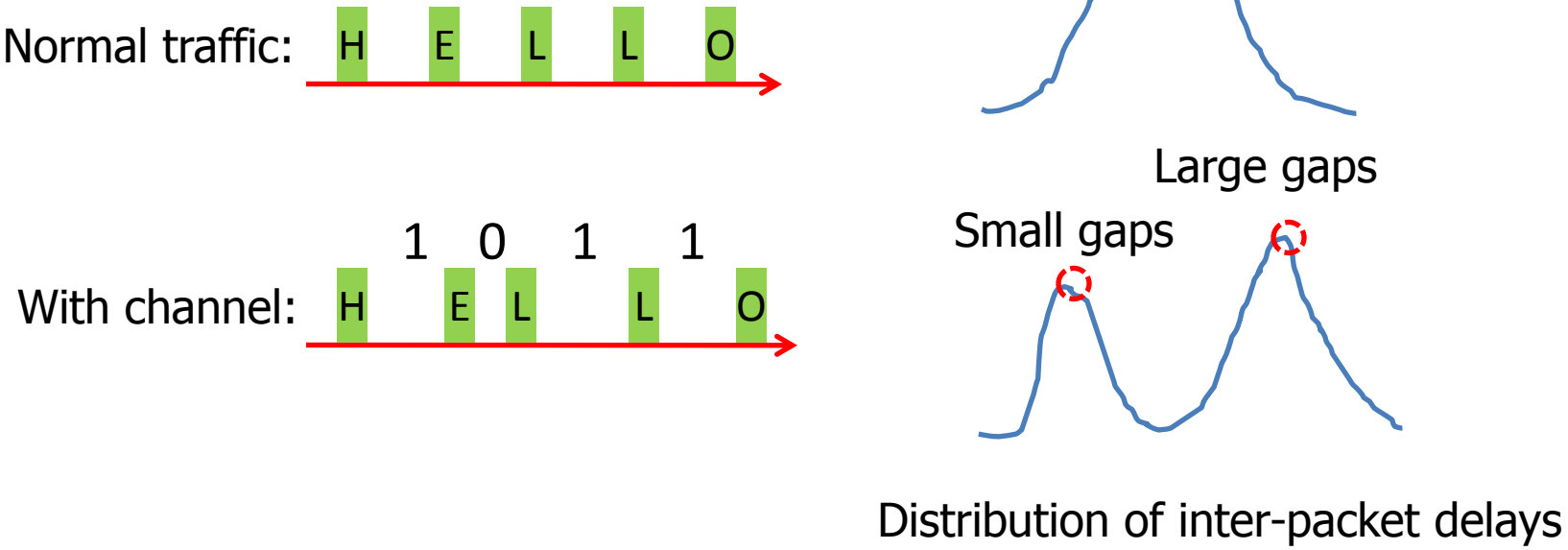


With channel:



Current approaches look for **specific statistical deviations**

State of the art: Statistics-based detection

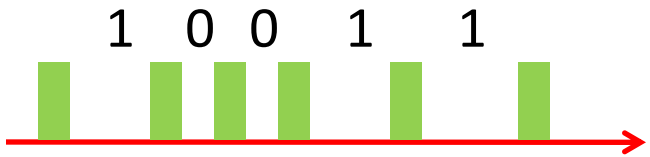


Current approaches look for **specific statistical deviations**

Problem: Making a new channel is easy!

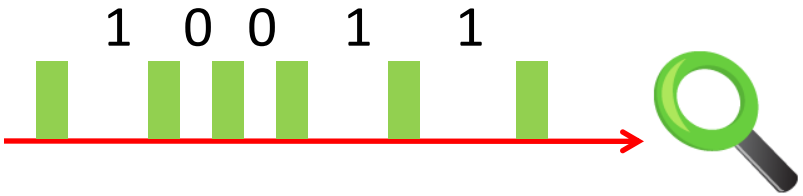
Existing detectors are channel-specific:

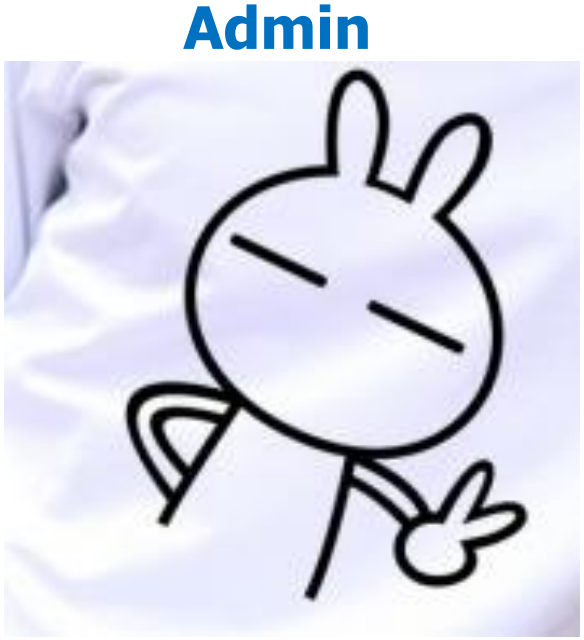
Problem: Making a new channel is easy!



Existing detectors are channel-specific:

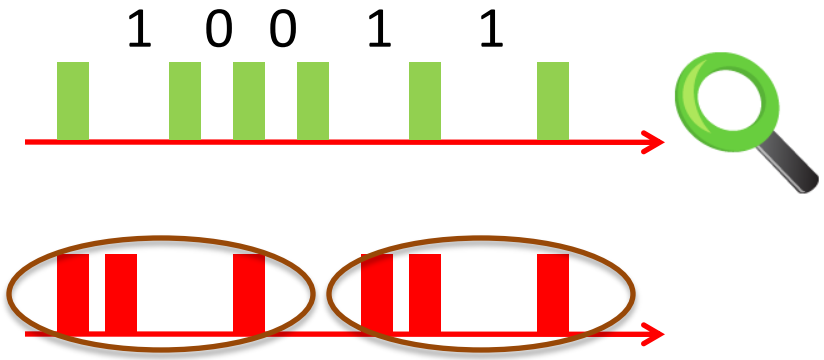
Problem: Making a new channel is easy!





Existing detectors are channel-specific:

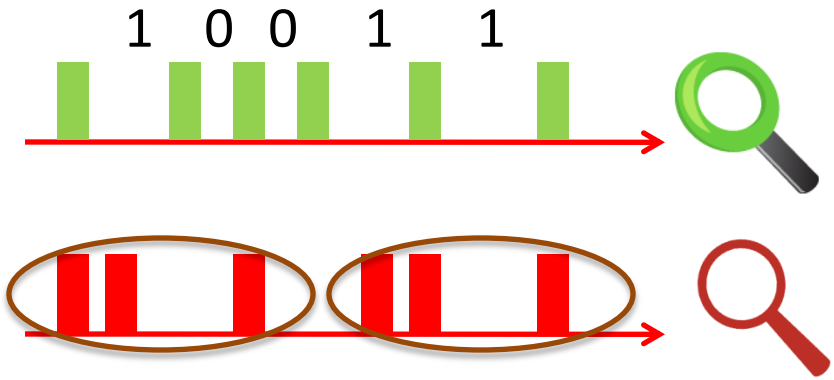
Problem: Making a new channel is easy!





Existing detectors are channel-specific:

Problem: Making a new channel is easy!



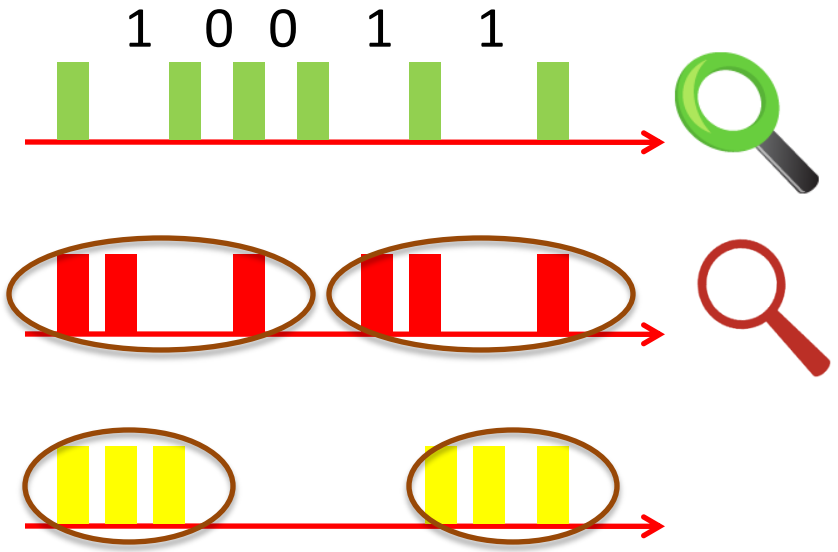
Admin



Existing detectors are channel-specific:

- Using detector A for channel B doesn't work

Problem: Making a new channel is easy!



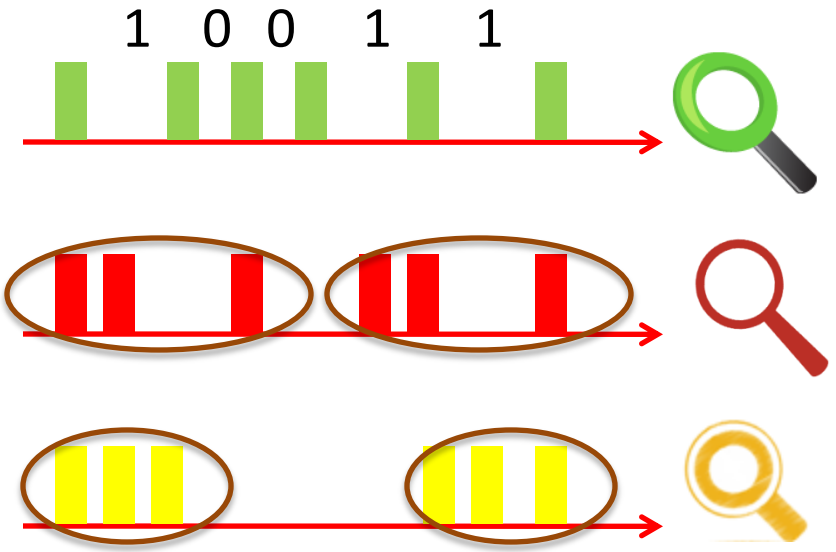
Admin



Existing detectors are channel-specific:

- Using detector A for channel B doesn't work
- Attacker can always invent a new modulation

Problem: Making a new channel is easy!

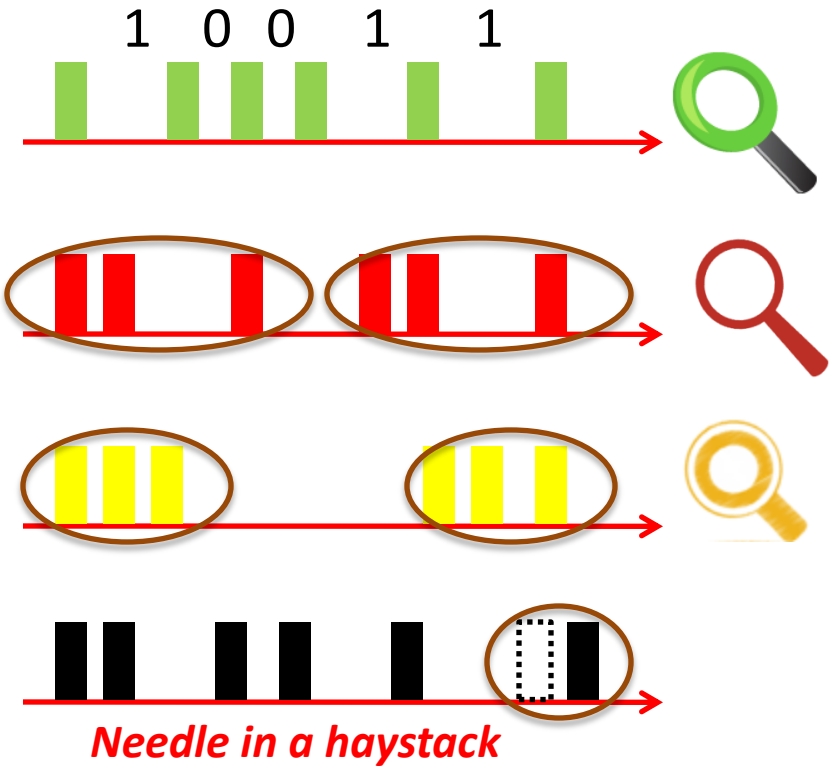




Existing detectors are channel-specific:

- Using detector A for channel B doesn't work
- Attacker can always invent a new modulation

Problem: Making a new channel is easy!



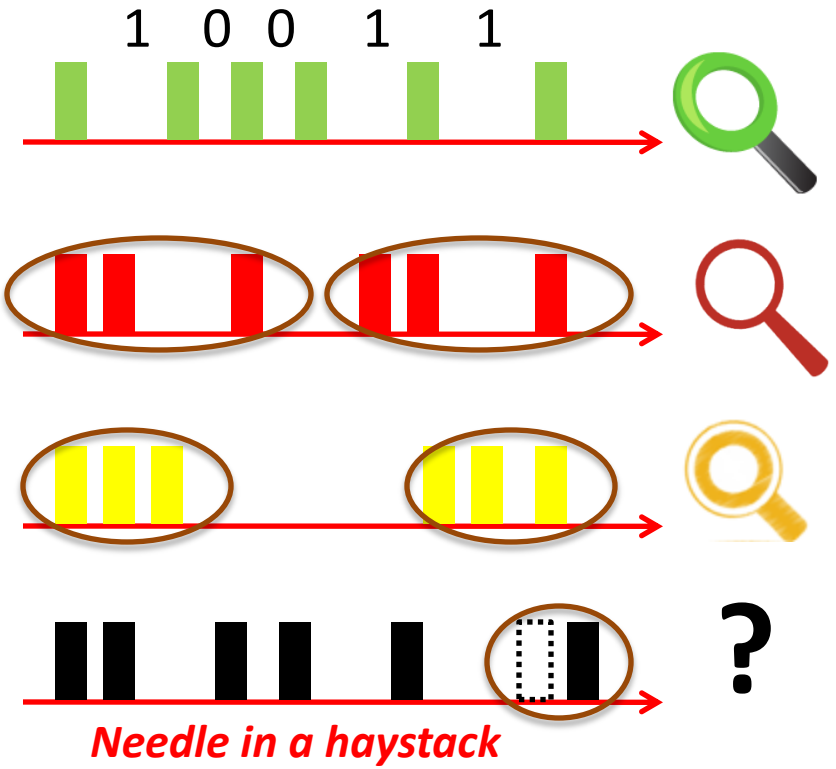
Admin



Existing detectors are channel-specific:

- Using detector A for channel B doesn't work
- Attacker can always invent a new modulation

Problem: Making a new channel is easy!

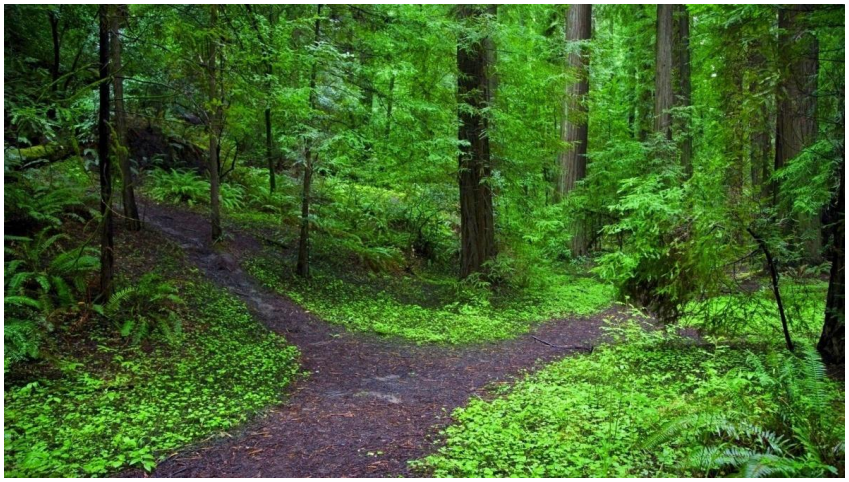




Existing detectors are channel-specific:

- Using detector A for channel B doesn't work
- Attacker can always invent a new modulation
- Low-rate channels ("Needle in a haystack") are hard to detect

Is there a different way?



Is there a different way?

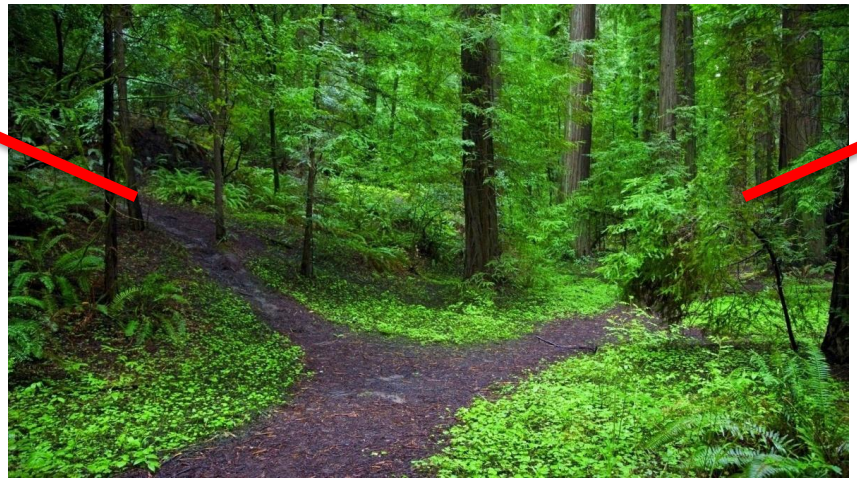


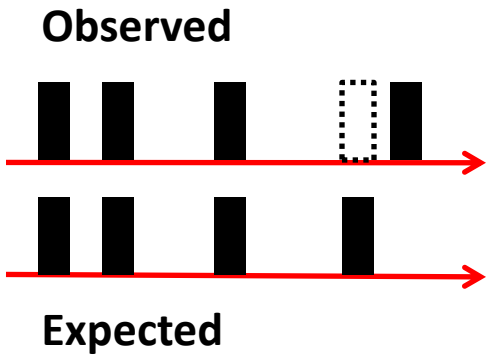


- Existing approaches detect specific anomalies

Is there a different way?



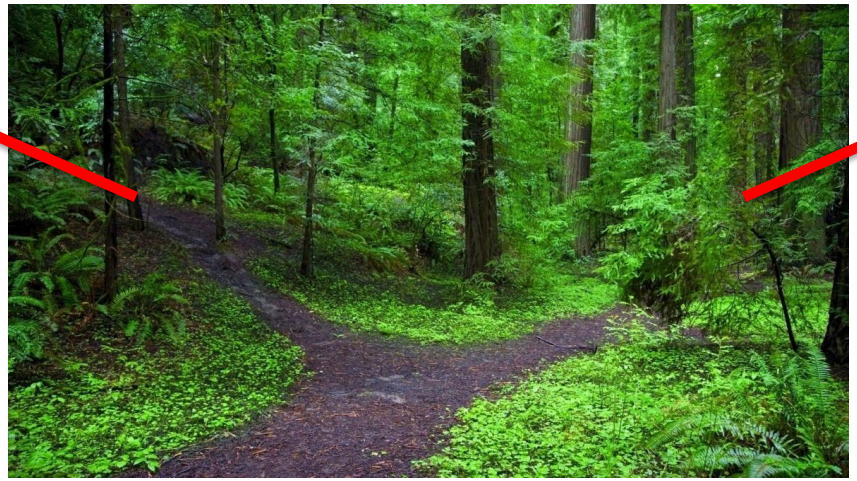


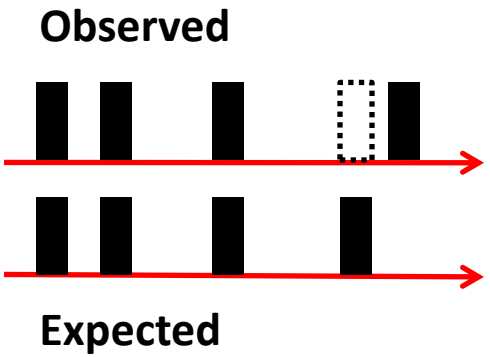


- Existing approaches detect specific anomalies
- **Our approach**: **Compare the observed timing to the expected timing**

Is there a different way?



