VICI

Virtual machine Introspection for Cognitive Immunity

Tim Fraser

Komoku Inc.

tfraser@komoku.com

SRS2 kickoff meeting 11 December 2006

VICI = VMI + repair + learning

Problem:

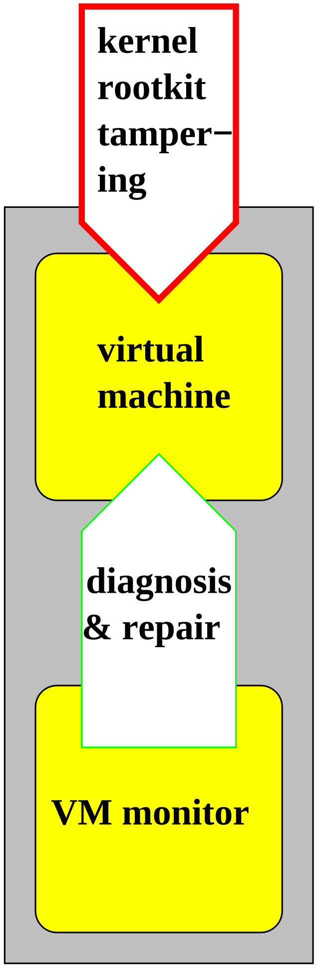
- Kernel rootkits are a growing threat.
- Static defenses insufficient.

Proposed solution:

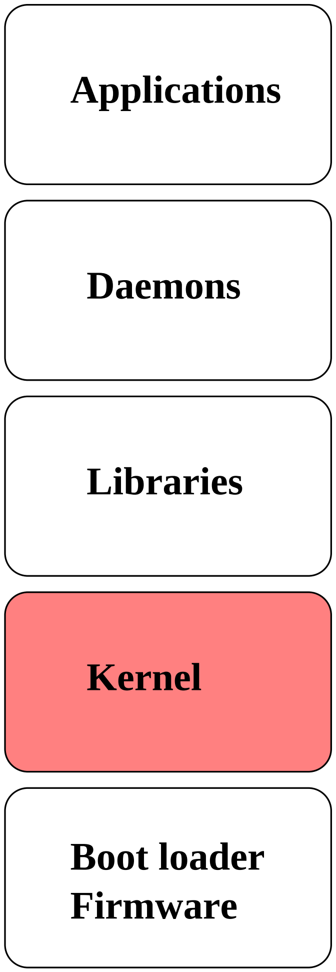
- VM Introspection for self-diagnosis
- Automated repair for self-healing.
- Learning for “cognitive immunity.”

Goal:

Enable COTS systems to endure, recover from repeated attacks.



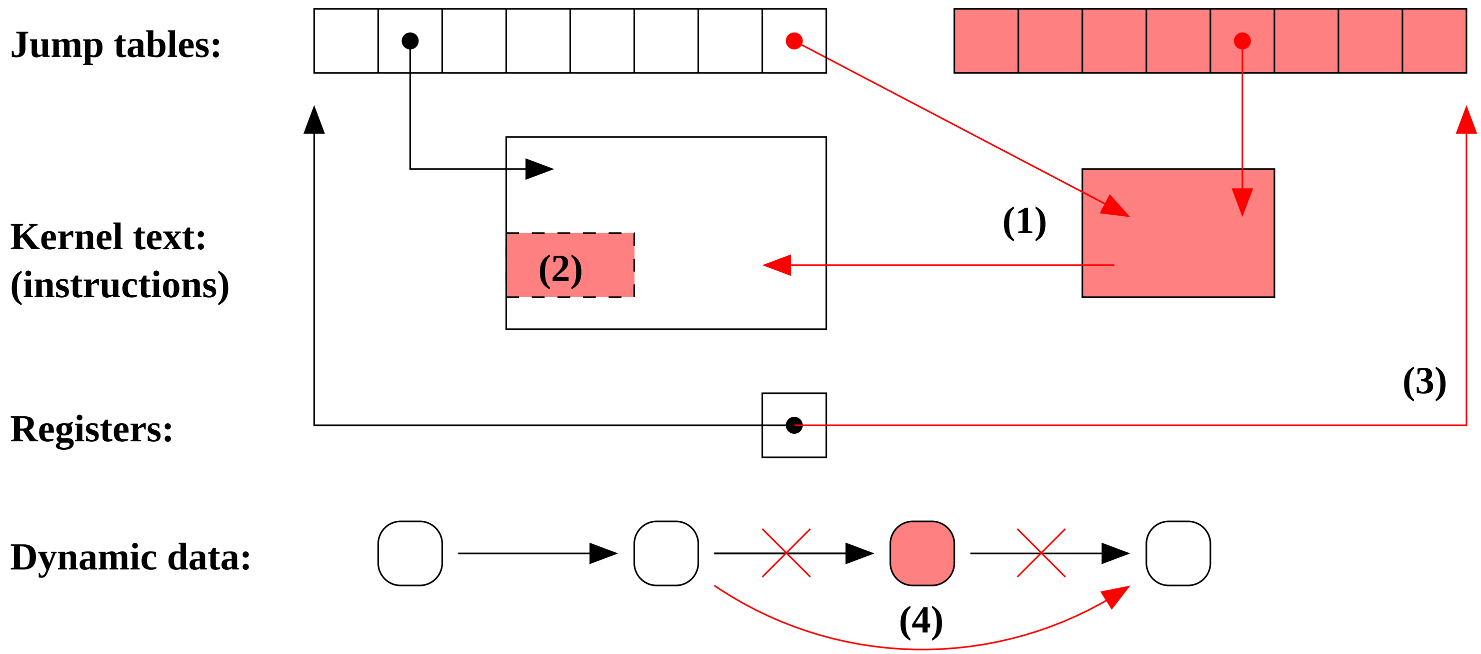
Kernel rootkit threat



1. Adversary gains administrative control.
2. Adversary installs rootkit to hide from administrators.
3. Adversary leaves backdoors, keyloggers bedhind.
4. Rootkit makes kernel *lie* to administrators: “All is well.”

Rootkits hard to detect from higher levels.

