Observation Rooms for Program Execution Monitoring

Liviu Iftode

Department of Computer Science

Rutgers University

Distributed Computing Laboratory (DisCo Lab) at Rutgers University

- Focus on Network-Centric Systems
- Two research areas
  - Defensive architectures
  - Pervasive computing
- Defensive Architectures
  - Self-Monitoring for Availability and Security
- Pervasive Systems
  - Distributed Embedded Systems, Vehicular Computing
- People
  - Ten graduate students
  - Visiting students: Finland, France, India, Romania, Spain
- International collaborations
  - INRIA/IRISA, UPC Barcelona, University of Cyprus, University of Helsinki, University Paris 6, Technical University of Bucharest,

***STOP: 0x000000D1 (0x00000000, 0xF73120AE, 0xC0000008, 0xC0000000)

A problem has been detected and Windows has been shut down to prevent damage
to your computer

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen, restart your
computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a
new installation, ask your hardware or software manufacturer for any Windows updates
you might need.

If problems continue, disable or remove any newly installed hardware or software.
Disable BIOS memory options such as caching or shadowing. If you need to use Safe
Mode to remove or disable components, restart your computer, press f8 to select
Advanced Startup Options, and then select Safe Mode.

*** WXYZ.SYS - Address F73120AE base at C0000000, DateStamp 36b072a3

Kernel Debugger Using: COM2 (Port 0x2f8, Baud Rate 19200)
Beginning dump of physical memory
Physical memory dump complete. Contact your system administrator or
technical support group.

Why do we get the "Blue Window" ?

- Software is too complicated for humans
- Windows is too complicated for humans
- Success is easier to measure for performance than for reliability
- Reliability is too expensive
- Microsoft people do not attend this conference
- Other reasons?

User Humiliation

- Rebooting is not a solution
  - Destructive: it destroys state
  - Disrupting: it takes time
  - Offending: need the power button to “convince” the computer to return to work

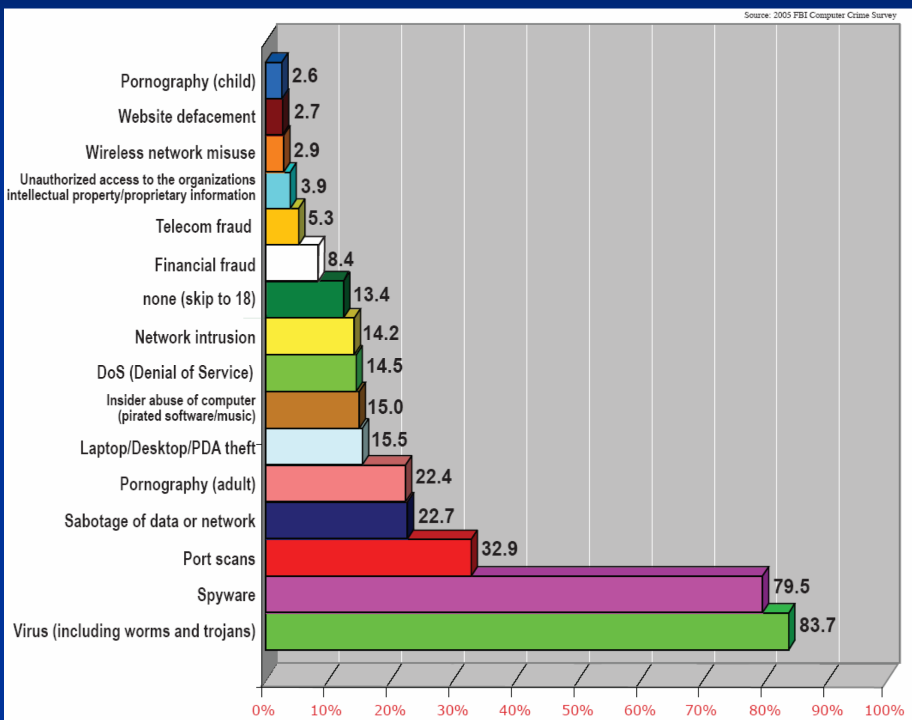
Frustration Scalability

- How to automatically detect a system failure, recover the application state and resume it immediately on another machine?

2004: Mars Rover's Incident

- There is no 100% software reliability even at NASA
- Rebooting does not always work
- How to handle the unexpected?