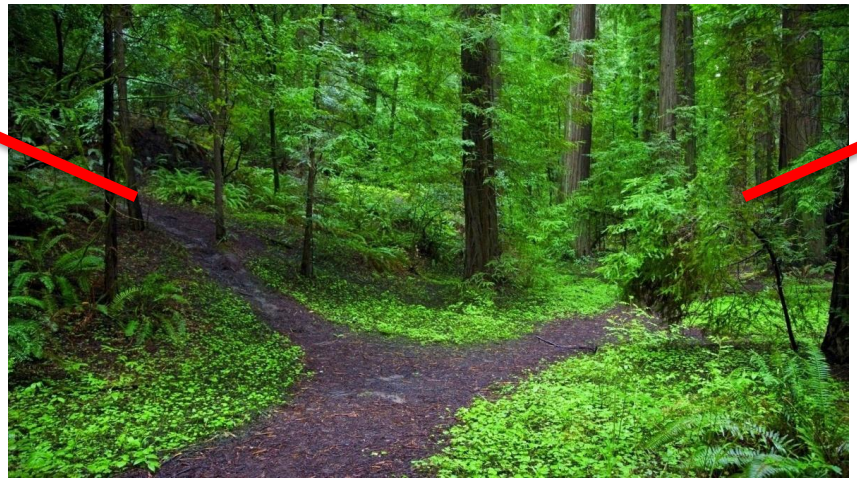


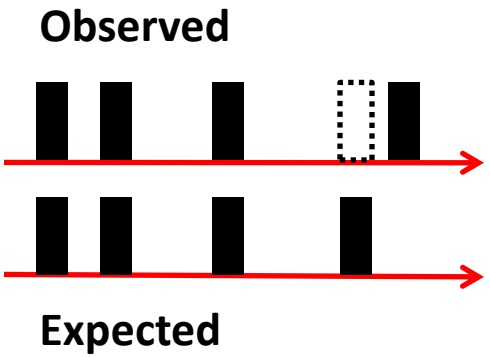


- Existing approaches detect specific anomalies
- **Our approach**: **Compare the observed timing to the expected timing**
- Works for covert timing channels in general
  → Can detect both known and unknown/novel channels!

Is there a different way?







- Existing approaches detect specific anomalies
- **Our approach**: **Compare the observed timing to the expected timing**
- Works for covert timing channels in general
  → Can detect both known and unknown/novel channels!
- But how do we know what timing we should expect?

How can we find the expected timing?

It would be difficult to **predict** the timing up front

How can we find the expected timing?

It would be difficult to **predict** the timing up front

- See, e.g., WCET analysis in real-time systems

How can we find the expected timing?

It would be difficult to **predict** the timing up front

- See, e.g., WCET analysis in real-time systems
- And WCET would only give us an upper bound - but we would need the exact timing!

How can we find the expected timing?

It would be difficult to **predict** the timing up front

- See, e.g., WCET analysis in real-time systems
- And WCET would only give us an upper bound - but we would need the exact timing!

Key insight: We only need to **reproduce** the timing!

- We can record the inputs of the machine and then replay them on a different machine!

How can we find the expected timing?

It would be difficult to **predict** the timing up front

- See, e.g., WCET analysis in real-time systems
- And WCET would only give us an upper bound - but we would need the exact timing!

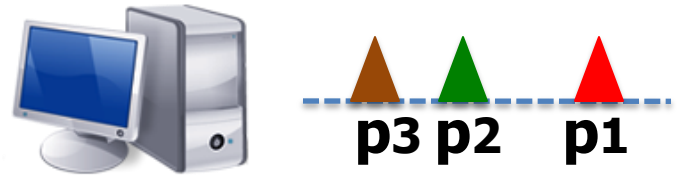
Key insight: We only need to **reproduce** the timing!

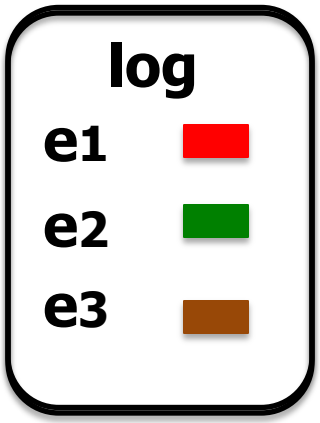
- We can record the inputs of the machine and then replay them on a different machine!
- Can we use deterministic replay to do this?

Why Deterministic Replay is not enough

Deterministic replay records and replays non-deterministic events

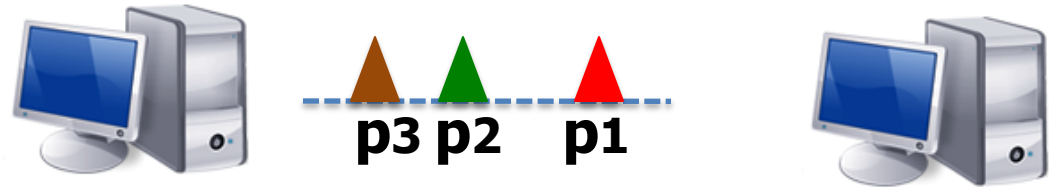
Why Deterministic Replay is not enough

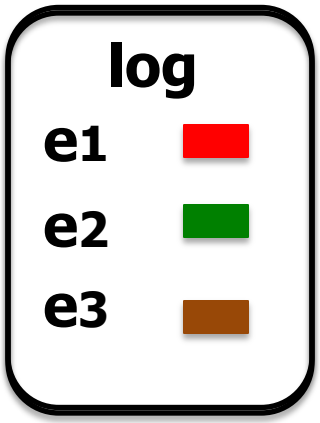




Deterministic replay records and replays non-deterministic events

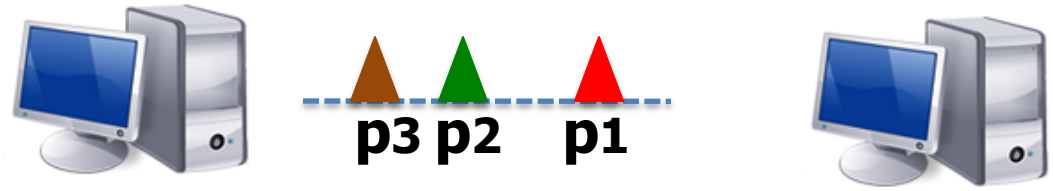
Why Deterministic Replay is not enough

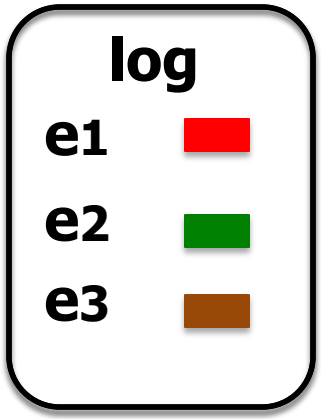




Deterministic replay records and replays non-deterministic events

Why Deterministic Replay is not enough

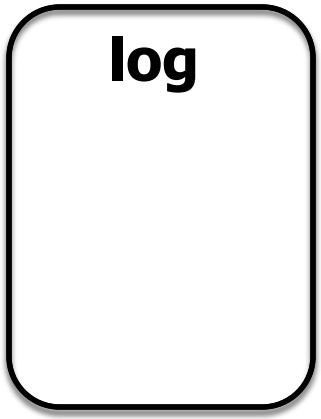




Deterministic replay records and replays non-deterministic events

Why Deterministic Replay is not enough

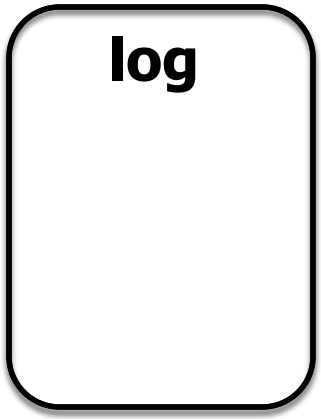




Deterministic replay records and replays non-deterministic events

Why Deterministic Replay is not enough





Deterministic replay records and replays non-deterministic events
This reproduces the **outputs** - but **not the timing**!

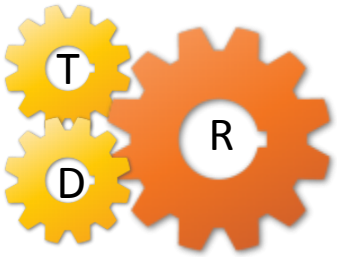
Time-deterministic replay (TDR)

Goal: Reproduce **both the outputs and the timing**

Time-deterministic replay (TDR)



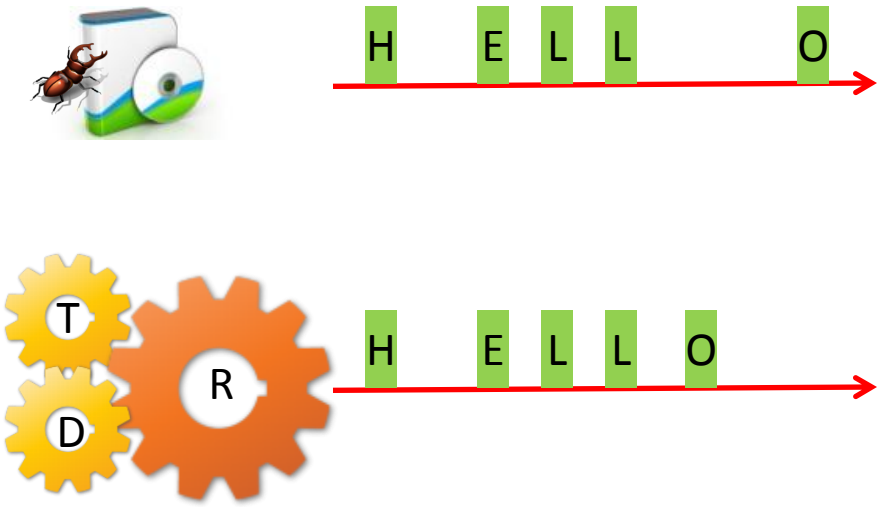




Goal: Reproduce **both the outputs and the timing**

With this, we can detect covert timing channels as follows:

Time-deterministic replay (TDR)

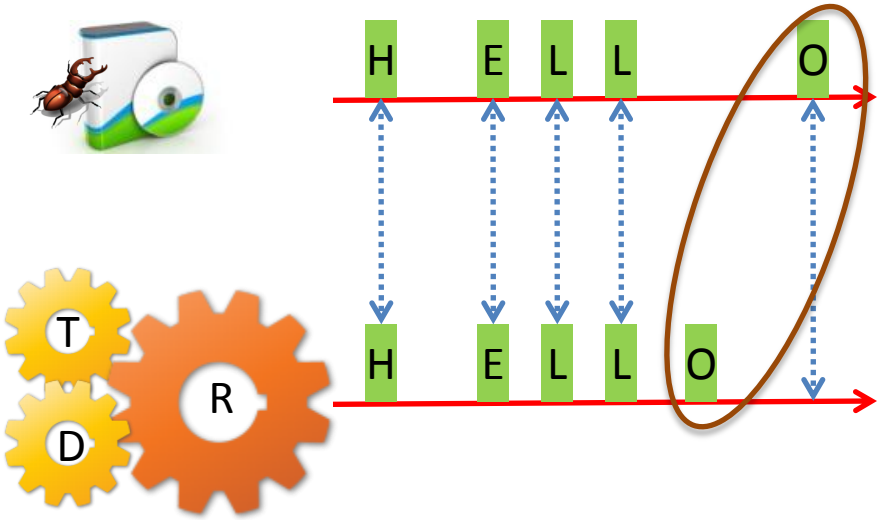


Goal: Reproduce **both the outputs and the timing**

With this, we can detect covert timing channels as follows:

(1) **Reproduce** the timing of every network output

Time-deterministic replay (TDR)



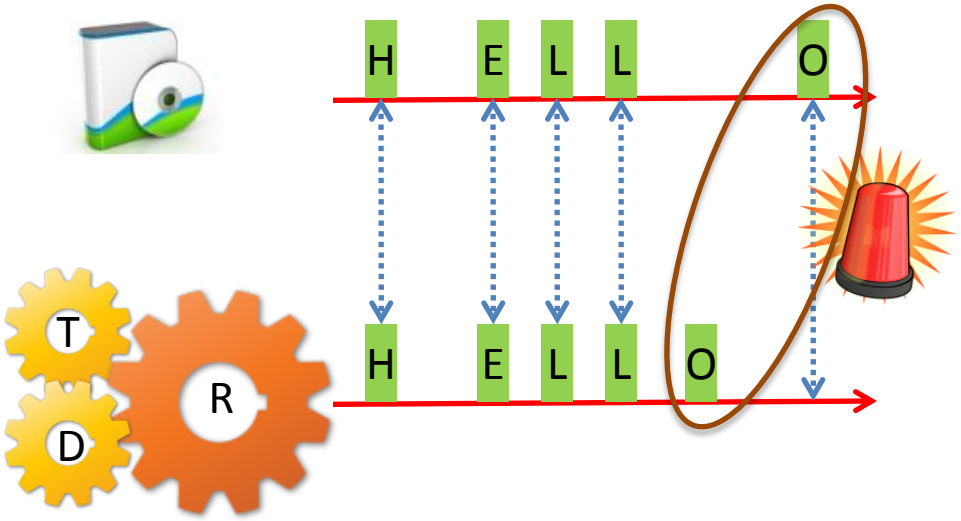
Goal: Reproduce **both the outputs and the timing**

With this, we can detect covert timing channels as follows:

(1) **Reproduce** the timing of every network output

(2) **Compare** the observed timing to the reproduced timing

Time-deterministic replay (TDR)



Goal: Reproduce **both the outputs and the timing**

With this, we can detect covert timing channels as follows:

(1) **Reproduce** the timing of every network output

(2) **Compare** the observed timing to the reproduced timing

(3) **Raise the alarm** if there is any discrepancy

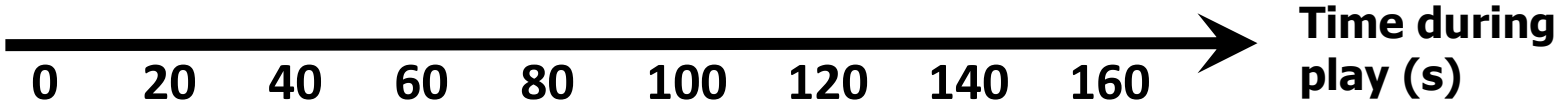
Outline

- ✓ - Motivation
- ✓ - Challenge
- - Time-deterministic replay (TDR)
  - -Deterministic replay vs. Time-deterministic replay
  - -Time noise, and how to reduce it
  - -Aligning play and replay
- - Sanity: A TDR prototype
  - -Design & Implementation
- - Evaluation
  - -Reducing time noise
  - -Aligning play and replay
  - -Detecting timing channels
- - Conclusion

Deterministic Replay is not enough

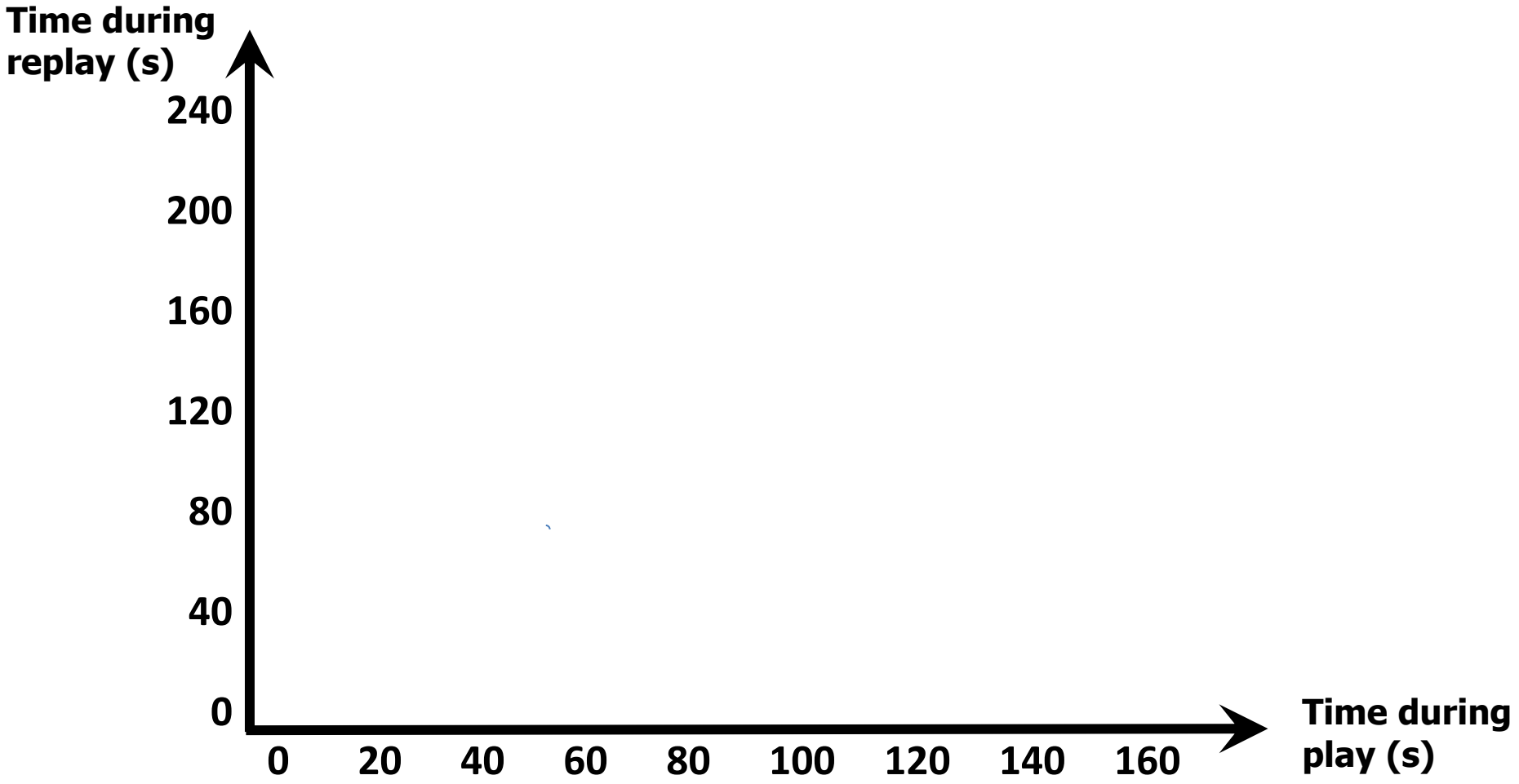
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



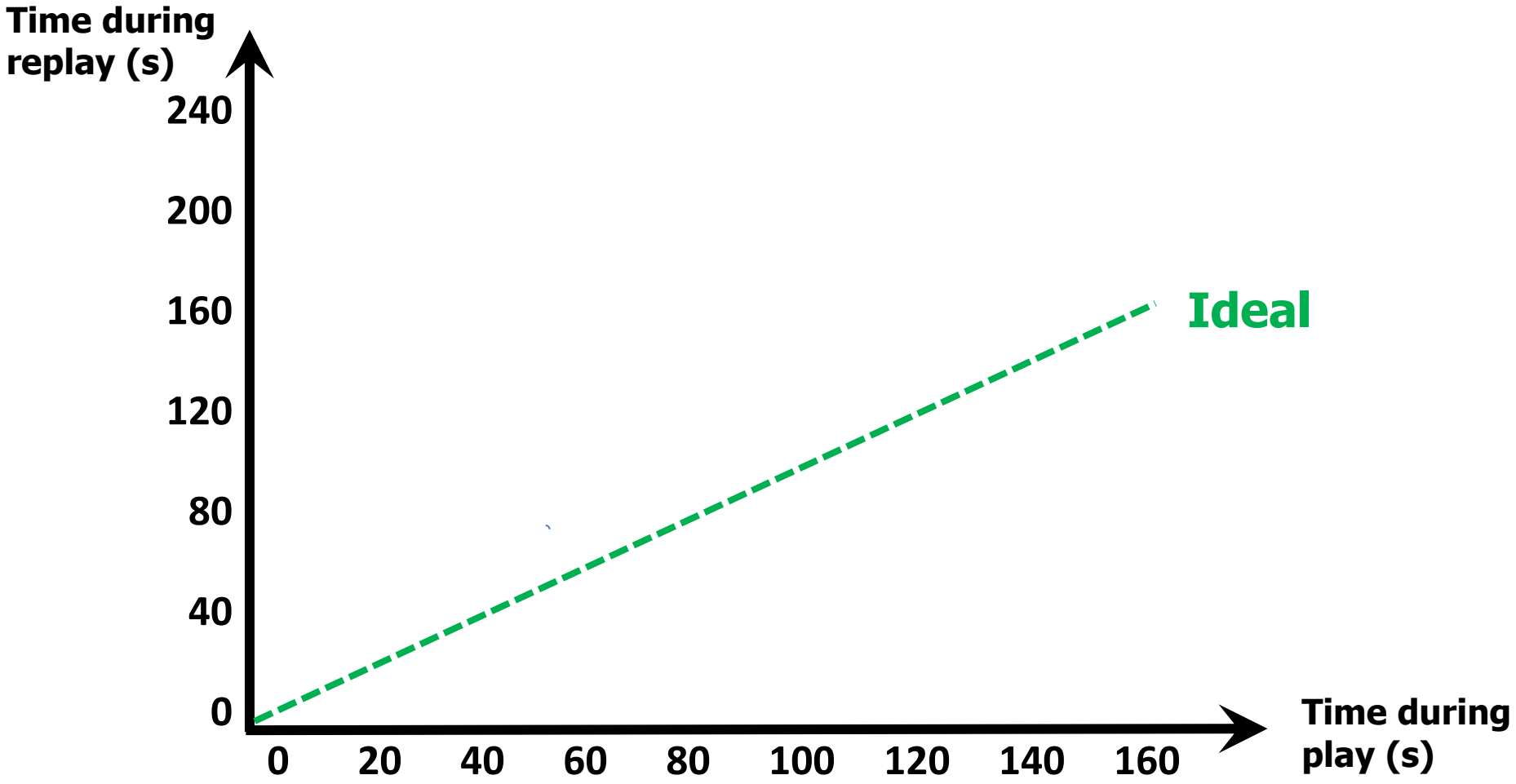
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