Kernel rootkit techniques



Ways a rootkit can modify a kernel to hide or promote an adversary over the long term:

1. redirect services
2. tamper with code
3. redirect registers
4. tamper with data

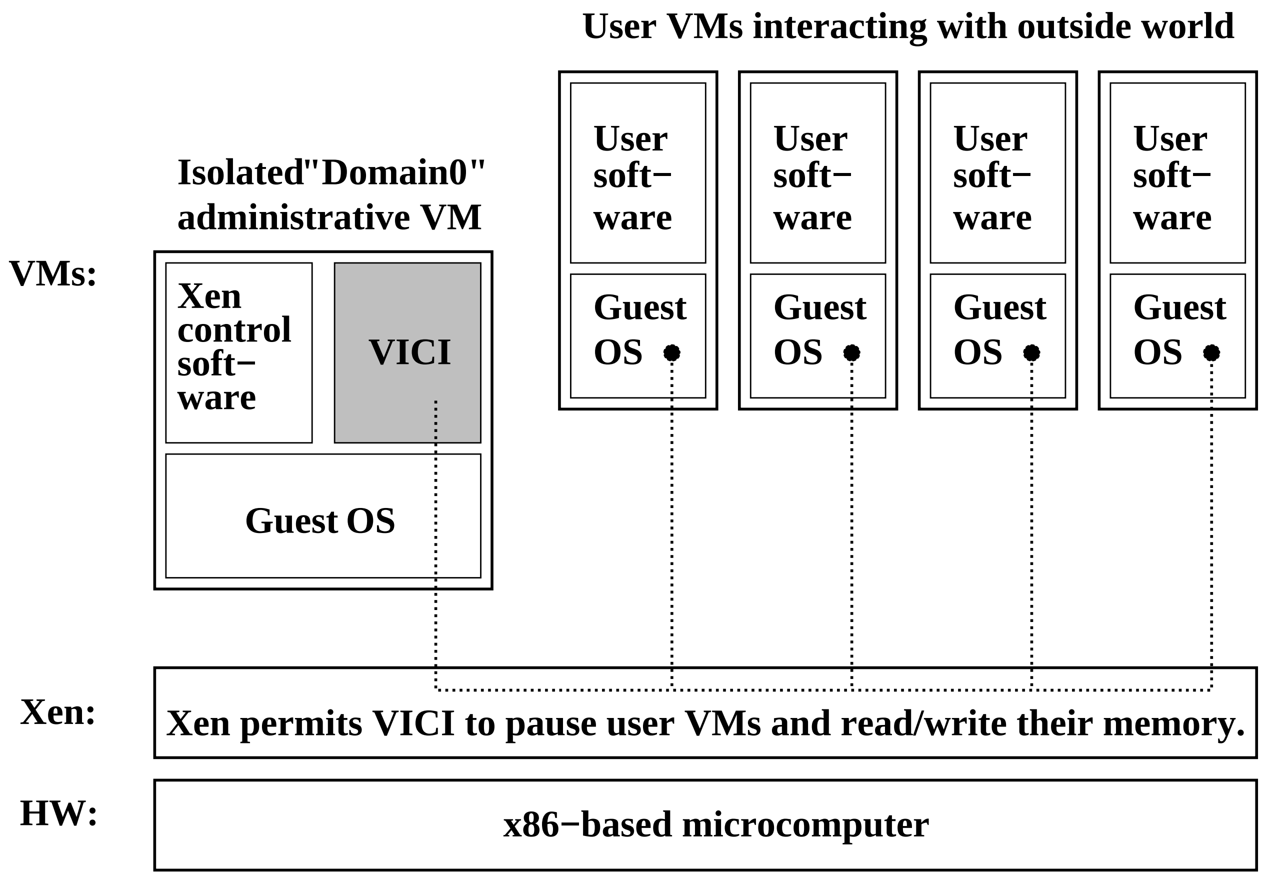
Defensive state of the art

- Detection tools are just being productized.
- Repair (rootkit removal) is a research problem.
- Persistent adversaries will overwhelm static defenses without the capability to learn and adapt ("Cognitive Immunity").

VICI project =

kernel rootkit detection + repair + learning

VICI prototype architecture



- VICI monitors, repairs guest OS kernels.
- All machines run commodity GNU/Linux OSs.

Technical approach

Adapt techniques demonstrated in similar environments:

Detection:

- VM Introspection [Garfinkel 2003]
- Semantic Integrity constraints [Petroni 2006]
- Komoku Monitoring Engine

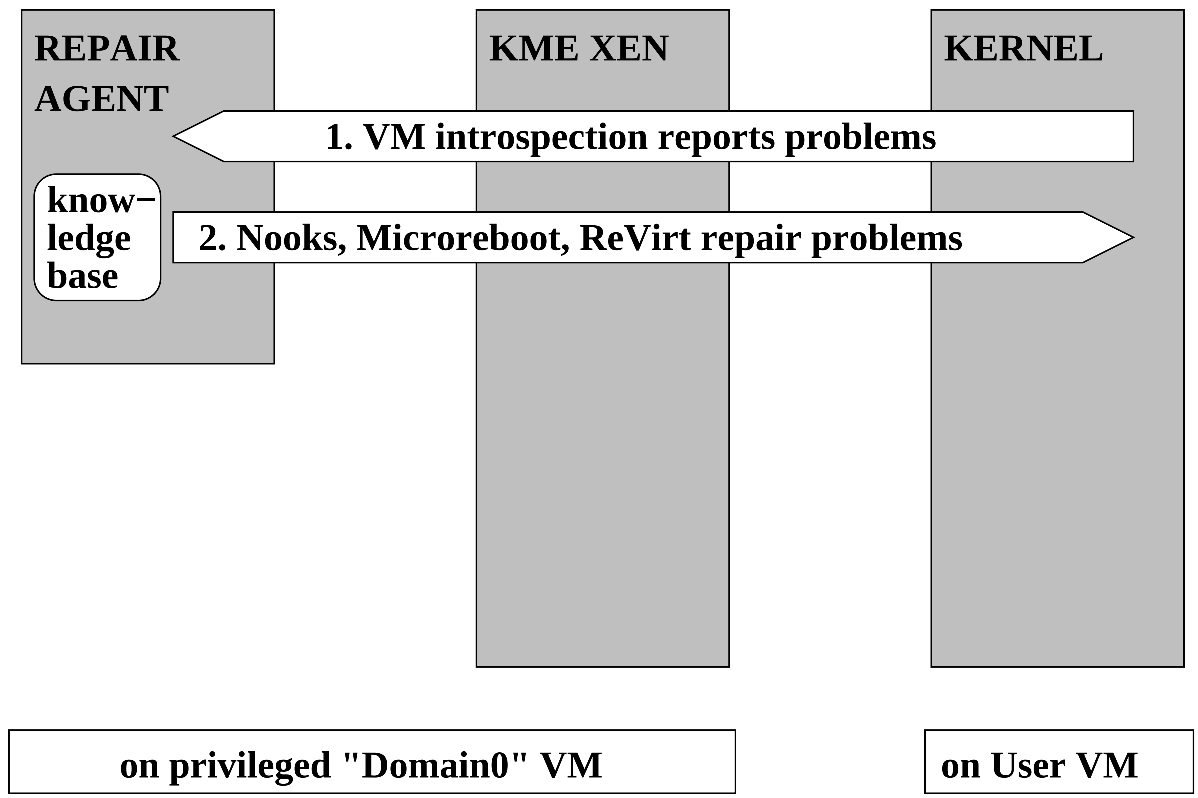
Repair:

- Microreboot [Candea 2004]
- Nooks [Swift 2003]
- ReVirt [Dunlap 2002]

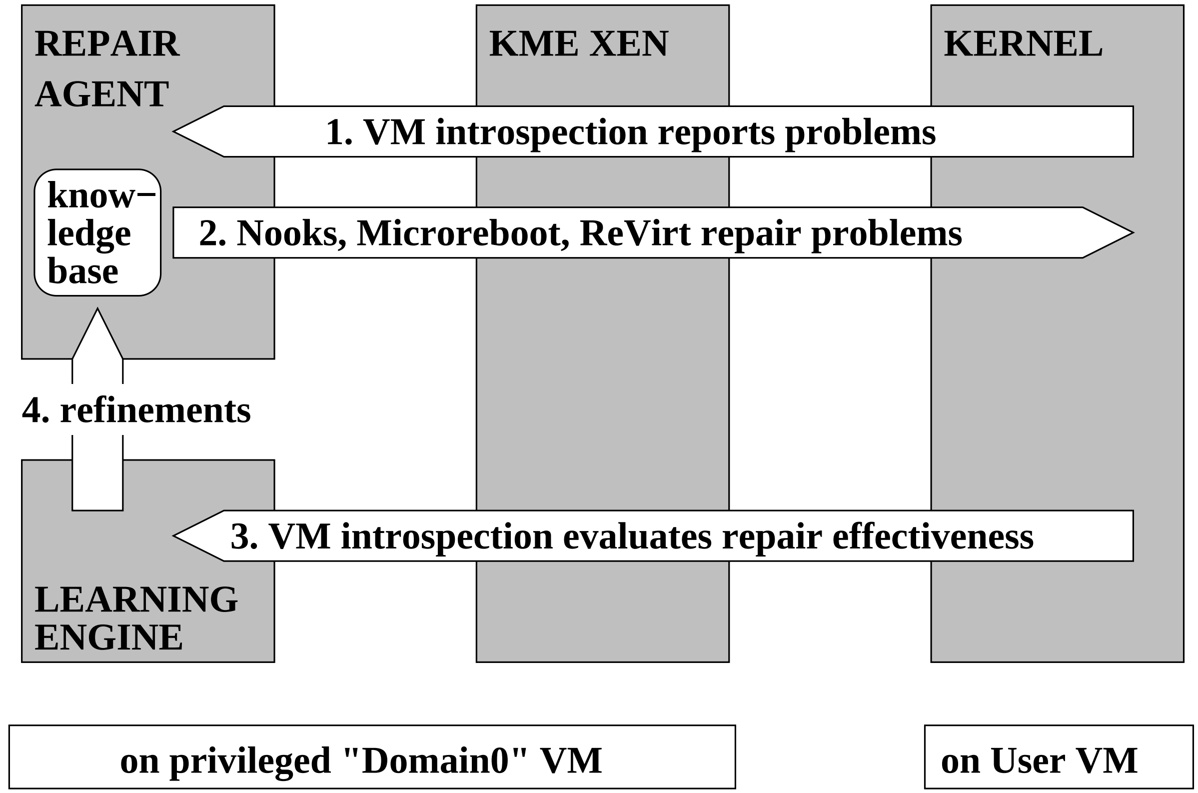
Cognitive Immunity:

- Circuit/behavior-based agent [Brooks 1991]
- learning for behavior-based agents [Maes 1990]

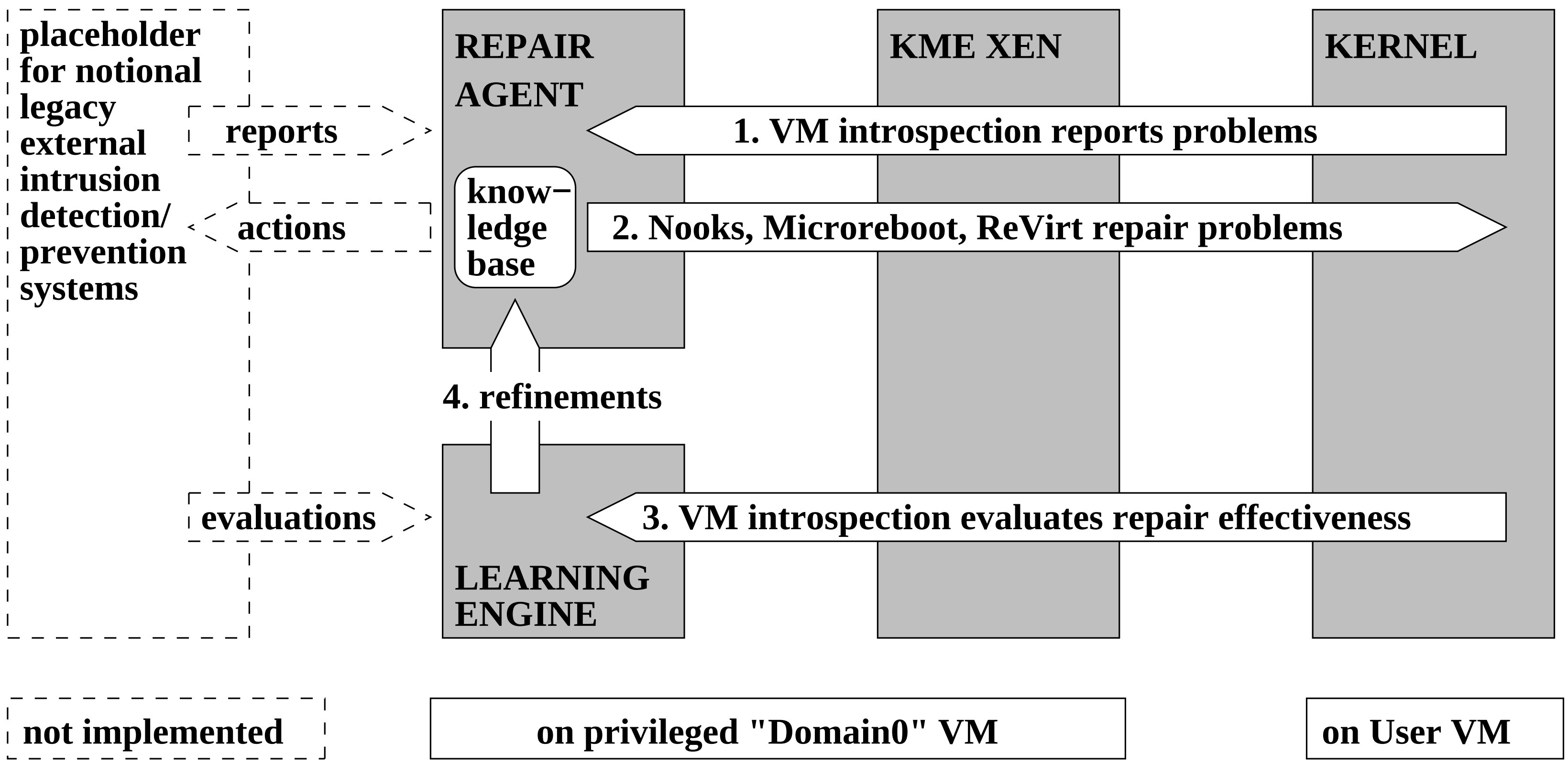
VICI theory of operation (1:3)