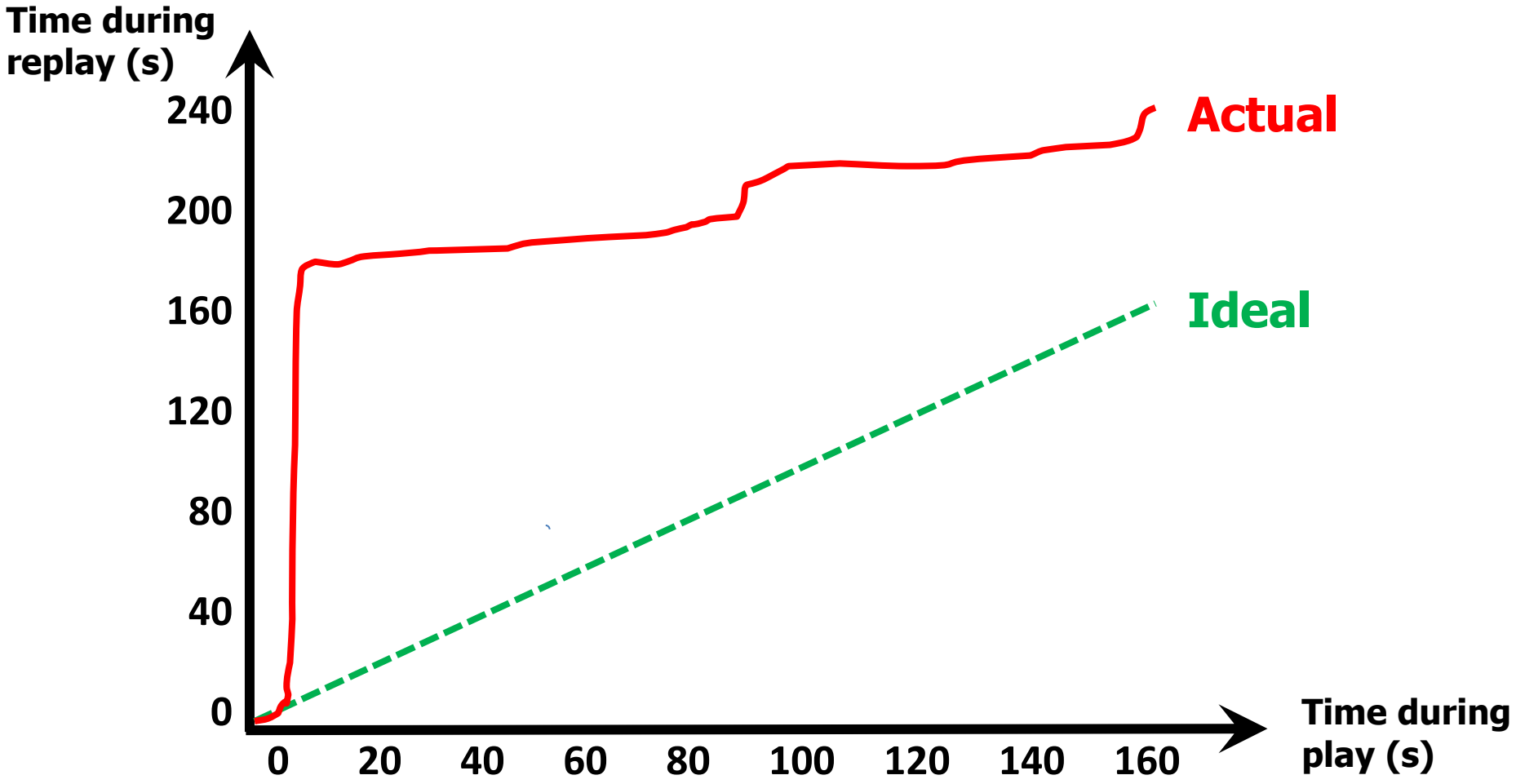
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



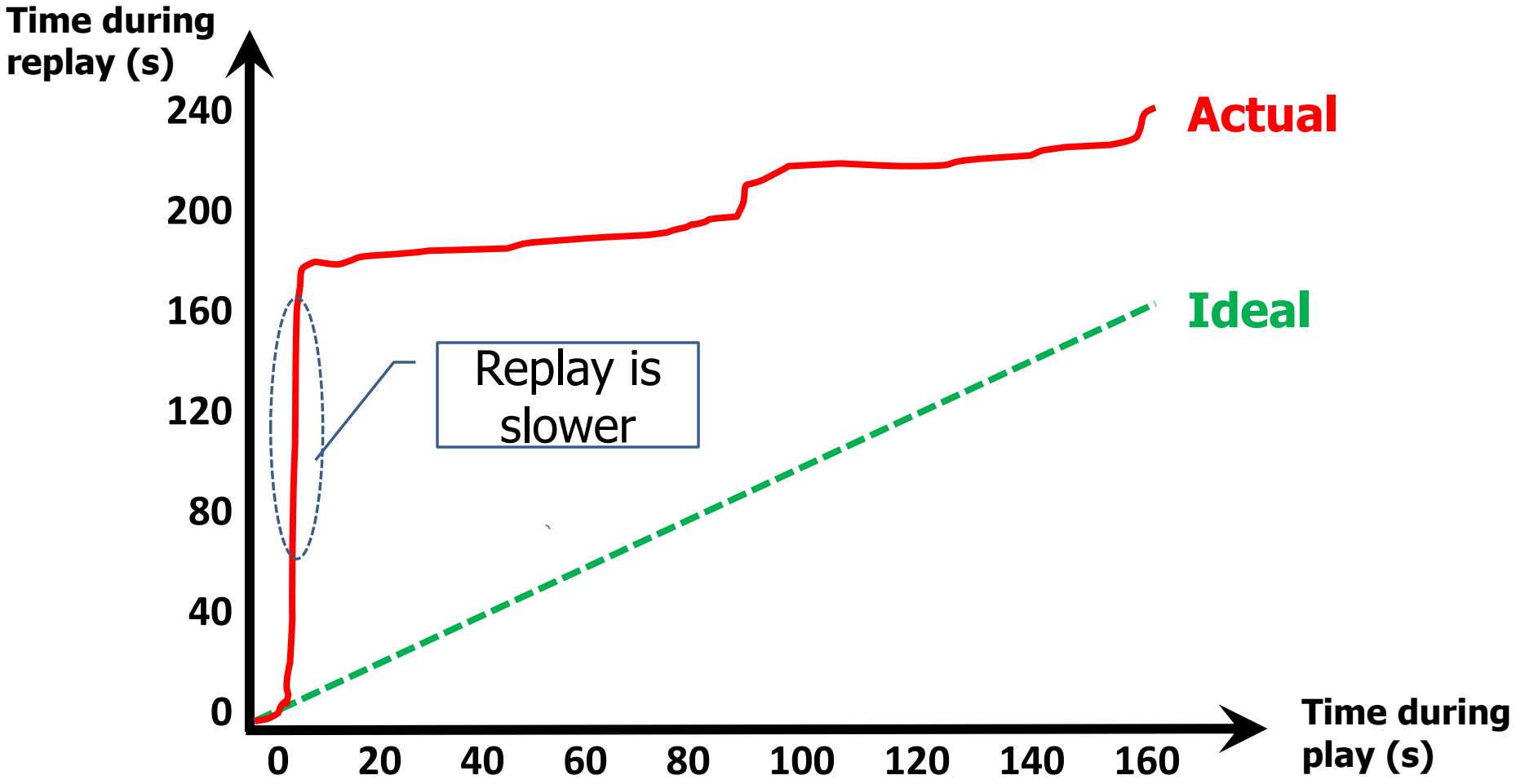
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



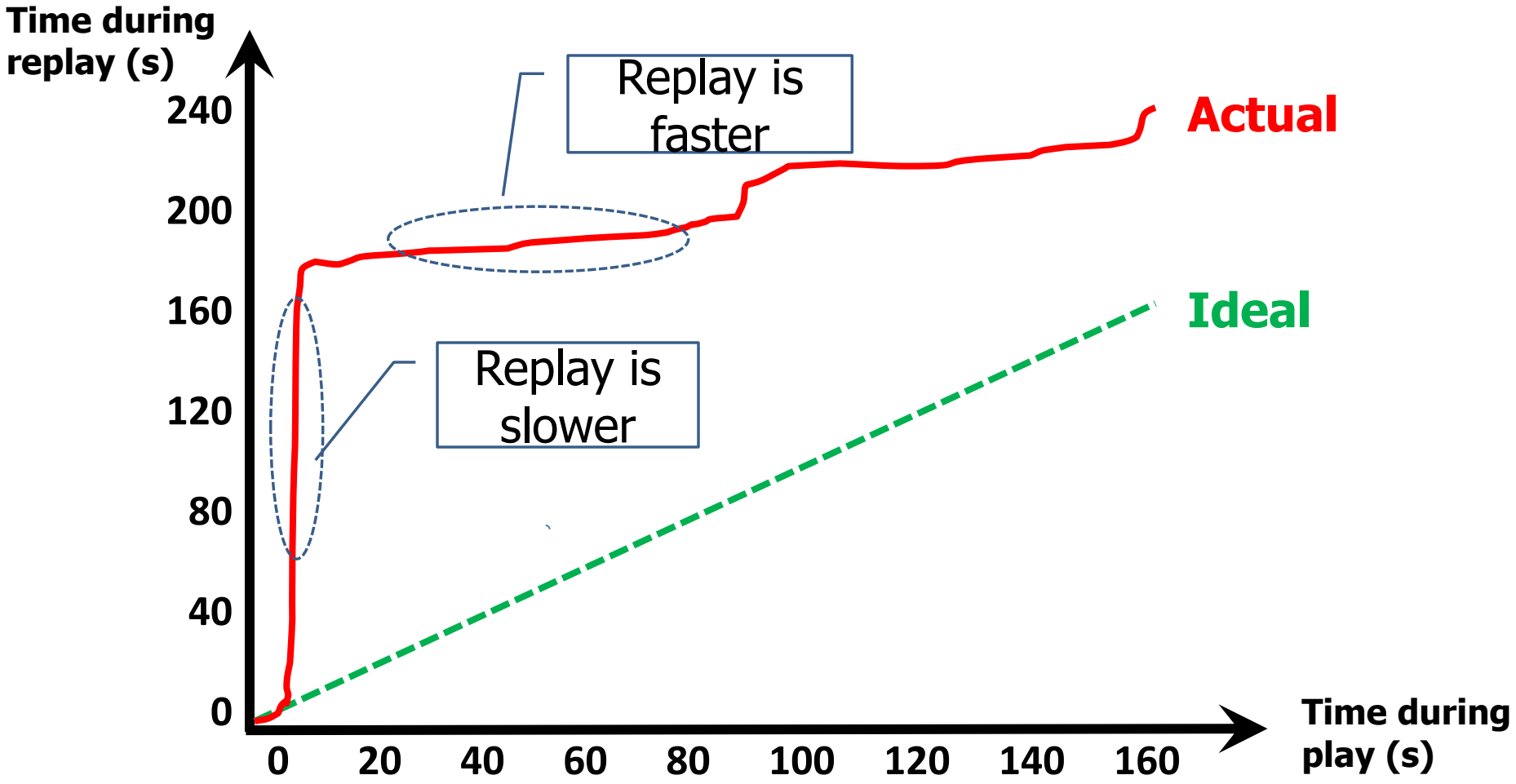
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



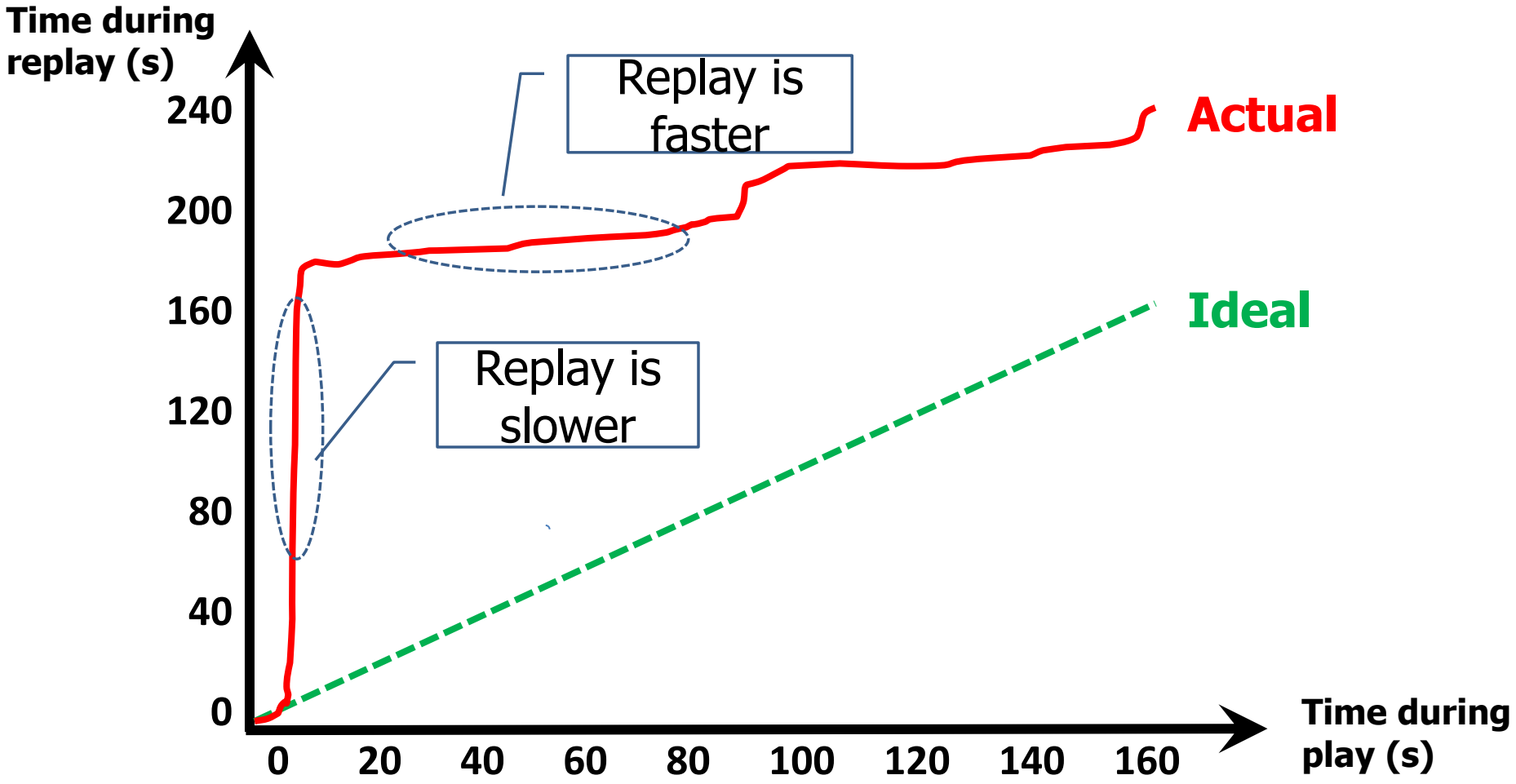
- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)

Deterministic Replay is not enough



- Experiment: Record and replay an HTTP server in an existing VMM with deterministic replay (XenTT)
- Result: Play and replay take widely different amounts of time

What is causing this discrepancy?

There are many different sources of timing variation ("time noise"), such as:



Different memory allocations and cache behavior



IRQs and system calls take different amounts of time



Disk accesses take different amounts of time



Kernel may interfere with execution or cache content



CPU features, such as frequency scaling and speculation

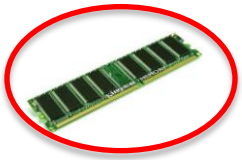


Non-deterministic scheduling decisions

See paper for details

What is causing this discrepancy?

There are many different sources of timing variation ("time noise"), such as:



Different memory allocations and cache behavior



IRQs and system calls take different amounts of time



Disk accesses take different amounts of time



Kernel may interfere with execution or cache content



CPU features, such as frequency scaling and speculation



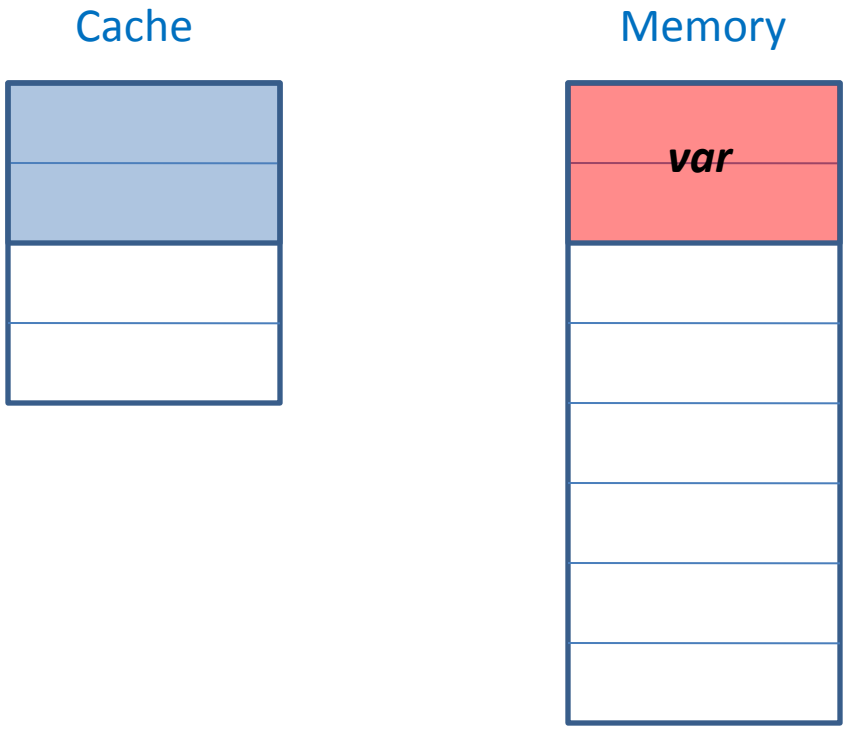
Non-deterministic scheduling decisions

See paper for details

Example: Controlling time noise from memory

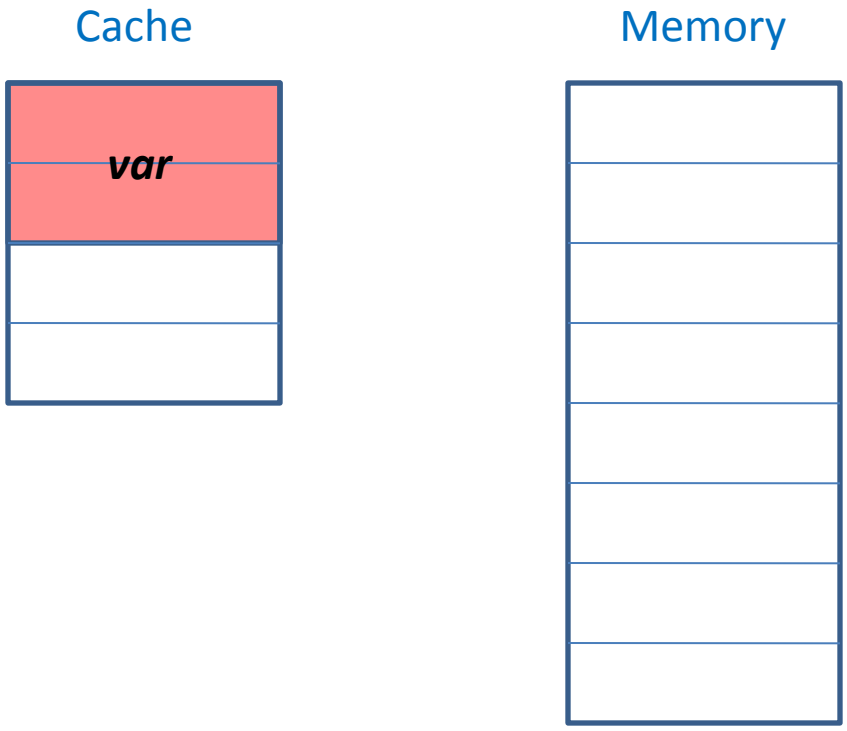
Problem: Different cache behaviors and memory locations during play and replay.

Example: Controlling time noise from memory



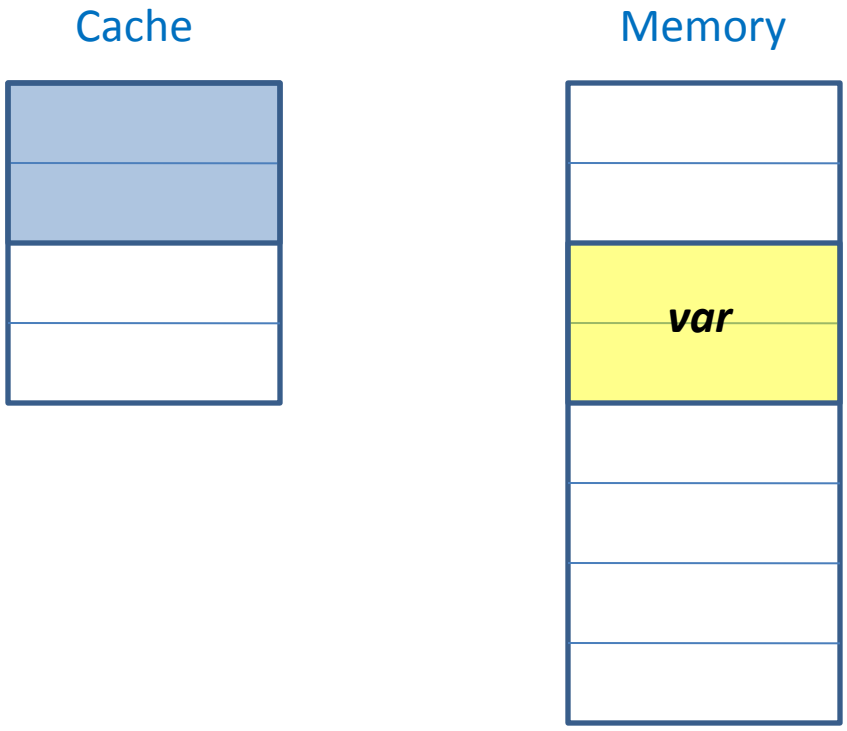
Problem: Different cache behaviors and memory locations during play and replay.

Example: Controlling time noise from memory



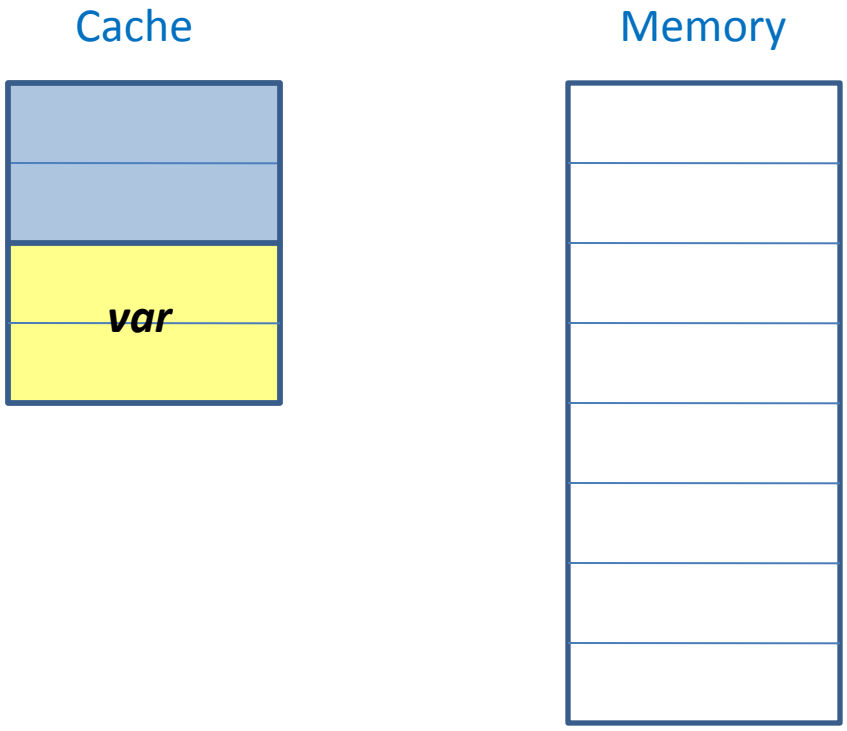
Problem: Different cache behaviors and memory locations during play and replay.

Example: Controlling time noise from memory



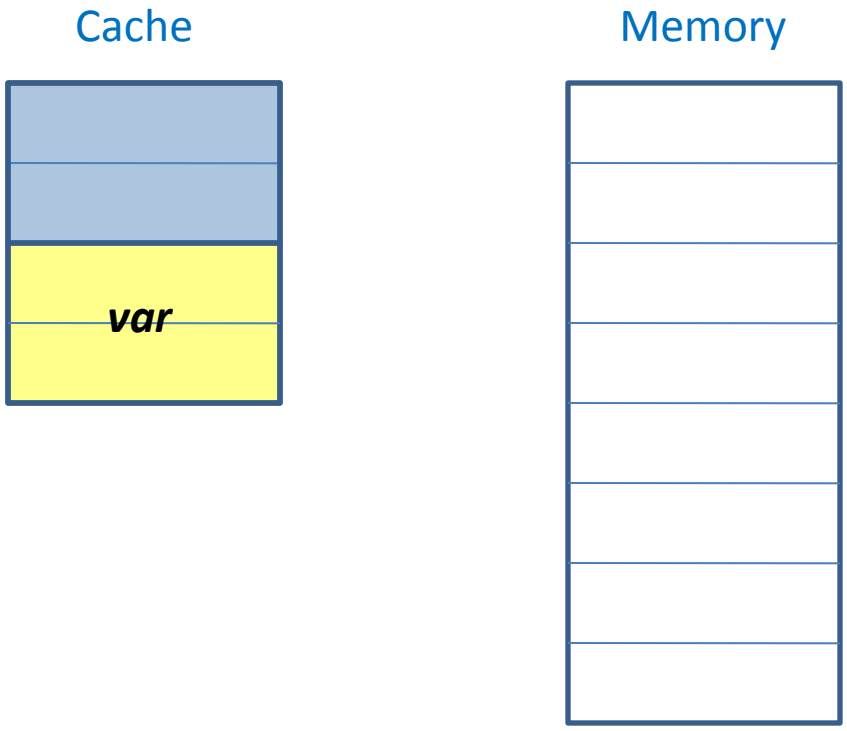
Problem: Different cache behaviors and memory locations during play and replay.

Example: Controlling time noise from memory



Problem: Different cache behaviors and memory locations during play and replay.

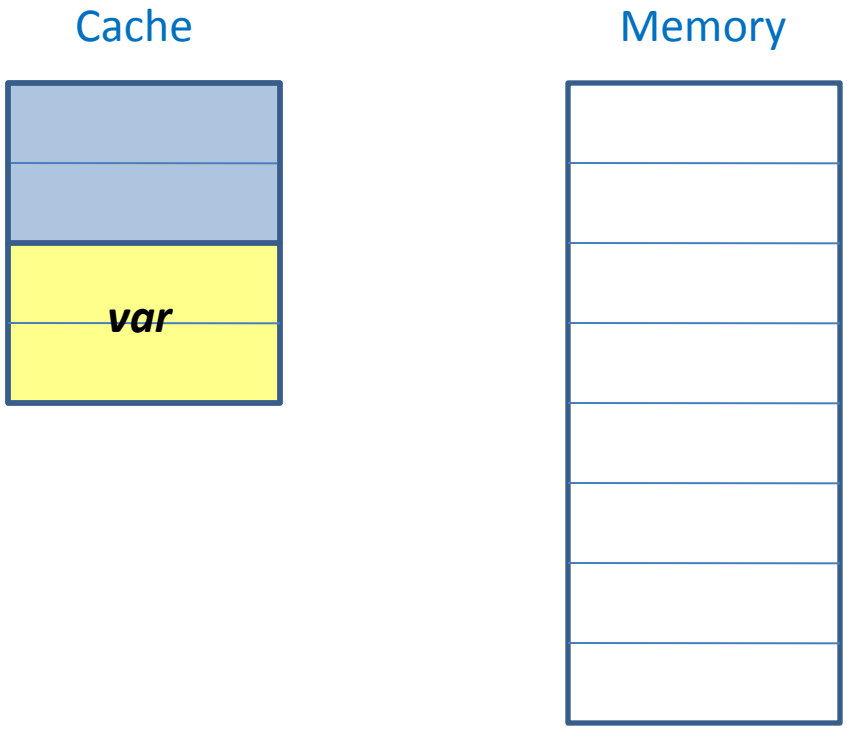
Example: Controlling time noise from memory



Problem: Different cache behaviors and memory locations during play and replay.

Solution: (1) Manage all memory allocations

Example: Controlling time noise from memory



Problem: Different cache behaviors and memory locations during play and replay.

Solution: (1) Manage all memory allocations

(2) Flush cache before execution

Techniques for mitigating time noise

Noise source	Mitigation technique(s) used	Effect
Divergence	Deterministic replay [19]	Eliminated
Randomness	Avoid or log random decisions	Eliminated
Scheduler	Deterministic multithreading [38]	Eliminated
Interrupts	Handle interrupts on a separate core	Reduced
Play vs. replay	Align JVM's control flow and memory accesses during play and replay	Eliminated
Caches	Flush caches at the beginning; use the same physical frames	Reduced
Paging	All memory is pinned and managed by JVM	Eliminated
I/O	Pad variable-time operations; use SSDs instead of HDDs	Reduced
Preemption	Run in kernel mode; do not share core with other apps	Eliminated
CPU features	Disable timing-relevant CPU features, such as frequency scaling	Reduced

Techniques for mitigating time noise

Noise source	Mitigation technique(s) used	Effect
Divergence	Deterministic replay [19]	Eliminated
Randomness	Avoid or log random decisions	Eliminated
Scheduler	Deterministic multithreading [38]	Eliminated
Interrupts	Handle interrupts on a separate core	Reduced
Play vs. replay	Align JVM's control flow and memory accesses during play and replay	Eliminated
Caches	Flush caches at the beginning; use the same physical frames	Reduced
Paging	All memory is pinned and managed by JVM	Eliminated
I/O	Pad variable-time operations; use SSDs instead of HDDs	Reduced
Preemption	Run in kernel mode; do not share core with other apps	Eliminated
CPU features	Disable timing-relevant CPU features, such as frequency scaling	Reduced

Techniques for mitigating time noise

Noise source	Mitigation technique(s) used	Effect
Divergence	Deterministic replay [19]	Eliminated
Randomness	Avoid or log random decisions	Eliminated
Scheduler	Deterministic multithreading [38]	Eliminated
Interrupts	Handle interrupts on a separate core	Reduced
Play vs. replay	Align JVM's control flow and memory accesses during play and replay	Eliminated
Caches	Flush caches at the beginning; use the same physical frames	Reduced
Paging	All memory is pinned and managed by JVM	Eliminated
I/O	Pad variable-time operations; use SSDs instead of HDDs	Reduced
Preemption	Run in kernel mode; do not share core with other apps	Eliminated
CPU features	Disable timing-relevant CPU features, such as frequency scaling	Reduced

Techniques for mitigating time noise

Noise source	Mitigation technique(s) used	Effect
Divergence	Deterministic replay [19]	Eliminated
Randomness	Avoid or log random decisions	Eliminated
Scheduler	Deterministic multithreading [38]	Eliminated
Interrupts	Handle interrupts on a separate core	Reduced
Play vs. replay	Align JVM's control flow and memory accesses during play and replay	Eliminated
Caches	Flush caches at the beginning; use the same physical frames	Reduced
Paging	All memory is pinned and managed by JVM	Eliminated
I/O	Pad variable-time operations; use SSDs instead of HDDs	Reduced
Preemption	Run in kernel mode; do not share core with other apps	Eliminated
CPU features	Disable timing-relevant CPU features, such as frequency scaling	Reduced

- Not all sources of time noise can be eliminated on commodity hardware (e.g., speculation)

Techniques for mitigating time noise

Noise source	Mitigation technique(s) used	Effect
Divergence	Deterministic replay [19]	Eliminated
Randomness	Avoid or log random decisions	Eliminated
Scheduler	Deterministic multithreading [38]	Eliminated
Interrupts	Handle interrupts on a separate core	Reduced
Play vs. replay	Align JVM's control flow and memory accesses during play and replay	Eliminated
Caches	Flush caches at the beginning; use the same physical frames	Reduced
Paging	All memory is pinned and managed by JVM	Eliminated
I/O	Pad variable-time operations; use SSDs instead of HDDs	Reduced
Preemption	Run in kernel mode; do not share core with other apps	Eliminated
CPU features	Disable timing-relevant CPU features, such as frequency scaling	Reduced

- Not all sources of time noise can be eliminated on commodity hardware (e.g., speculation)
- But we can still achieve a very low noise level

Problem: Play and replay have different logic

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay have different logic

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Problem: Play and replay have different logic

```
void accessInt(int *value, int *buf) {
   if (isPlay)
      *buf = *value;
   else
      *value  = *buf;
}
```

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Problem: Play and replay have different logic

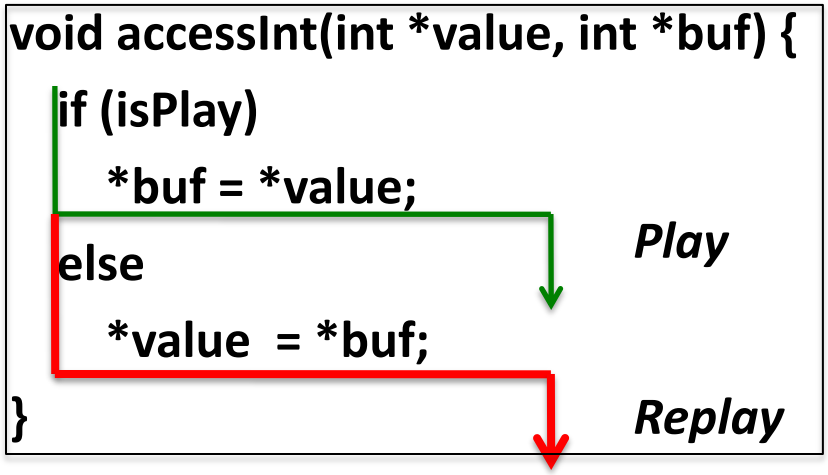
```
void accessInt(int *value, int *buf) {
   if (isPlay)
      *buf = *value;
   else
      *value  = *buf;
}
```



Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

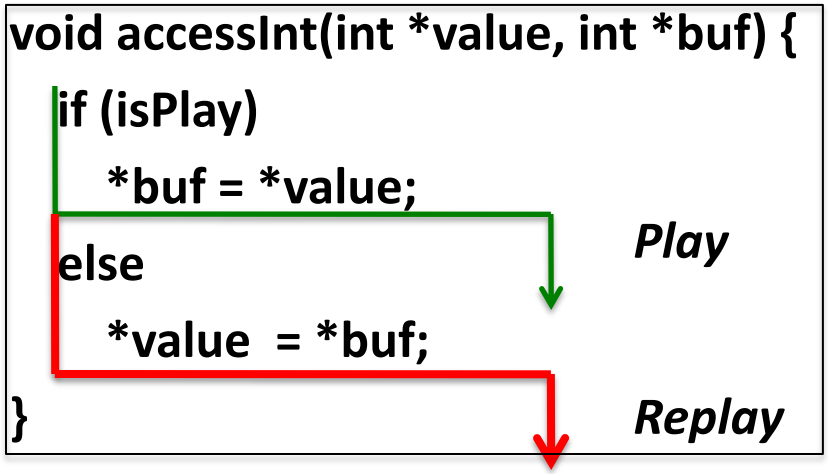
Problem: Play and replay have different logic



Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Problem: Play and replay have different logic



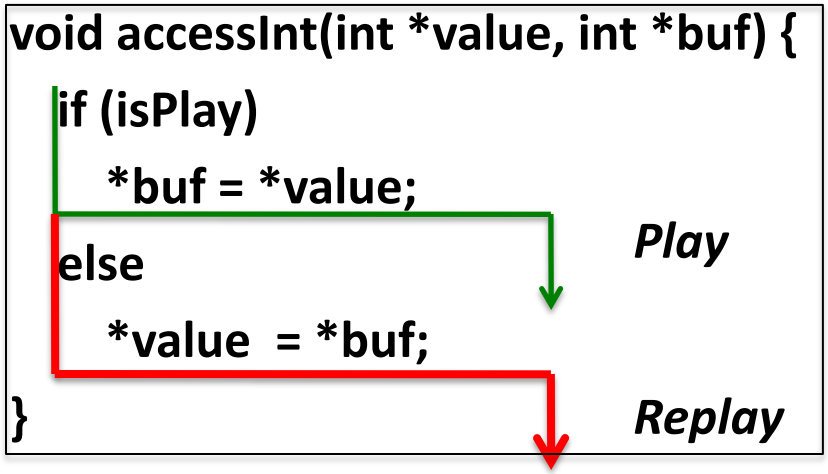
Different memory access patterns

Different branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Problem: Play and replay have different logic



Different memory access patterns

Different branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Solution: Carefully design the code to **align play and replay**

Problem: Play and replay have different logic

```
void accessInt(int *value, int *buf) {
   if (isPlay)
      *buf = *value;
   else
      *value  = *buf;
}
```

Play

Replay

```
void accessInt(int *value, int *buf) {
   int temp = (*value) & playMask;
   temp = temp | (*buf &
~playMask);
   *value = *buf = temp;
}
```

Different memory access patterns

Different branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Solution: Carefully design the code to **align play and replay**

Problem: Play and replay have different logic

```
void accessInt(int *value, int *buf) {
  if (isPlay)
    *buf = *value;
  else
    *value  = *buf;
}
```

Play

Replay

```
void accessInt(int *value, int *buf) {
  int temp = (*value) & playMask;
  temp = temp | (*buf &
~playMask);
  *value = *buf = temp;
}
```

Play

Different memory access patterns

Different branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Solution: Carefully design the code to **align play and replay**

Problem: Play and replay have different logic

```
void accessInt(int *value, int *buf) {
  if (isPlay)
    *buf = *value;
  else
    *value  = *buf;
}
```

Play

Replay

```
void accessInt(int *value, int *buf) {
  int temp = (*value) & playMask;
  temp = temp | (*buf &
~playMask);
  *value = *buf = temp;
}
```

Play

Replay

Different memory access patterns

Different branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Solution: Carefully design the code to **align play and replay**

Problem: Play and replay have different logic

```
void accessInt(int *value, int *buf) {
  if (isPlay)
    *buf = *value;
  else
    *value  = *buf;
}
```

Play

Replay

Different memory access patterns

Different branches taken

```
void accessInt(int *value, int *buf) {
  int temp = (*value) & playMask;
  temp = temp | (*buf &
~playMask);
  *value = *buf = temp;
}
```

Play

Replay

Same memory access pattern

No branches taken

Would deterministic hardware, e.g., a PRET machine (Edwards and Lee, 2007), solve all our problems?