VICI theory of operation (2:3)



VICI theory of operation (3:3)



Remainder of talk

Technical approaches to:

1. Detection
2. Repair
3. Learning

Detection

Purpose: Support self-diagnosis:

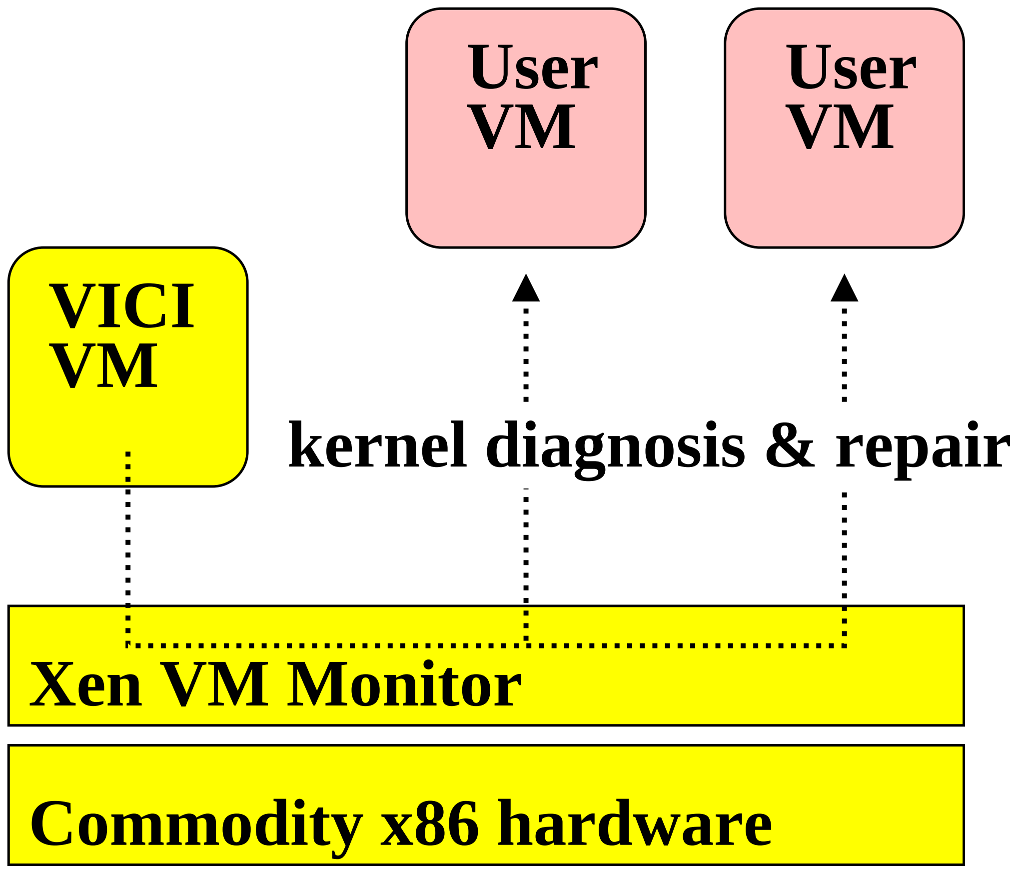
- Detect kernel tampering.
- Direct choice of repair actions.

Goal: Meet SRS2 detection requirement: 50% of attacks.

Measurement: Trials against sample kernel rootkits or similar manual tampering.

KME/Xen exists but will need enhancement.

Virtual Machine Introspection



+ Good monitor isolation.
+ Monitor can see machine registers, can pause VMs.