Problem: Play and replay involve different operations

Solution: Carefully design the code to **align play and replay**

Outline

- ✓ - Motivation
- ✓ - Challenge
- ✓ - Time-deterministic replay (TDR)
  - ✓ -Deterministic replay vs. Time-deterministic replay
  - ✓ -Time noise, and how to reduce it
  - ✓ -Aligning play and replay
- - Sanity: A TDR prototype
  - -Design & Implementation
- - Evaluation
  - -Reducing time noise
  - -Aligning play and replay
  - -Detecting timing channels
- - Conclusion

Prototype implementation: Sanity

- Ideal: Implement TDR in an existing VMM, such as Xen

Prototype implementation: Sanity

- Ideal: Implement TDR in an existing VMM, such as Xen
- But time-determinism is difficult to add to existing codebases
  - Reason: Complex interactions between unrelated functions, e.g., through the cache

Prototype implementation: Sanity

- Ideal: Implement TDR in an existing VMM, such as Xen
- But time-determinism is difficult to add to existing codebases
  - Reason: Complex interactions between unrelated functions, e.g., through the cache
- Our approach: Build a VMM from scratch
  - Chose Java VM because of its simplicity

Prototype implementation: Sanity

- Ideal: Implement TDR in an existing VMM, such as Xen
- But time-determinism is difficult to add to existing codebases
  - Reason: Complex interactions between unrelated functions, e.g., through the cache
- Our approach: Build a VMM from scratch
  - Chose Java VM because of its simplicity
  - No advanced features yet (e.g., no JIT)
    → Can't expect to compete with Oracle JVM

Prototype implementation: Sanity

- Ideal: Implement TDR in an existing VMM, such as Xen
- But time-determinism is difficult to add to existing codebases
  - Reason: Complex interactions between unrelated functions, e.g., through the cache
- Our approach: Build a VMM from scratch
  - Chose Java VM because of its simplicity
  - No advanced features yet (e.g., no JIT)
    → Can't expect to compete with Oracle JVM
- We rely on the Linux kernel for device I/O (e.g., network)
- Sanity is implemented as a kernel module

How Sanity shields itself from the kernel

How Sanity shields itself from the kernel



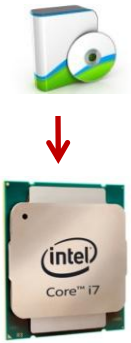
Timed core



Support core

- To avoid interference from the kernel, we run the TDR engine on a separate core

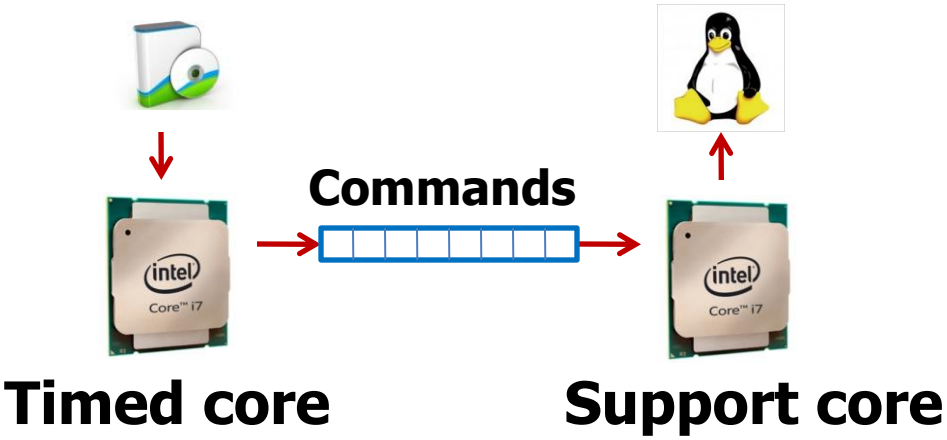
How Sanity shields itself from the kernel



Timed core **Support core**

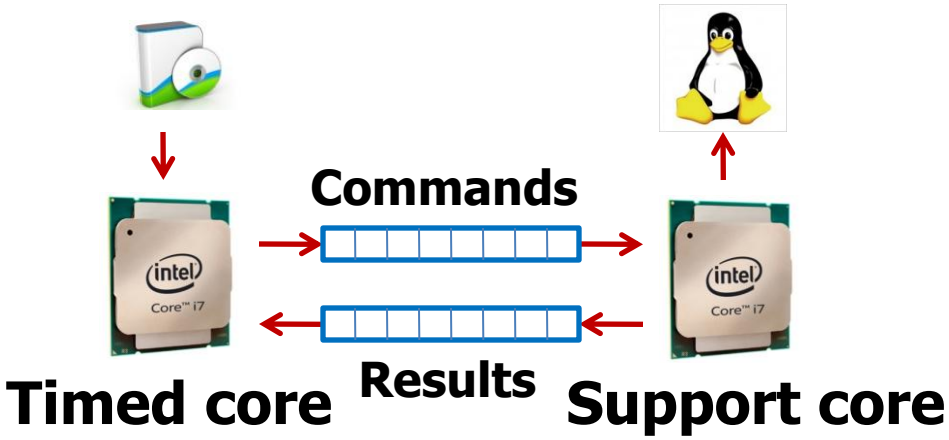
- To avoid interference from the kernel, we run the TDR engine on a separate core

How Sanity shields itself from the kernel



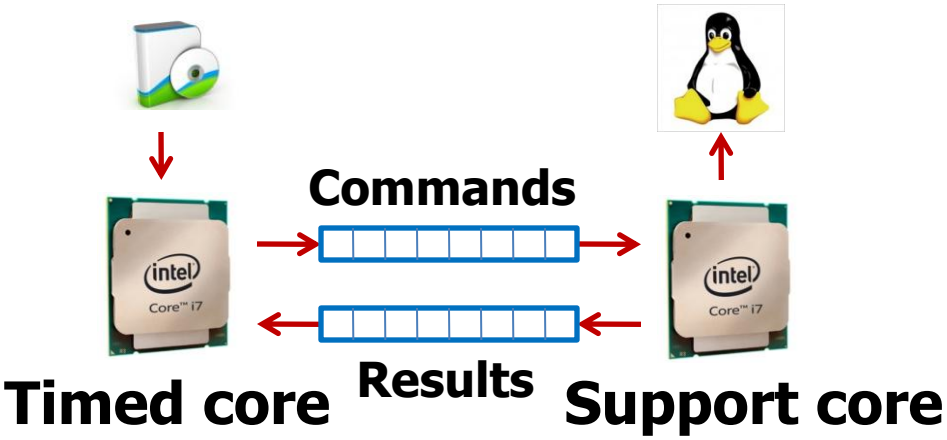
- To avoid interference from the kernel, we run the TDR engine on a separate core

How Sanity shields itself from the kernel



- To avoid interference from the kernel, we run the TDR engine on a separate core

How Sanity shields itself from the kernel



- To avoid interference from the kernel, we run the TDR engine on a separate core
- Limitation: Need two cores to do the work of one

Outline

- ✓ - Motivation
- ✓ - Challenge
- ✓ - Time-deterministic replay (TDR)
  - ✓ -Deterministic replay vs. Time-deterministic replay
  - ✓ -Time noise, and how to reduce it
  - ✓ -Aligning play and replay
- ✓ - Sanity: A TDR prototype
  - ✓ -Design & Implementation
- - Evaluation
  - -Reducing time noise
  - -Aligning play and replay
  - -Detecting timing channels
- - Conclusion

Evaluation: Overview

Q1: How well can Sanity reduce time noise?

Q2: How well can Sanity align play and replay?

Q3: How fast is Sanity?

Q4: How large is Sanity's log?

Q5: How well can Sanity detect covert timing channels?

Evaluation: Overview

- ✓ Q1: How well can Sanity reduce time noise?
- ✓ Q2: How well can Sanity align play and replay?
- Q3: How fast is Sanity?
- Q4: How large is Sanity's log?
- ✓ Q5: How well can Sanity detect covert timing channels?

Experimental setup

- Experiments were run on a Dell Optiplex 9020 workstation (3.4GHz Intel i7-4770 CPU, 16 GB RAM, 128GB Vector ZDO SSD, Linux 3.12)

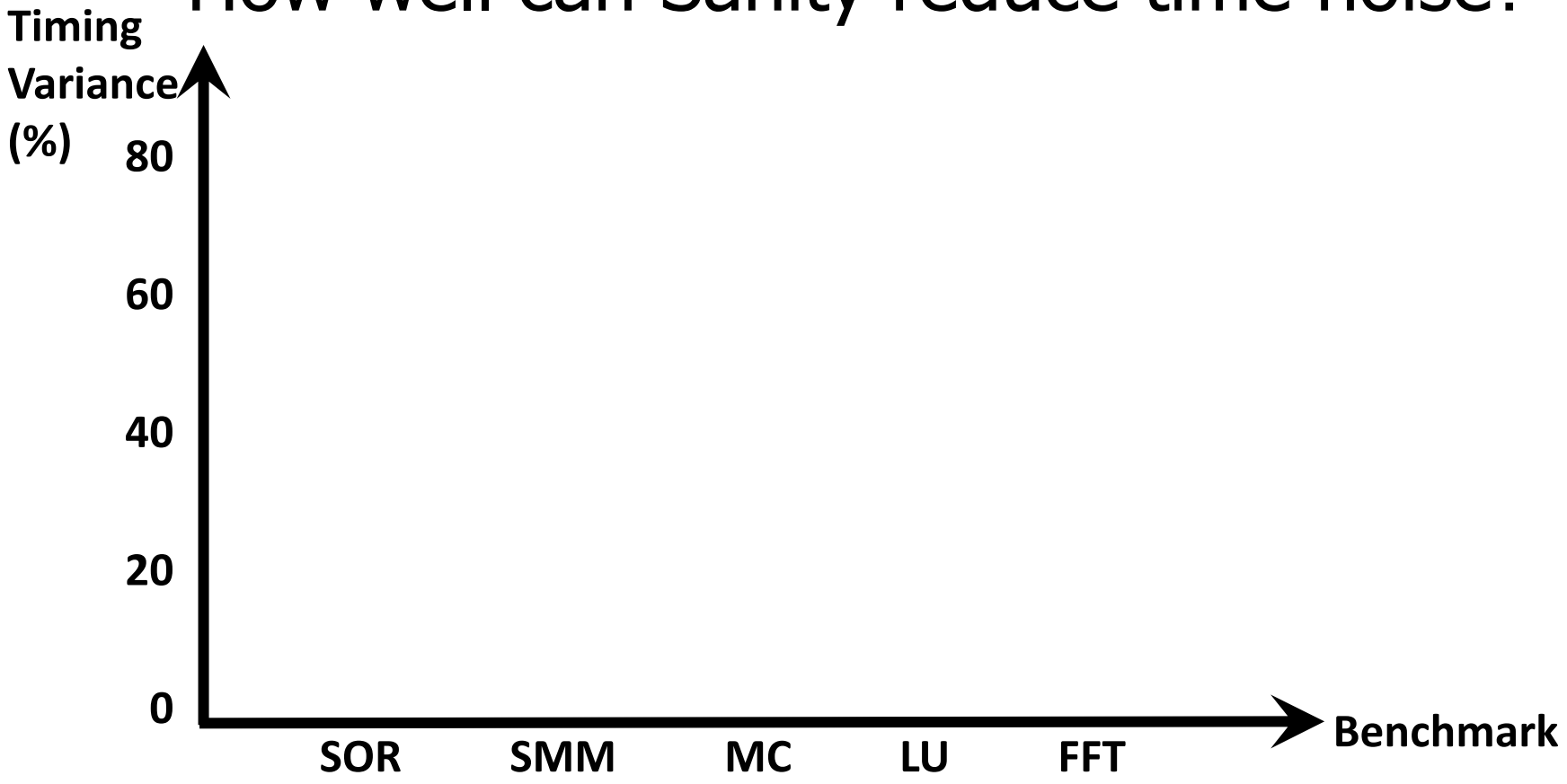
- We use two applications:
  - SciMark2 (computation-intensive benchmark)
  - nfsj: Open-source NFS server

- Baseline: Oracle's SE 7u51 JVM

How well can Sanity reduce time noise?

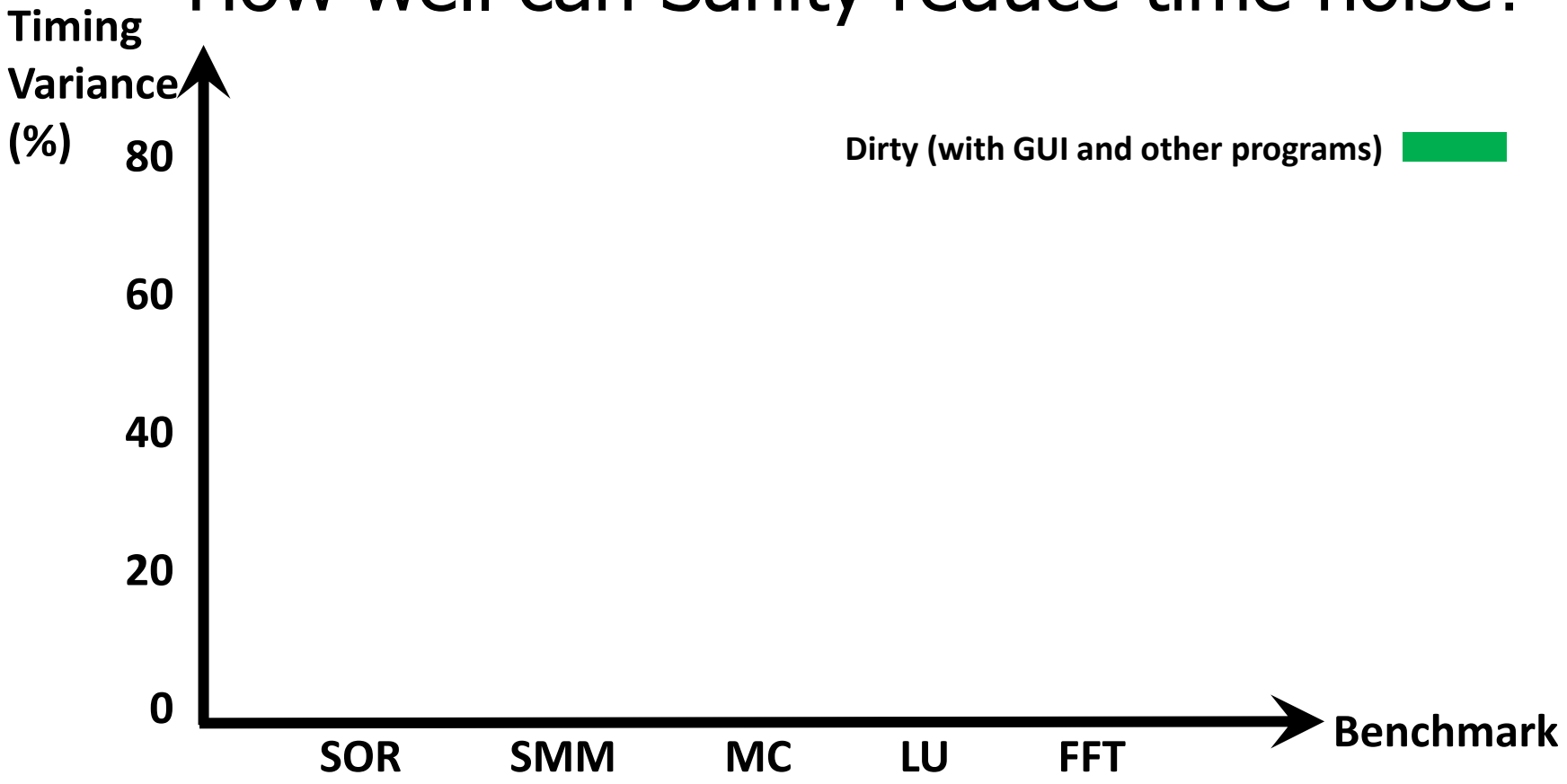
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



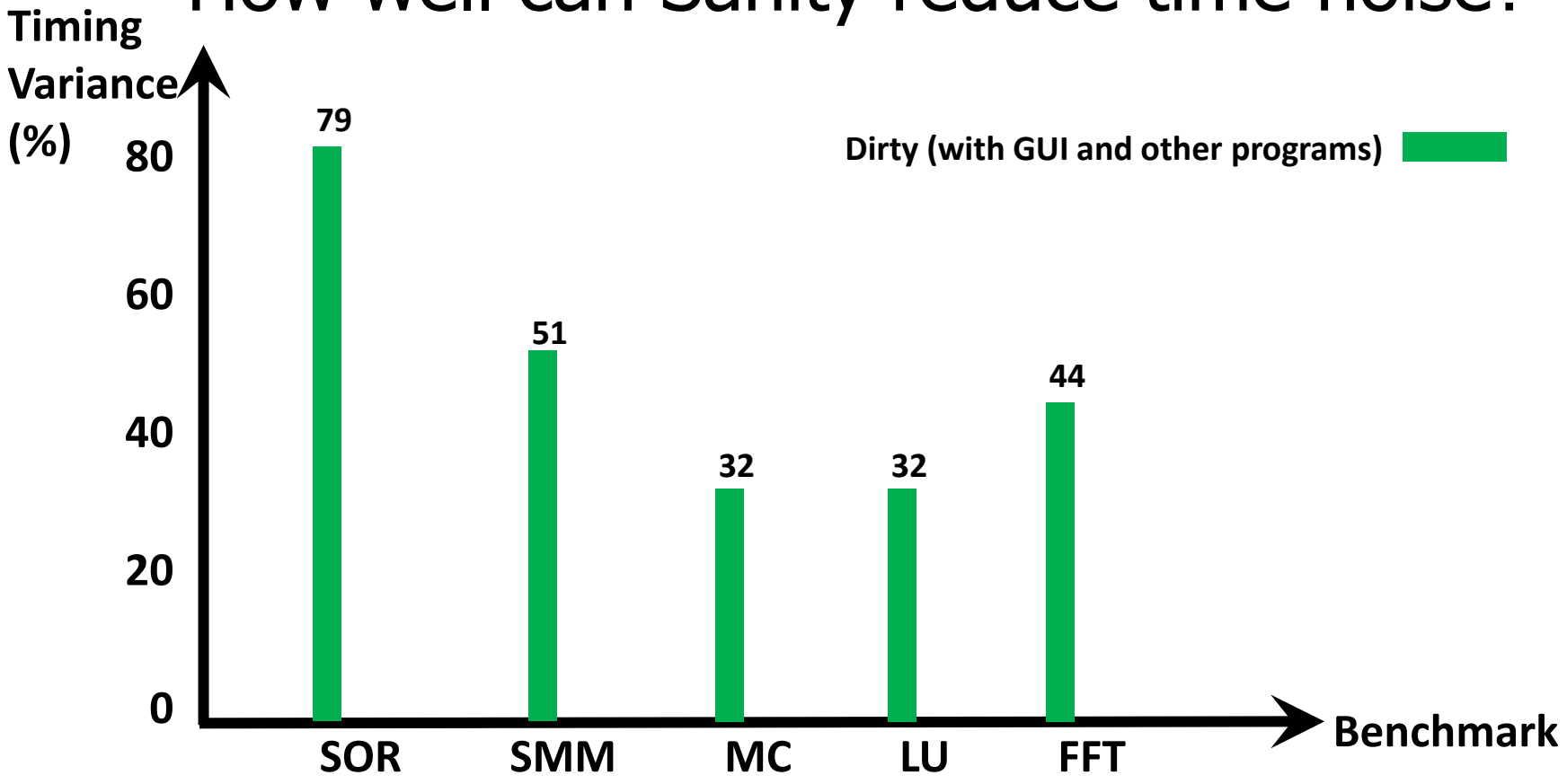
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



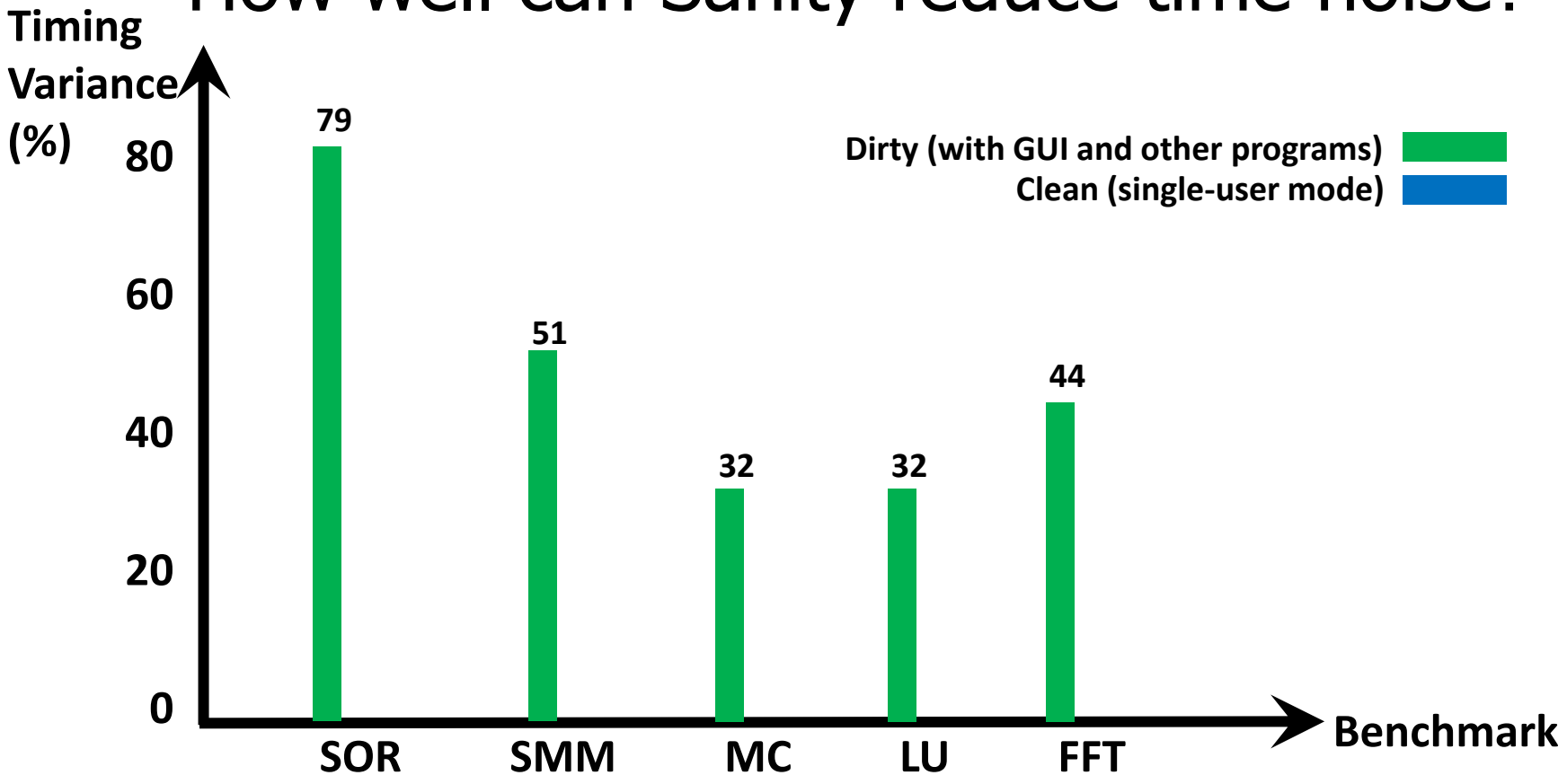
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



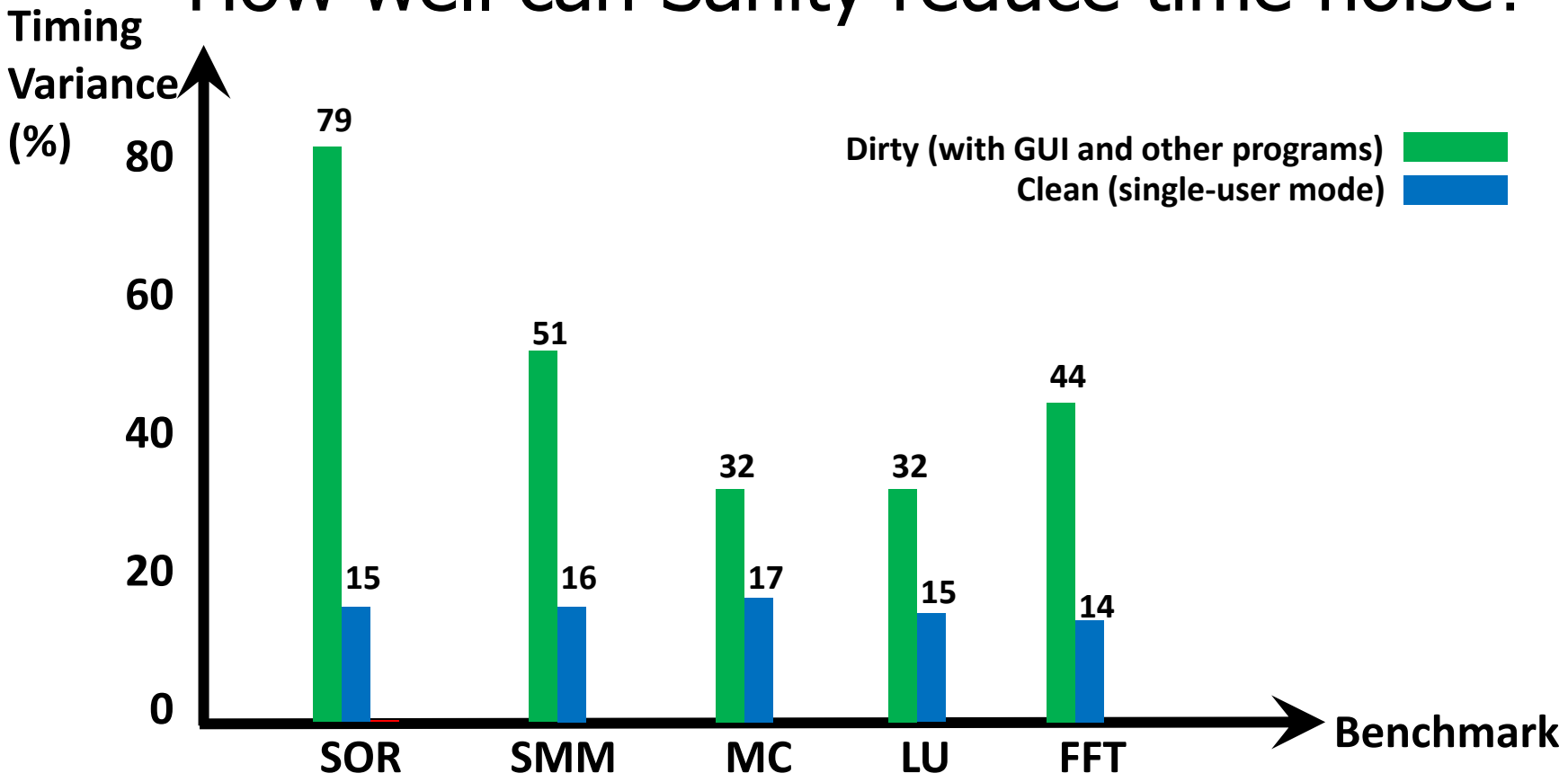
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



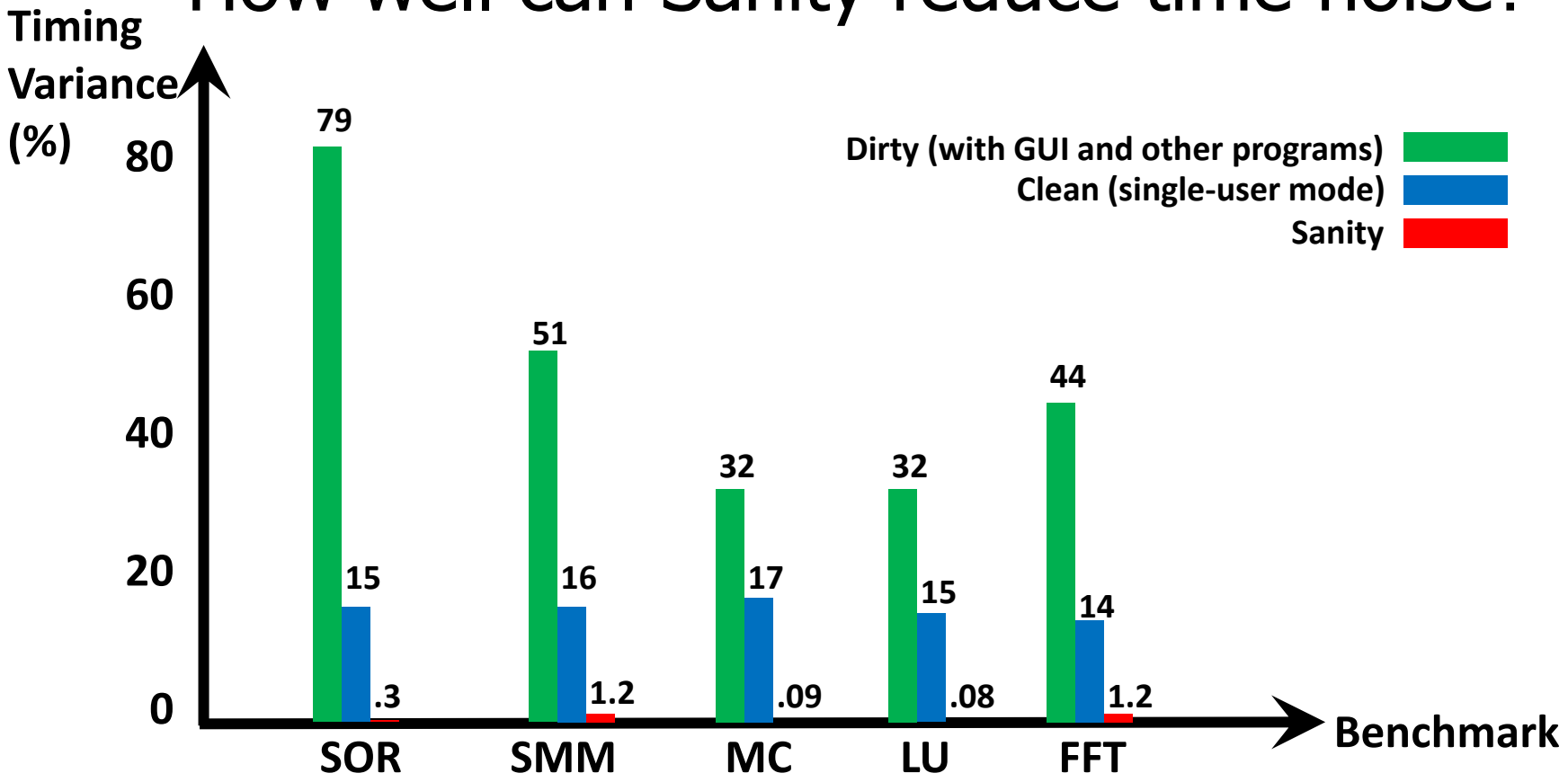
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



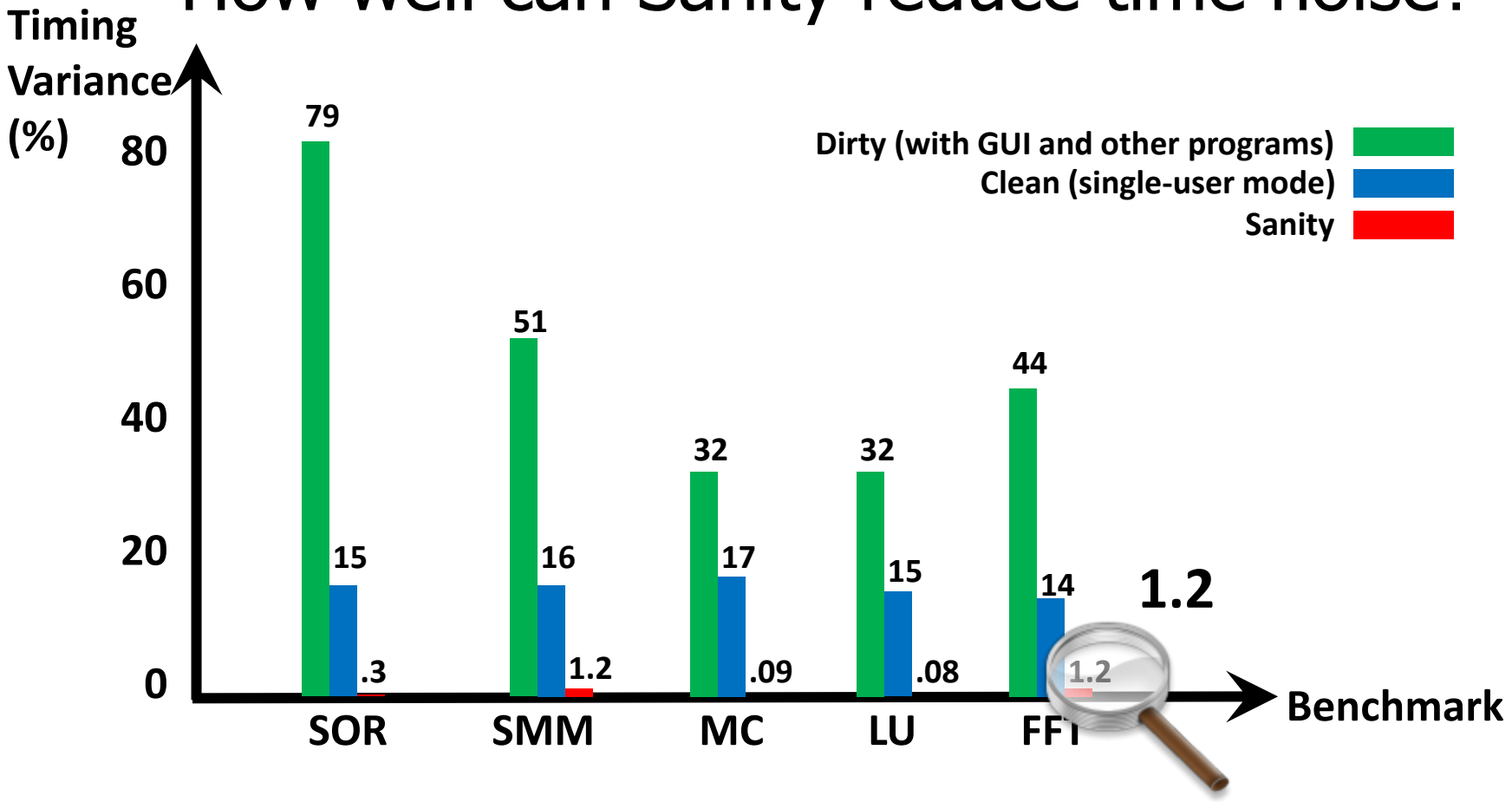
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



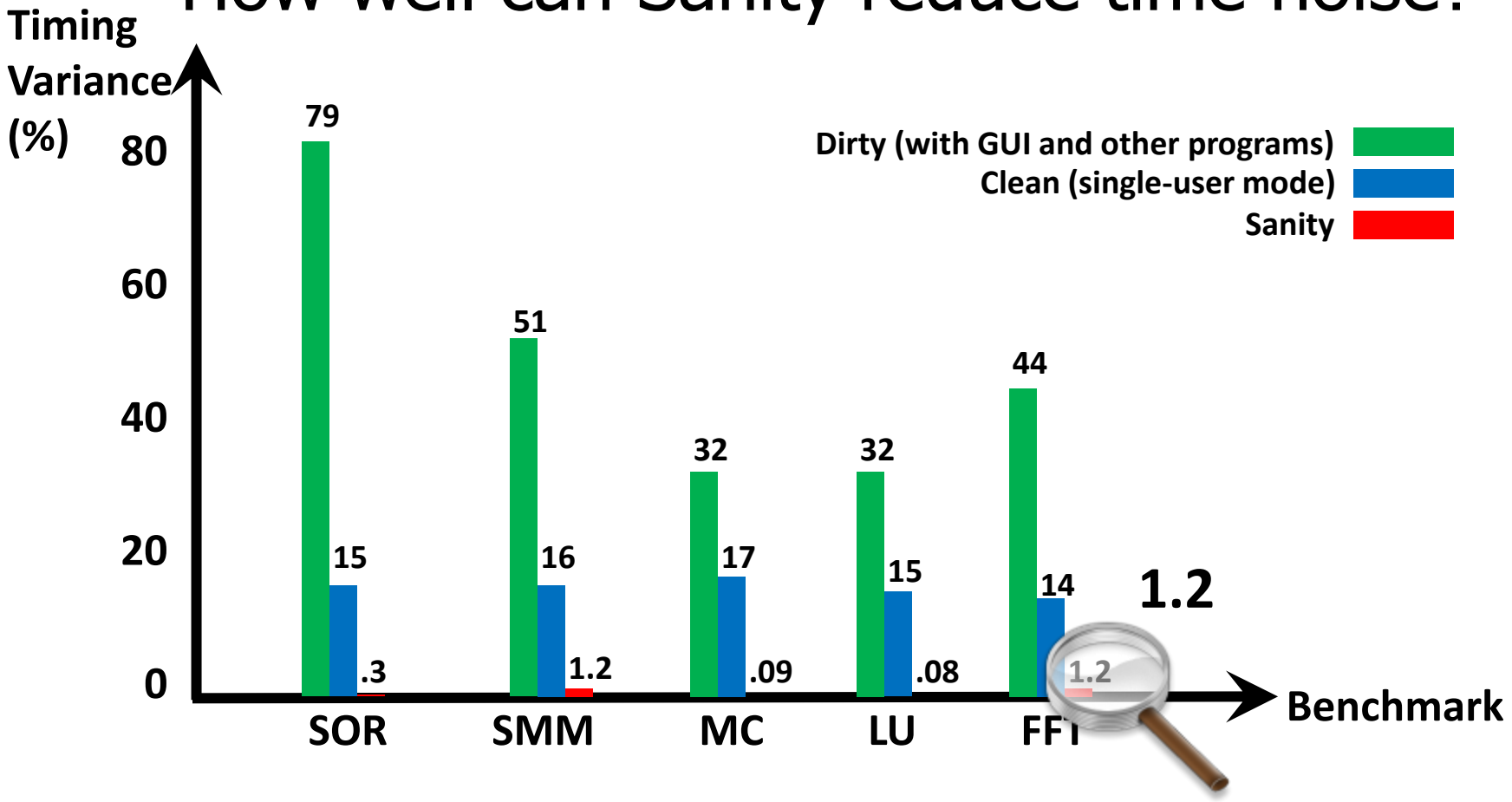
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



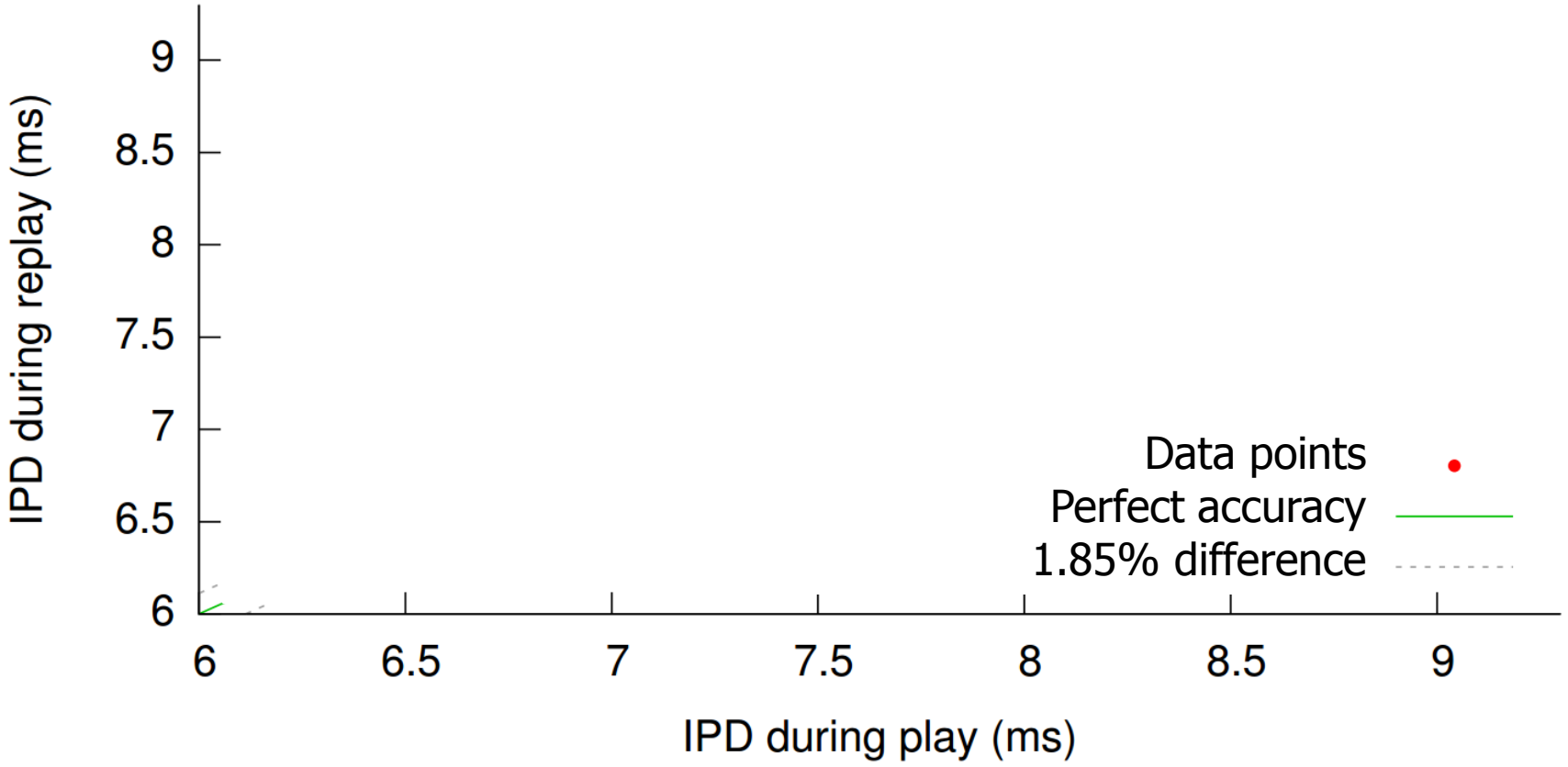
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity

How well can Sanity reduce time noise?