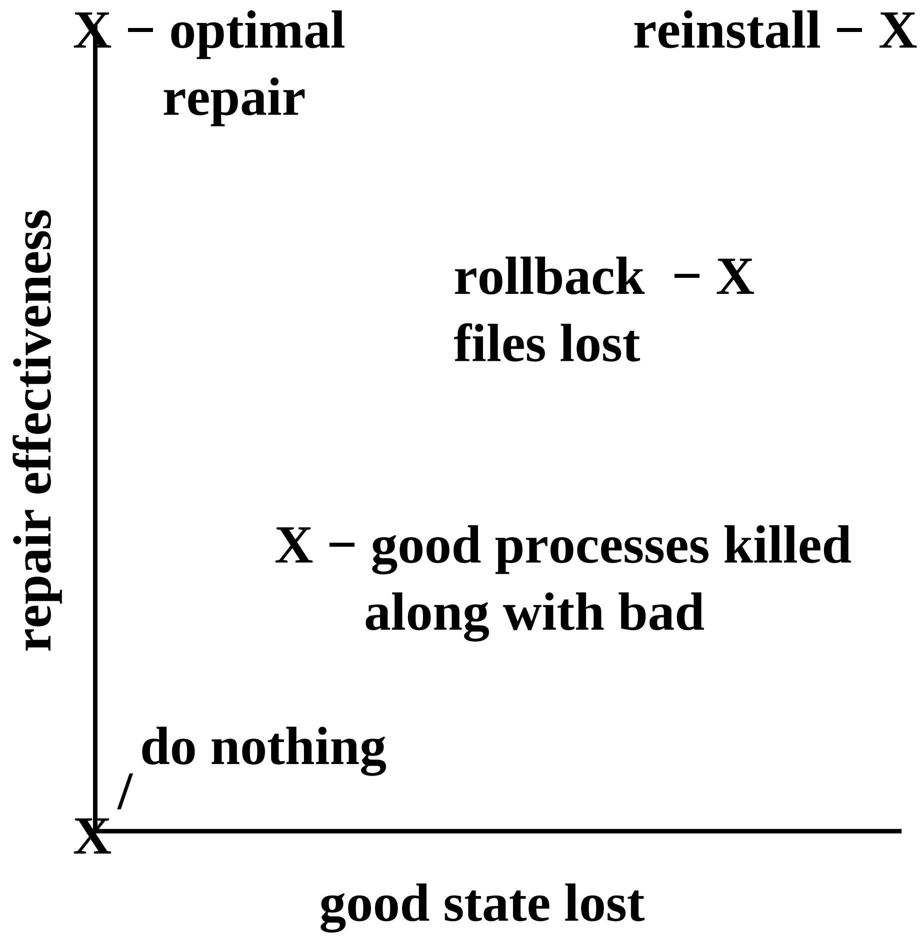
Addressing detection problem risks

- Copilot, follow-on Komoku products already do detection.
- Borrowing Komoku KME/Xen technology.
- Detection predicates based on manual analysis.
- Experience with Semantic Integrity [Petroni 2006].
- Detection predicates coded manually.
- Virtual machines can be paused.
- Entire VM state is visible.

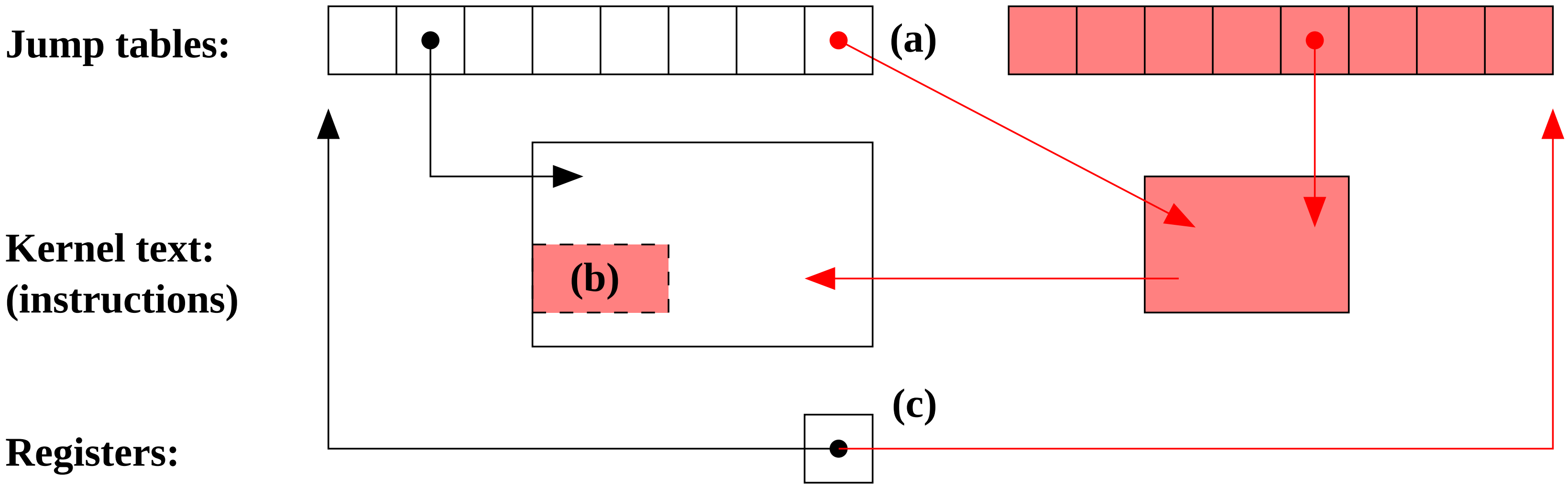
Repair

Goal:

- Implement a collection of “repair actions”.
- Remove, address tampering.
- Save as much good state as possible. (“surgical repair”)



Simple tampering, simple repair



Method:

1. Pause VM.
2. Restore original values or patch rootkit code.
3. Unpause VM.

Good for:

(a) pointers

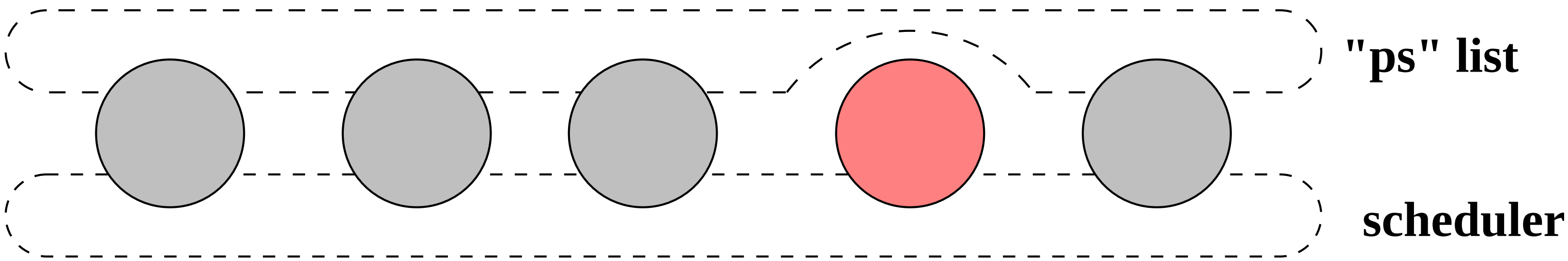
(b) text

(c) registers

What about data?

Tampering with dynamic data

DKOM process hiding example:



- You can imagine tweaking links above, but...
- What about more complex cases?
  - Can't tell what is bad?
  - Can't tell what good should be?

Approximate repair

Issues when repairing kernel data structures:

Isolation:

- Kernel may inadvertently spread corruption.
- initial cause of crashes may be distant in time,
- originating in distant components.

Synchronization:

- If a repair makes a component forget an ongoing transaction
- VICI must help it resynchronize.

Microreboot

Basic idea:

- Reboot part of an application [Candea 2004].
- Lose less good state than a complete reboot.
- Reboot more parts until all seems well.

Original Context:

"Crash-only software" [Candea 2003]

- many small isolated Java components
- state stored in external DBMS
- retryable requests
- sharing with expirable leases

Microreboot in Linux?