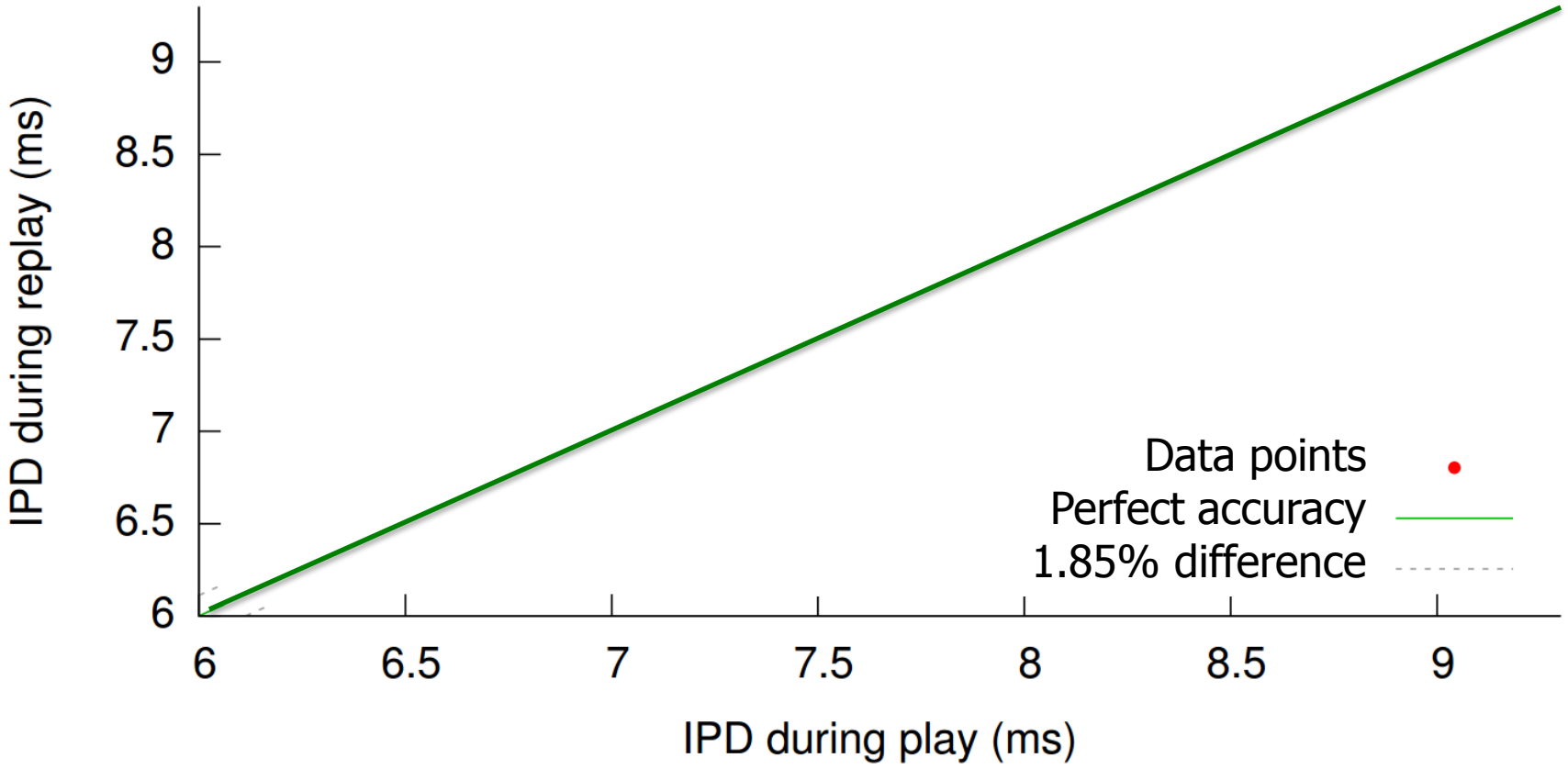
- Experiment: Run SciMark2 for 100 times in Oracle's JVM and Sanity
- Sanity's time-determinism is **orders of magnitude better** than that of a standard JVM!

How well can Sanity reproduce timing?



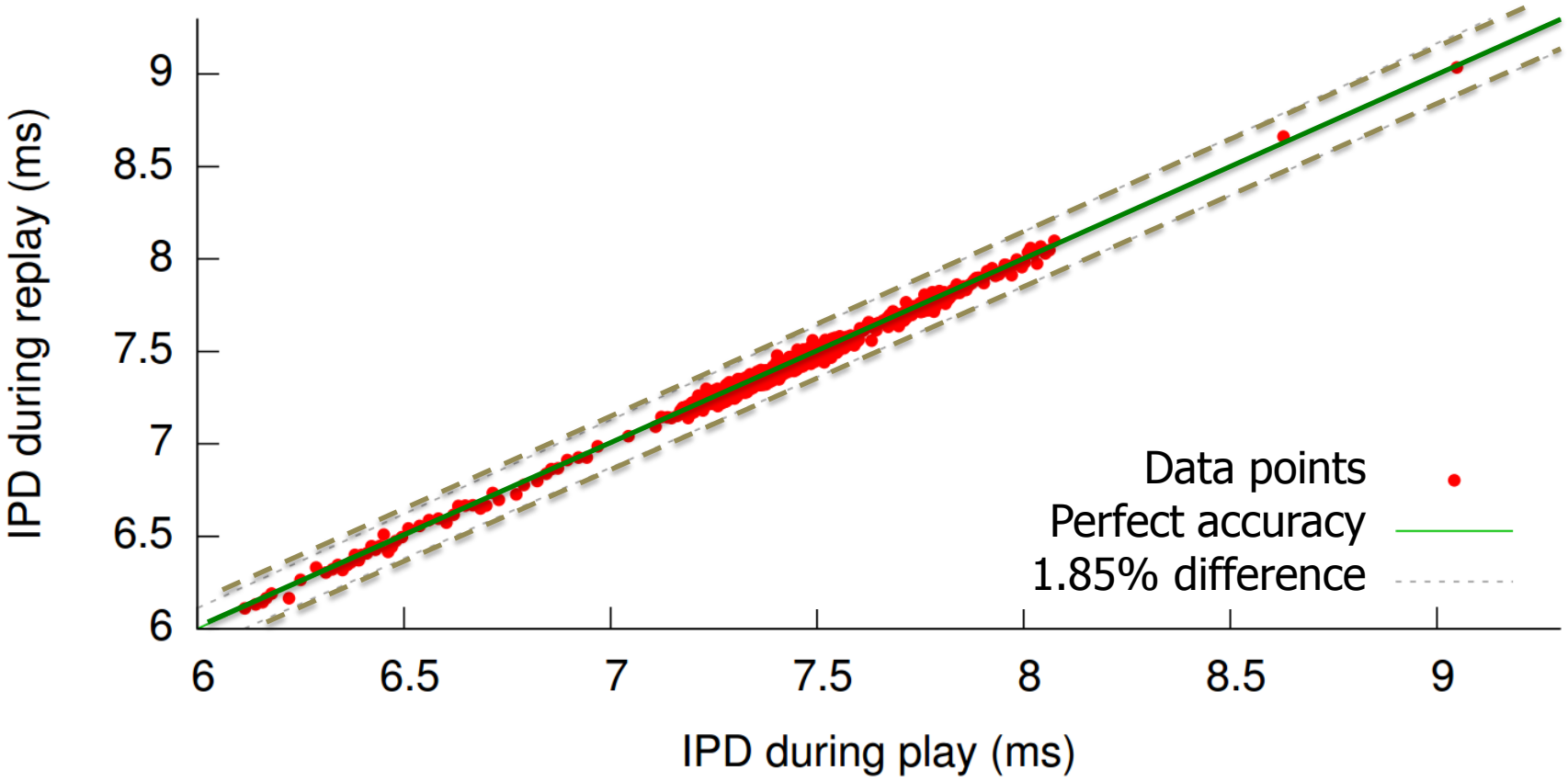
- Experiment: Run nfsj and serve 30 files, then replay.

How well can Sanity reproduce timing?



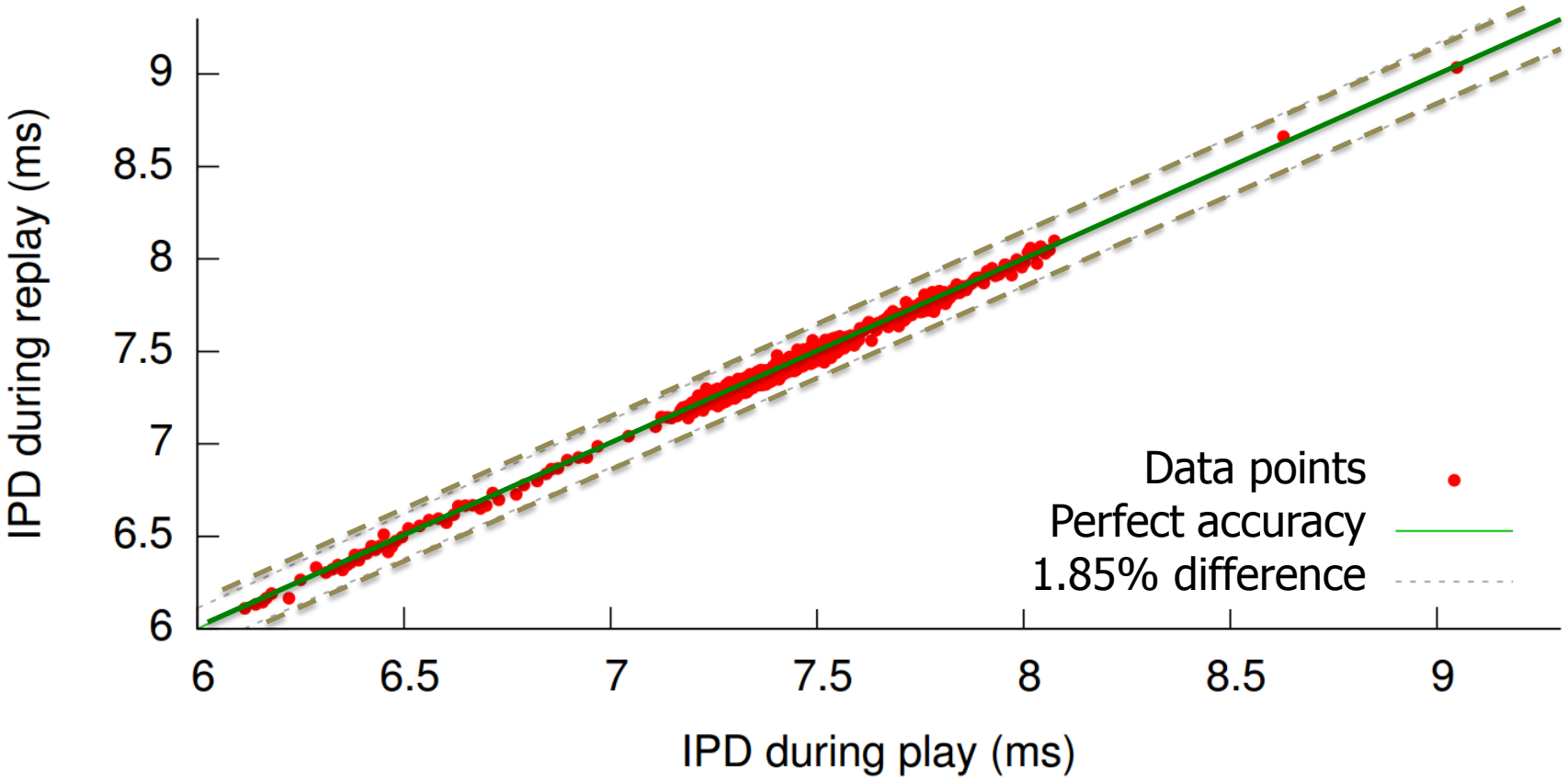
- Experiment: Run nfsj and serve 30 files, then replay.

How well can Sanity reproduce timing?



- Experiment: Run nfsj and serve 30 files, then replay.

How well can Sanity reproduce timing?



- Experiment: Run nfsj and serve 30 files, then replay.
- Sanity can **almost perfectly** reproduce the timing of network outputs during replay

How well can Sanity detect timing channels?

How well can Sanity detect timing channels?

We implemented three known channels:

- IP covert timing channel (IPCTC) [CBS-CCS'04]
- Traffic replay covert timing channel (TRCTC) [Cabuk-thesis'06]
- Model-based covert timing channel (MBCTC) [GWWJ-RAID'08]

How well can Sanity detect timing channels?

We implemented three known channels:

- IP covert timing channel (IPCTC) [CBS-CCS'04]
- Traffic replay covert timing channel (TRCTC) [Cabuk-thesis'06]
- Model-based covert timing channel (MBCTC) [GWWJ-RAID'08]

Plus one new channel:

- "Needle in a haystack" (worst case for detector)



How well can Sanity detect timing channels?

We implemented three known channels:

- IP covert timing channel (IPCTC) [CBS-CCS'04]
- Traffic replay covert timing channel (TRCTC) [Cabuk-thesis'06]
- Model-based covert timing channel (MBCTC) [GWWJ-RAID'08]

Plus one new channel:

- "Needle in a haystack" (worst case for detector)



We used four known detectors:

- Shape test [CBS-CCS'04]
- KS test [PNR-S&P'06]
- Regularity test [CBS-CCS'04]
- Corrected conditional entropy test [GW-CCS'02]

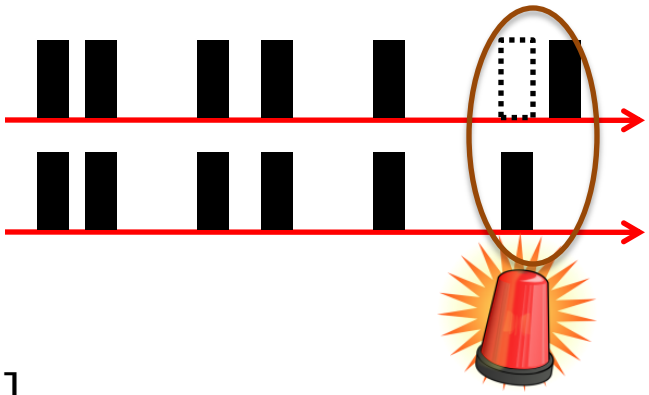
How well can Sanity detect timing channels?

We implemented three known channels:

- IP covert timing channel (IPCTC) [CBS-CCS'04]
- Traffic replay covert timing channel (TRCTC) [Cabuk-thesis'06]
- Model-based covert timing channel (MBCTC) [GWWJ-RAID'08]

Plus one new channel:

- "Needle in a haystack" (worst case for detector)

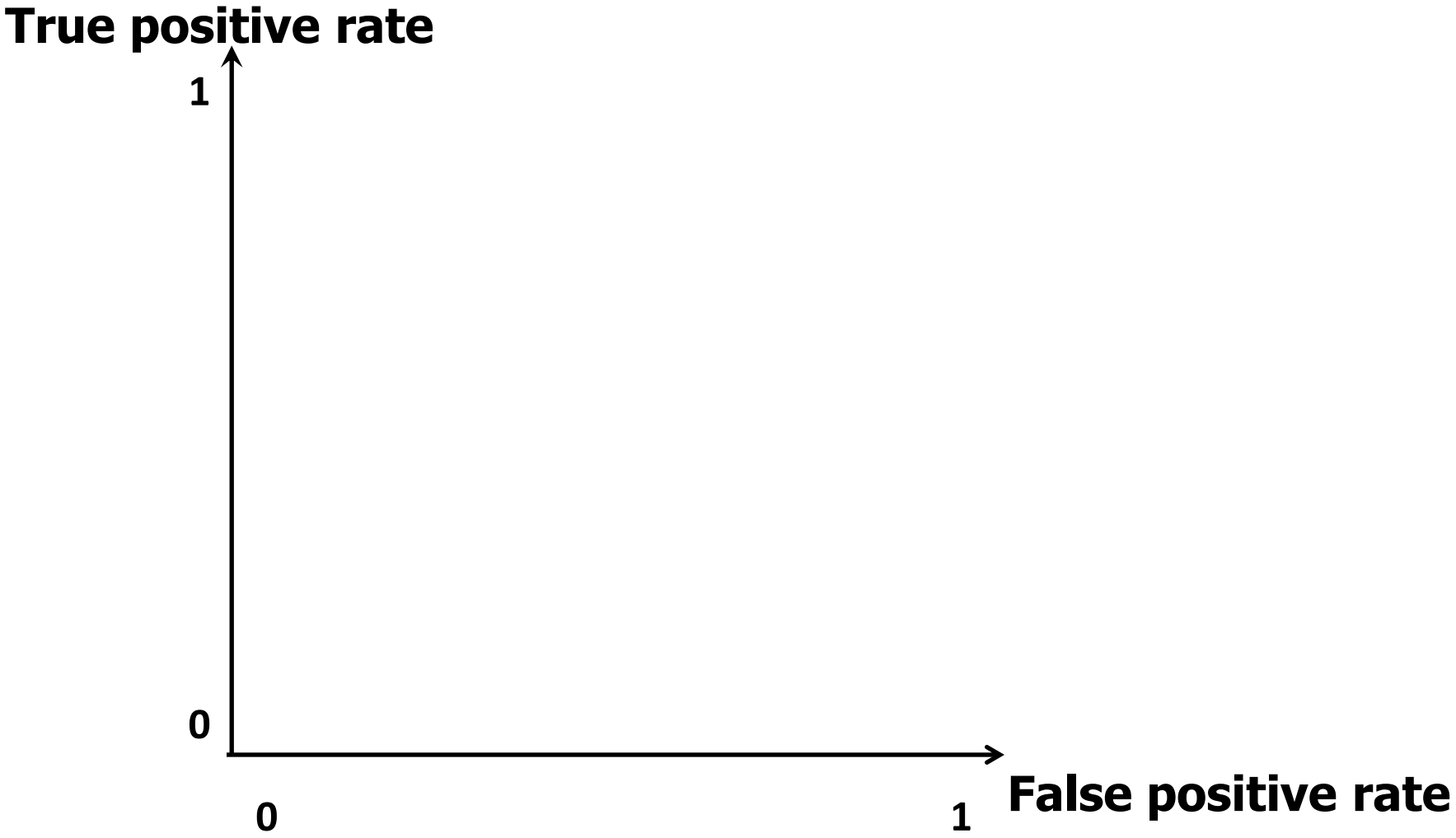


We used four known detectors:

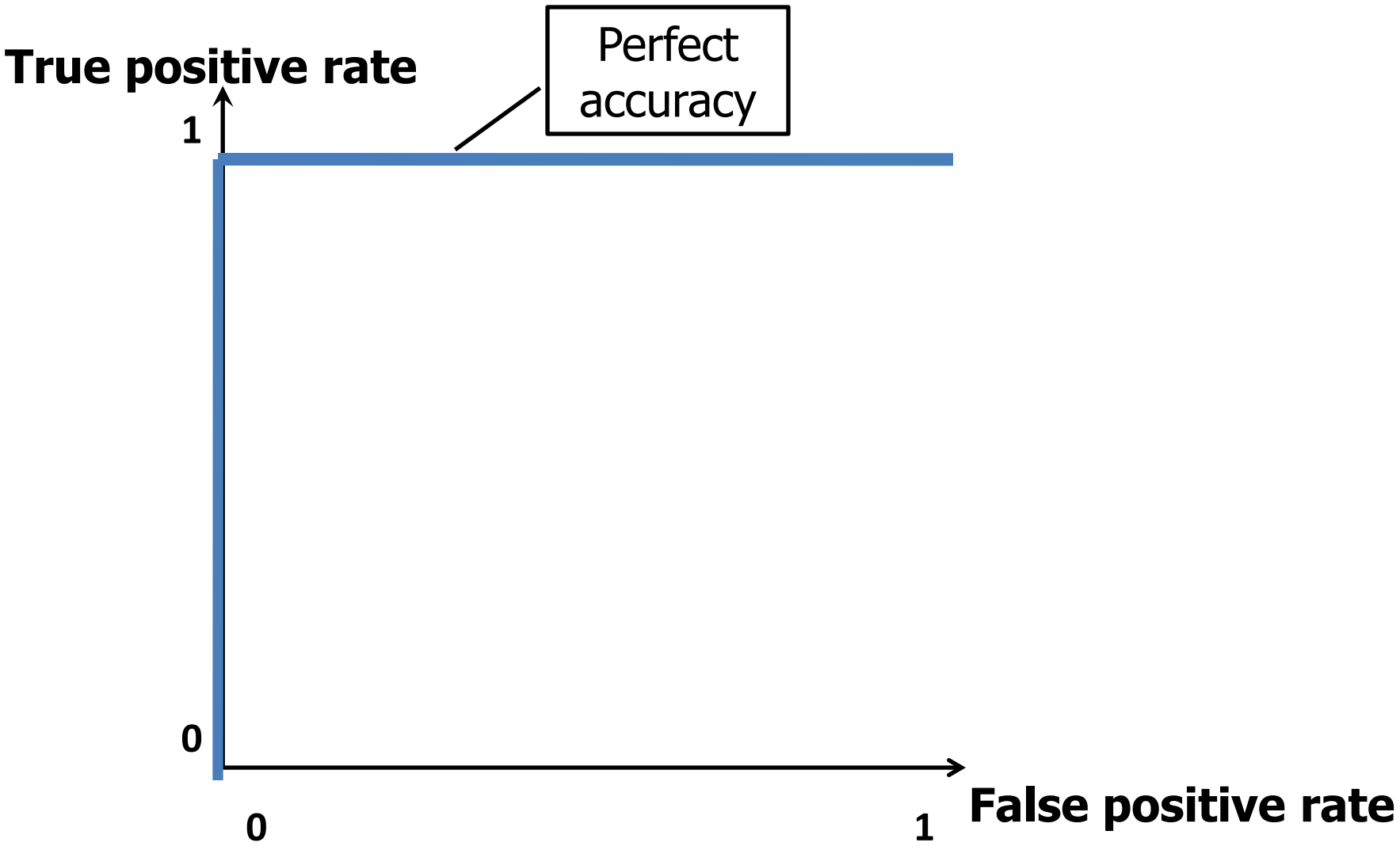
- Shape test [CBS-CCS'04]
- KS test [PNR-S&P'06]
- Regularity test [CBS-CCS'04]
- Corrected conditional entropy test [GW-CCS'02]