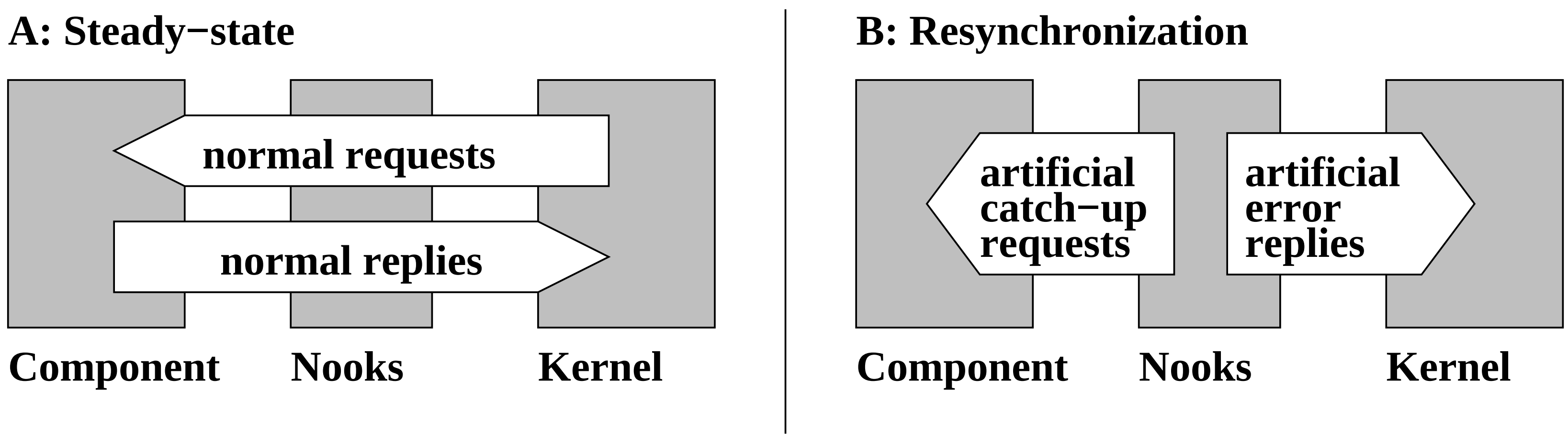
Apply to convenient subsystems, drivers.

Isolation:

- Good rootkits avoid crashes.
- In case of crash or secondary corruption,
- try ReVirt [Dunlap 2002] (progressive VM checkpoint restoration).

Synchronization: try a Nooks-like intermediary.

Nooks



- Nooks protects kernel from device driver crashes [Swift 2003].
- We're not using Nooks isolation, just its synchronization approach.
- Apply to COTS kernels using interposition.

Addressing repair problem risks

- Rootkits aim to keep kernel stable, avoid fatal corruption.
- Simple repairs should be common, low risk.
- We have Linux source.
- Manual analysis and repair action coding sufficient.
- VMs can be paused; entire state visible.
- For more complex cases, we're adapting techniques that have already been demonstrated.

Cognitive Immunity

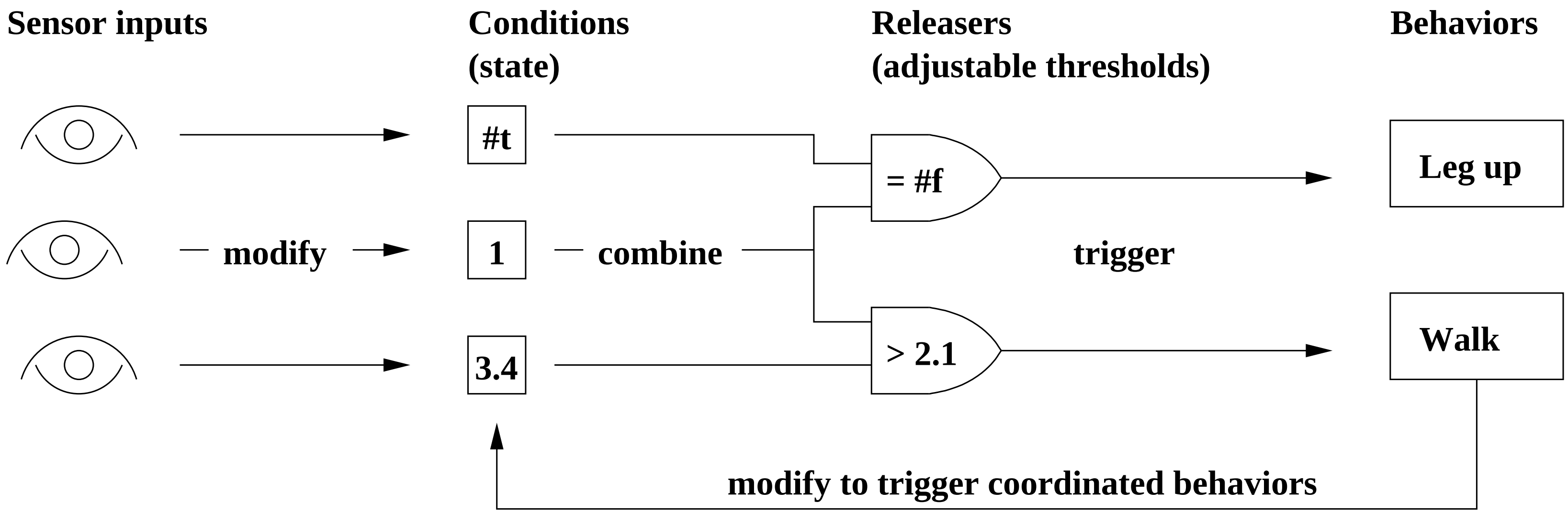
Purpose: Learn best response to repeated attacks.

Goal:

- Learn the response that sacrifices the least good state.
- Once learned, choose that response first.

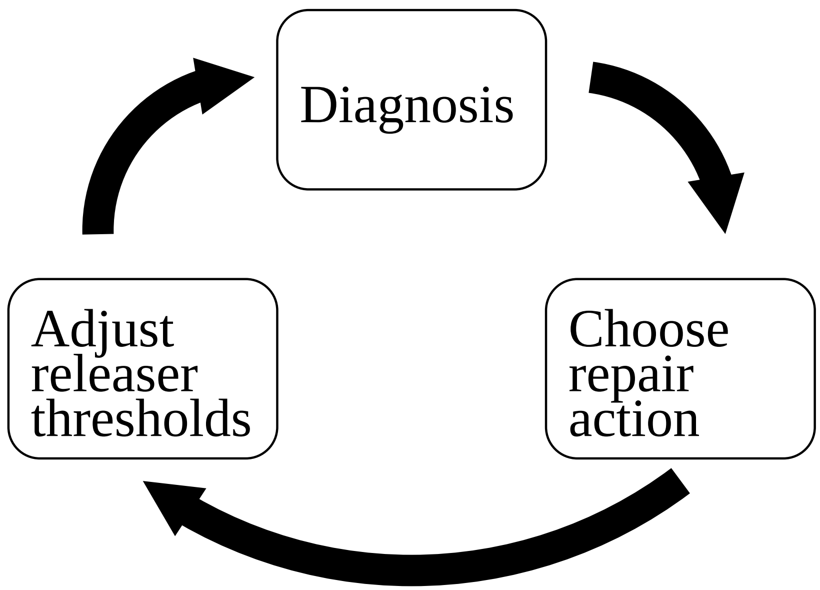
Measurement: Observe whether or not VICI converges upon a best response after various trials.

Circuit/behavior-based reasoning



- Control for bug-like walking robots [Brooks 2001].
- "A reflex agent with state" [Russel 2003].
- Biological analogy: conditions = hormone levels.
- Learning will adjust releaser thresholds.

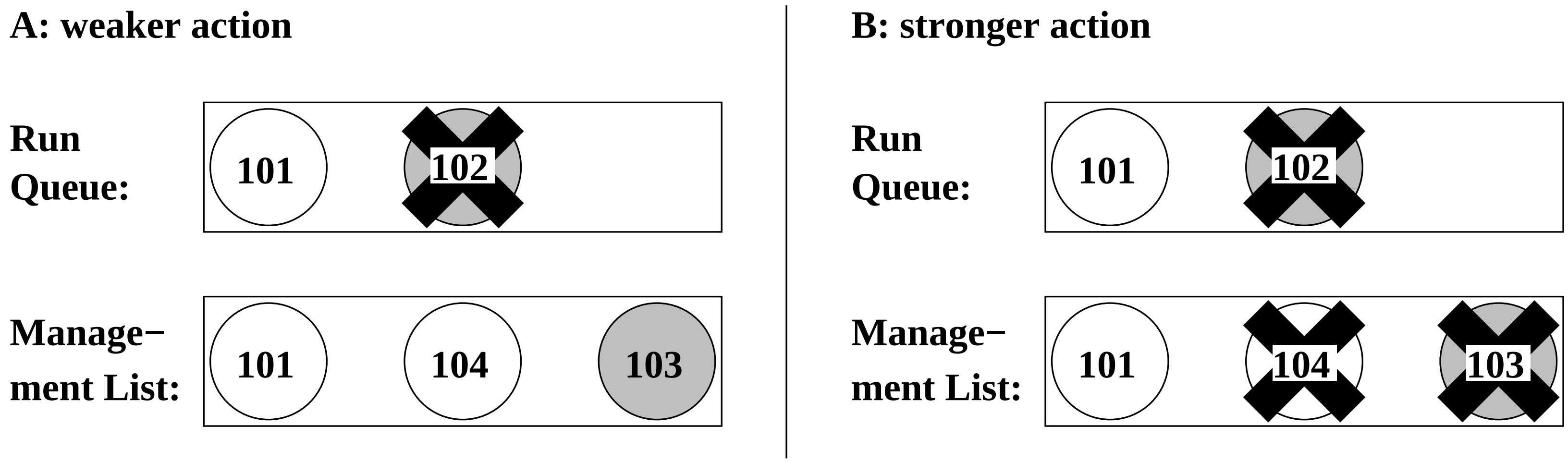
Circuit/behavior-based learning



Goal: Learn to choose the least costly effective repair.

problem		set of relevant repair actions
The foo data	$\Longrightarrow$	tweak existing foo
structure has	·	replace entire foo
been tampered	·	restart foo-using subsystem
with!	·	restore system checkpoint (!!)

Example learning scenario



1. Hiding rootkit process detected (102).
2. Weaker repair kills hider but misses non-hiding helper (103).
3. Stronger repair gets both, but also kills good process (104).

Addressing reasoning and learning risks

- Relatively modest reasoning and learning requirements:
  - No complex planning, only thresholds learned.
- Condition-releaser-behavior network coded manually.
- Detection predicates, repair actions coded manually.
- VICI learns only to choose best response from a finite set of possibilities, all applicable.

Q: Can an adversary trick VICI into dropping its guard?

A: Worst case, VICI can learn to choose only more effective alternatives.

Summary

VICI =

- VM Introspection for self-diagnosis +
- automatic repair for self-healing +
- learning for Cognitive Immunity

Approach: adapt existing KME Xen, SI, Microreboot, Nooks, ReVirt technologies.

Expected impact:

- Enable COTS systems to endure, recover from repeated attacks.
- Emphasis on kernel increases impact.

Extra slides

Project milestones

Phase 1 (6 months)

- Prototype demonstrating basic detection and repair functionality

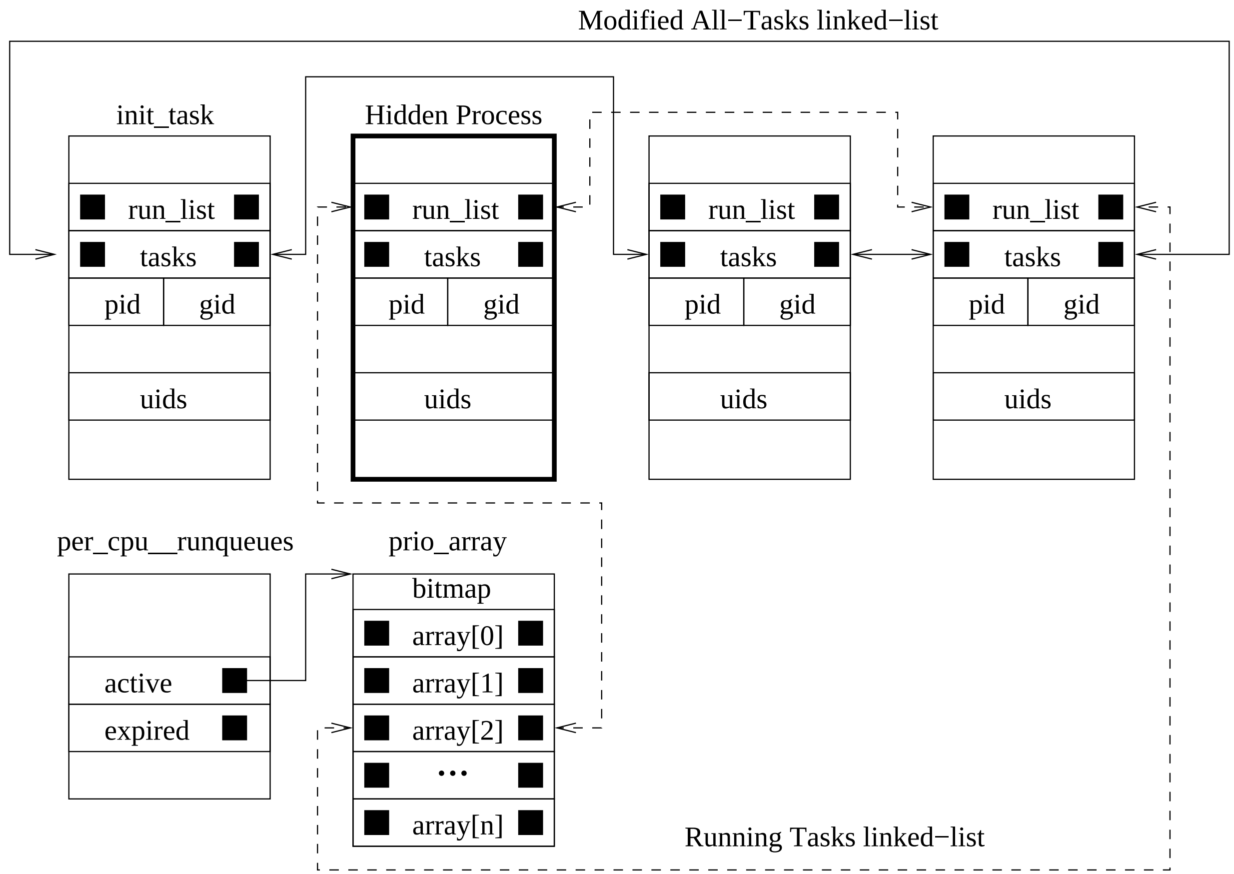
Phase 2 (6 months)