Plus our new Sanity-based detector

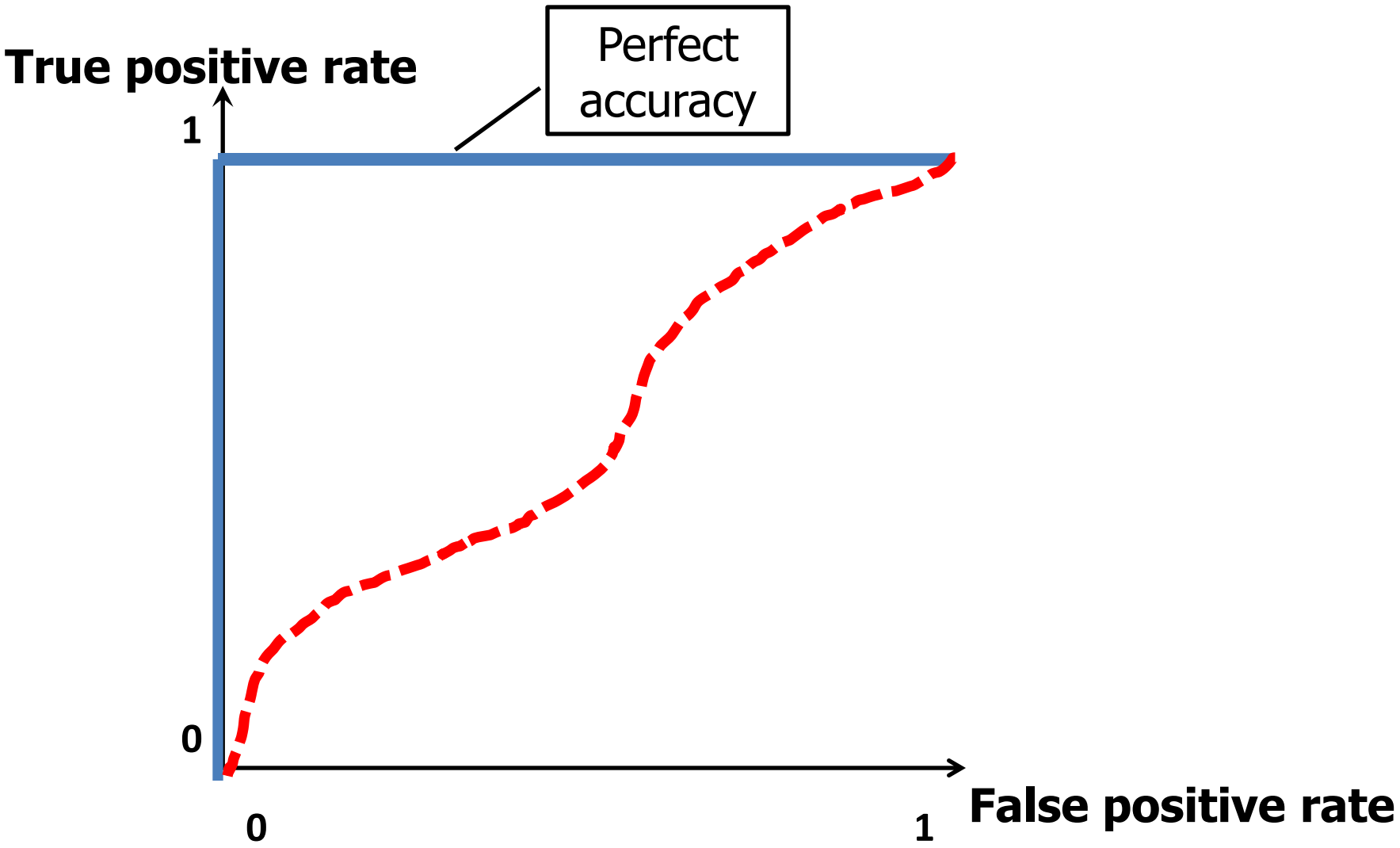
How to measure the quality of a detector



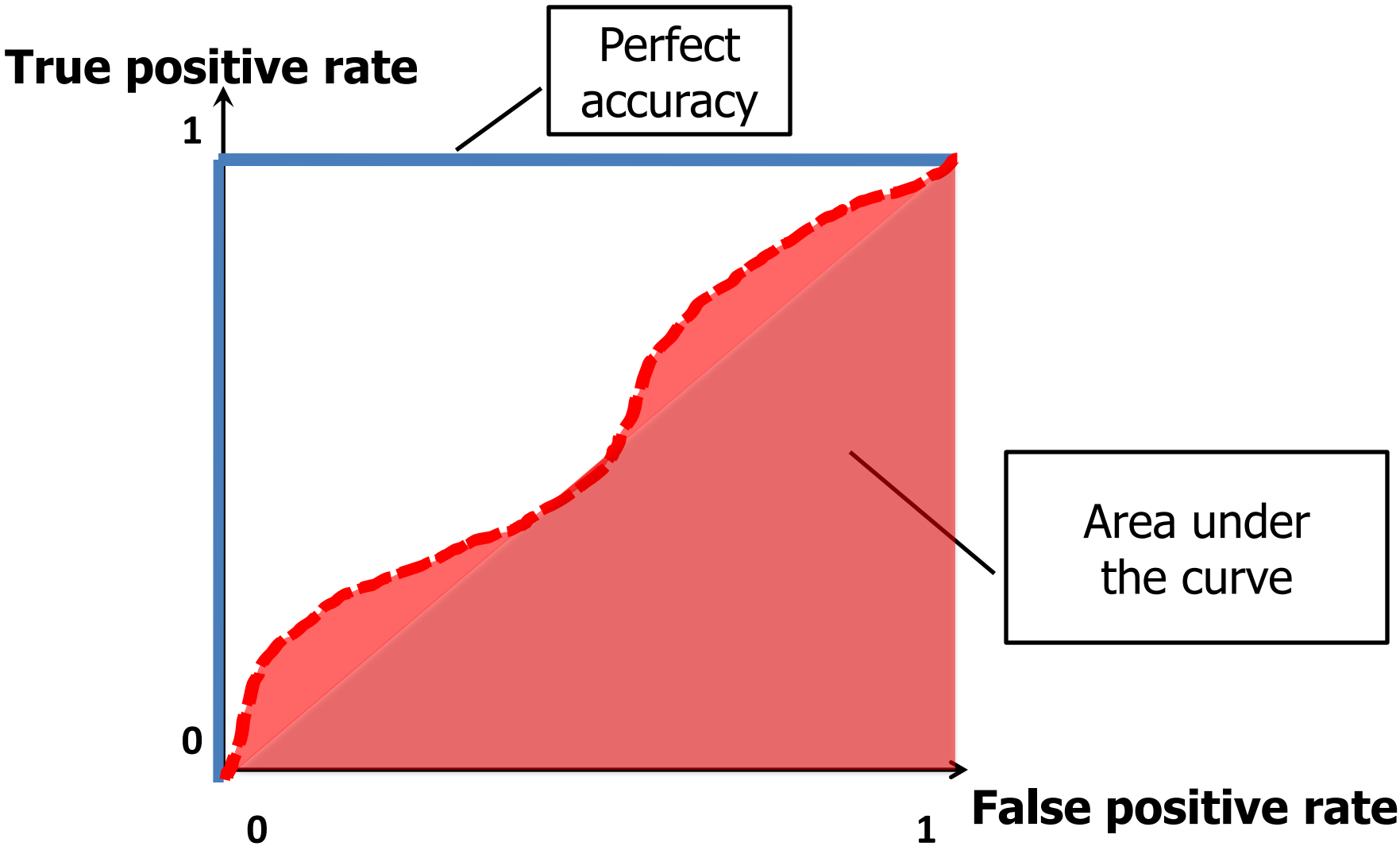
How to measure the quality of a detector



How to measure the quality of a detector



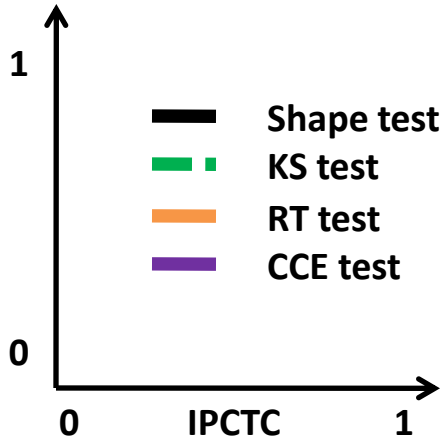
How to measure the quality of a detector



How well can Sanity detect timing channels?

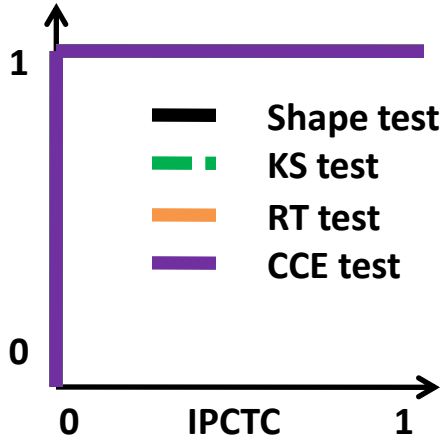
- Experiment: Run each channel against each detector

How well can Sanity detect timing channels?



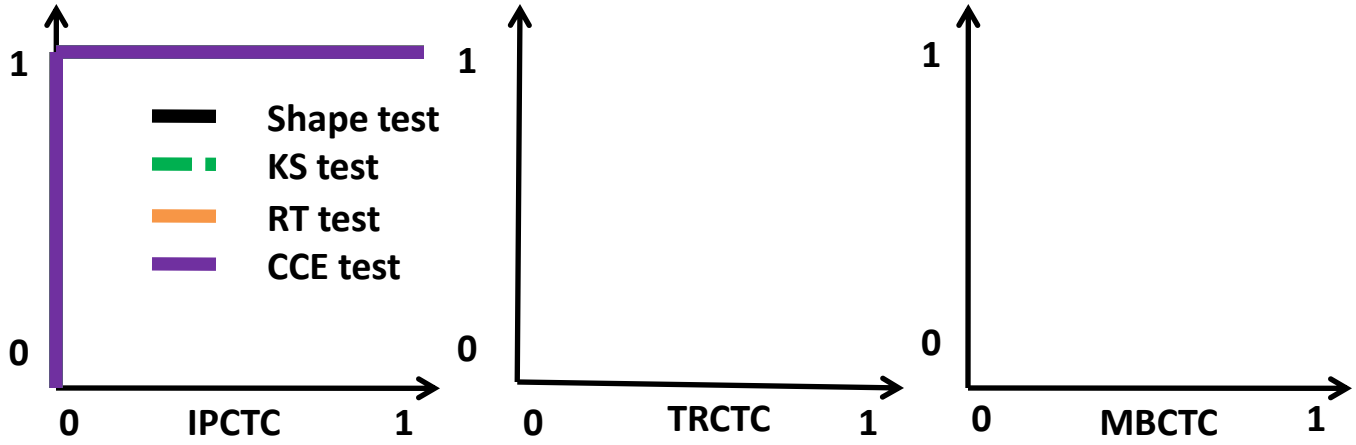
- Experiment: Run each channel against each detector

How well can Sanity detect timing channels?



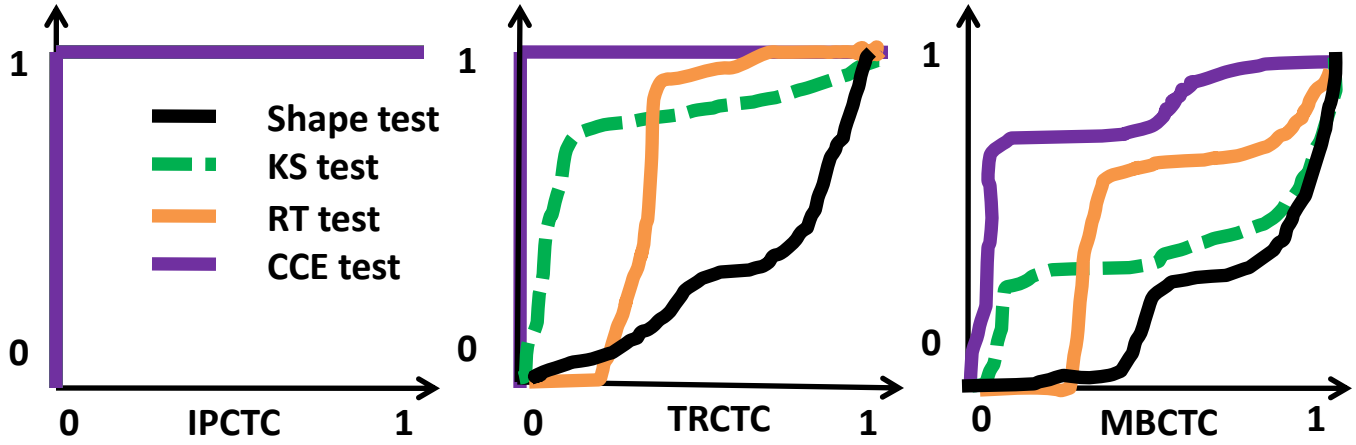
- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy

How well can Sanity detect timing channels?



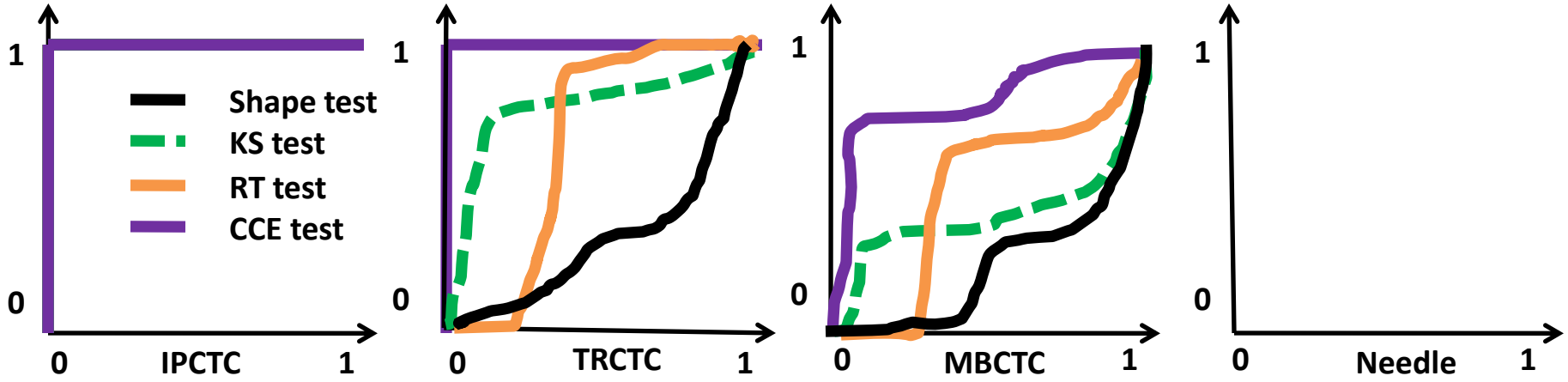
- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy

How well can Sanity detect timing channels?



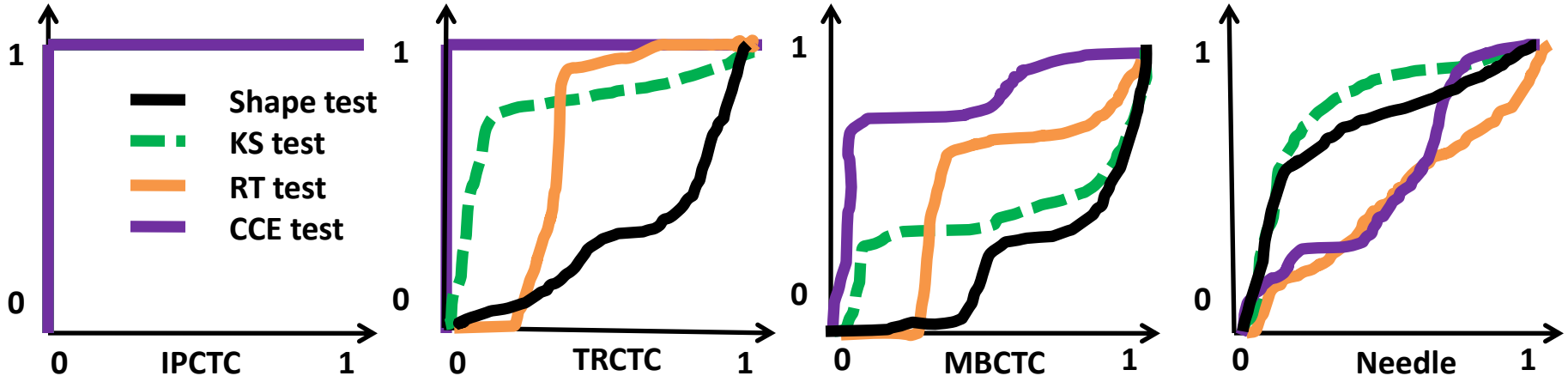
- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy
  - Existing detectors do worse for more sophisticated channels

How well can Sanity detect timing channels?



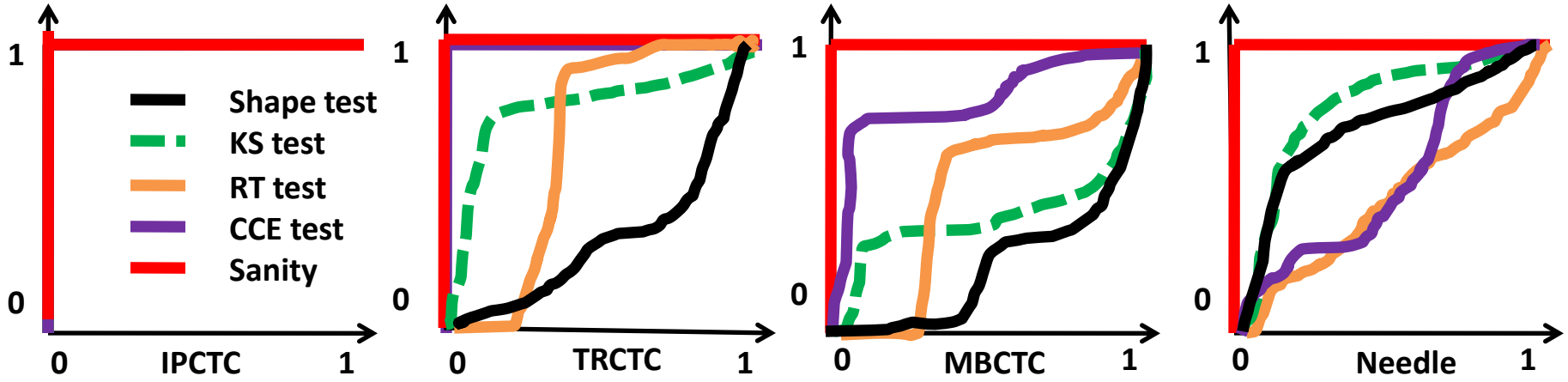
- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy
  - Existing detectors do worse for more sophisticated channels

How well can Sanity detect timing channels?



- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy
  - Existing detectors do worse for more sophisticated channels
  - Existing detectors cannot detect "Needle in a haystack" well

How well can Sanity detect timing channels?



- Experiment: Run each channel against each detector
- Observations:
  - All detectors can detect IPCTC with perfect accuracy
  - Existing detectors do worse for more sophisticated channels
  - Existing detectors cannot detect "Needle in a haystack" well
  - **Sanity detects <u>all</u> channels with perfect accuracy! (no false positives, no false negatives)**

Summary

- Goal: Detect covert timing channels
- Existing detectors look for signs of specific, known channels
  - Result: "Cat-and-mouse game" with the attacker
- Our approach: Compare the observed timing to what it 'should be' if the machine is not compromised
  - Works for all timing channels, including novel ones
  - Key challenge: How do we know what the timing should be?
- We introduce time-deterministic replay (TDR)
- We have built a TDR prototype called Sanity
  - Reproduces timing to within 2% (on commodity hardware)
  - Can be used to detect a variety of existing and novel covert timing channels with perfect accuracy