- Prototype demonstrating basic learning functionality

Phase 3 (6 months)

- Final prototype capable of meeting full project detection, repair, and learning requirements.
- Red team evaluation.

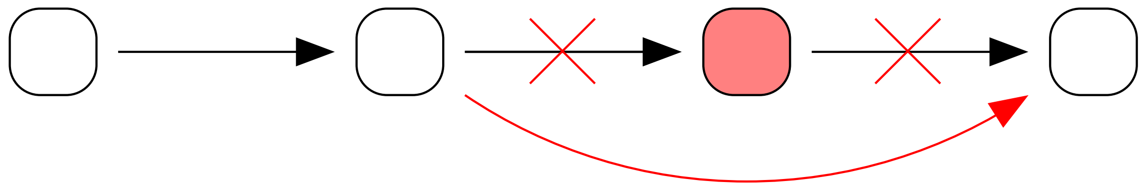
Semantic Integrity Constraints



- Diagram from [Petroni 2006].
- Changes to many interrelated dynamic structures.

Tampering with dynamic data

Dynamic data:



Q1: How to distinguish:

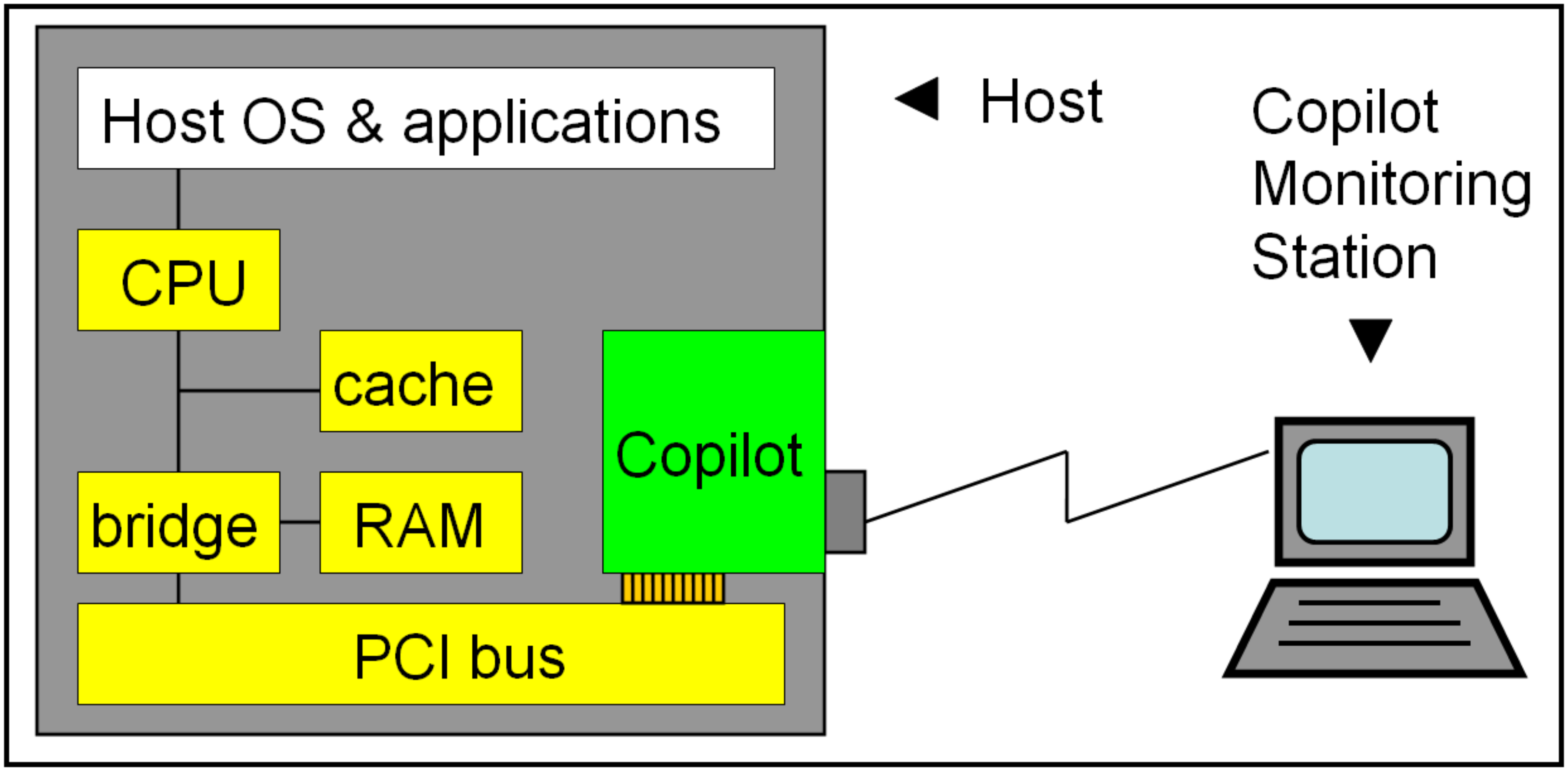
- good state
- bad state
- inconsistent mid-update state

A: Manual source analysis, semantic integrity [Petroni 2006], manual repair action implementation.

Q2: What if perfect surgical repair isn't possible?

A: Approximate with Microreboot, Nooks, ReVirt.

Copilot



\+ Board provides good monitor isolation.

– Registers not visible. – Can't pause the machine.