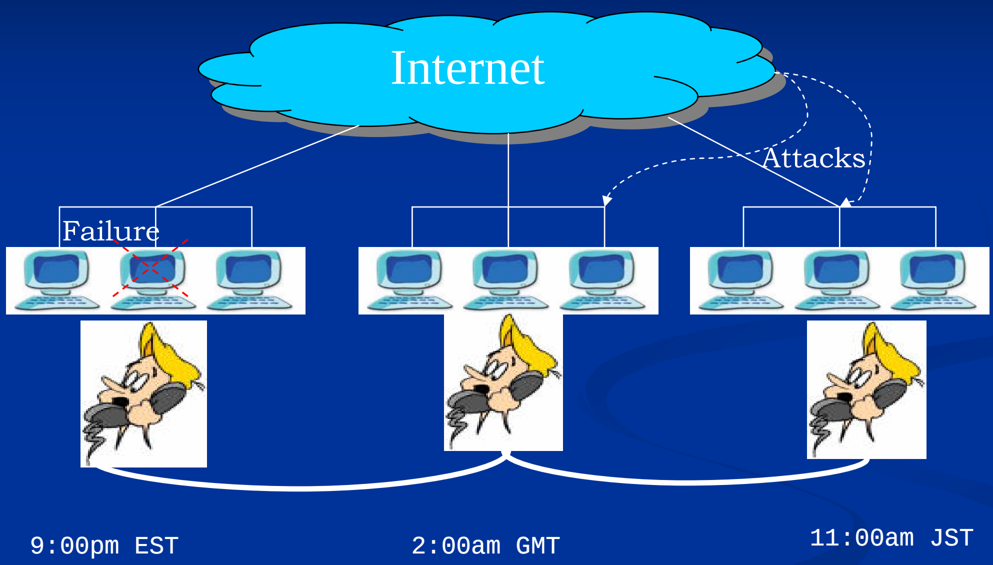
2005, FBI Computer Crime Survey



Software Bugs Make Computers Vulnerable

- Vulnerabilities attract attacks
- Damage spreads fast because attacks execute automatically
- Intrusion detection is not enough
- Early detection must be followed by automated containment
  - Monitor system behavior to discover suspicious anomalies
  - Execute containment actions automatically when attacks are detected

Planetary-Scale Service Maintenance



- Human operators, phone calls and emails are slow and do not scale

Problem and Solution

- Despite decades of research, computer programs
  - continue to have bugs and fail
  - remain vulnerable to various attacks
  - require human intervention for healing
- Defensive Architectures: augment computer systems with trusted and standalone *observation rooms* for
  - Execution/Communication monitoring
  - Failure prediction and detection
  - Healing actions
- *Observation rooms*
  - Passive: monitoring
  - Active: monitoring, diagnosis and healing

Observation Room Requirements

- Isolated: external to the OS
- Autonomous: without involving local OS
- Non-intrusive: no need to change the OS or the network protocols
- Highly available: work even when the OS fails
- Trustworthy: OS cannot alter its functioning
- Responsive: OS cannot indefinitely delay its operations
- Efficient and scalable: low overhead
- Accurate: few false positives
- Comprehensive: few false negatives
- Flexible: programmable
- Easy to deploy
- Distributed/Cooperative monitoring: Inter-room communication

Main Questions:

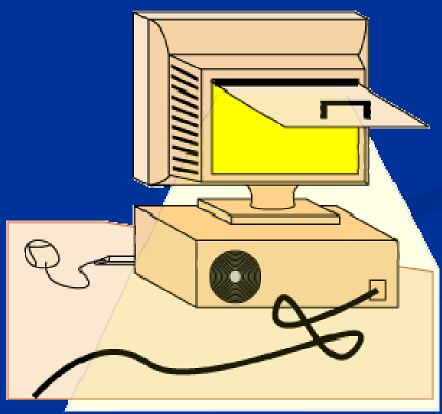
- How to implement an *observation room*?
- Where to place it?

Outline

- Introduction
- Remote Observation Rooms (Backdoors)
- Virtual Observation Rooms (Paladin)
- Network Observation Rooms (FileWall)
- Observation Rooms for MultiCore Processors
- Conclusions

Remote Observation Rooms

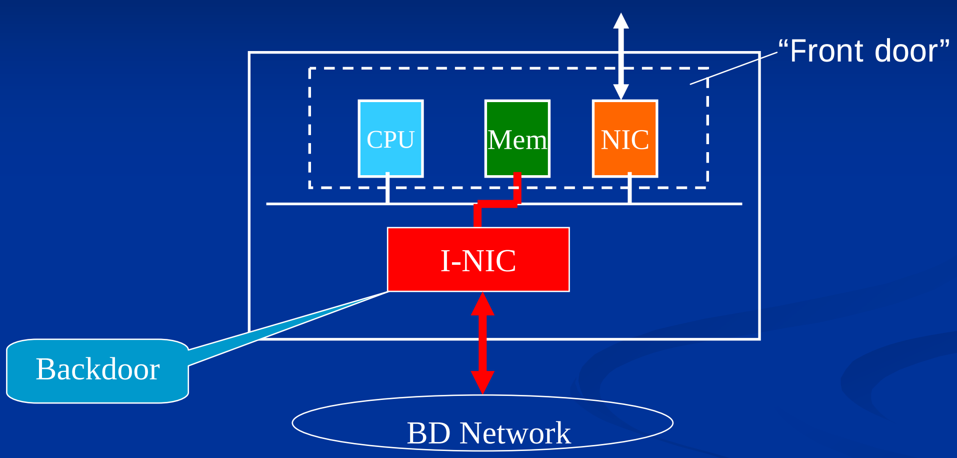
- Use another computer to host the observation room
- Remote access supported through a “friendly” *backdoor*



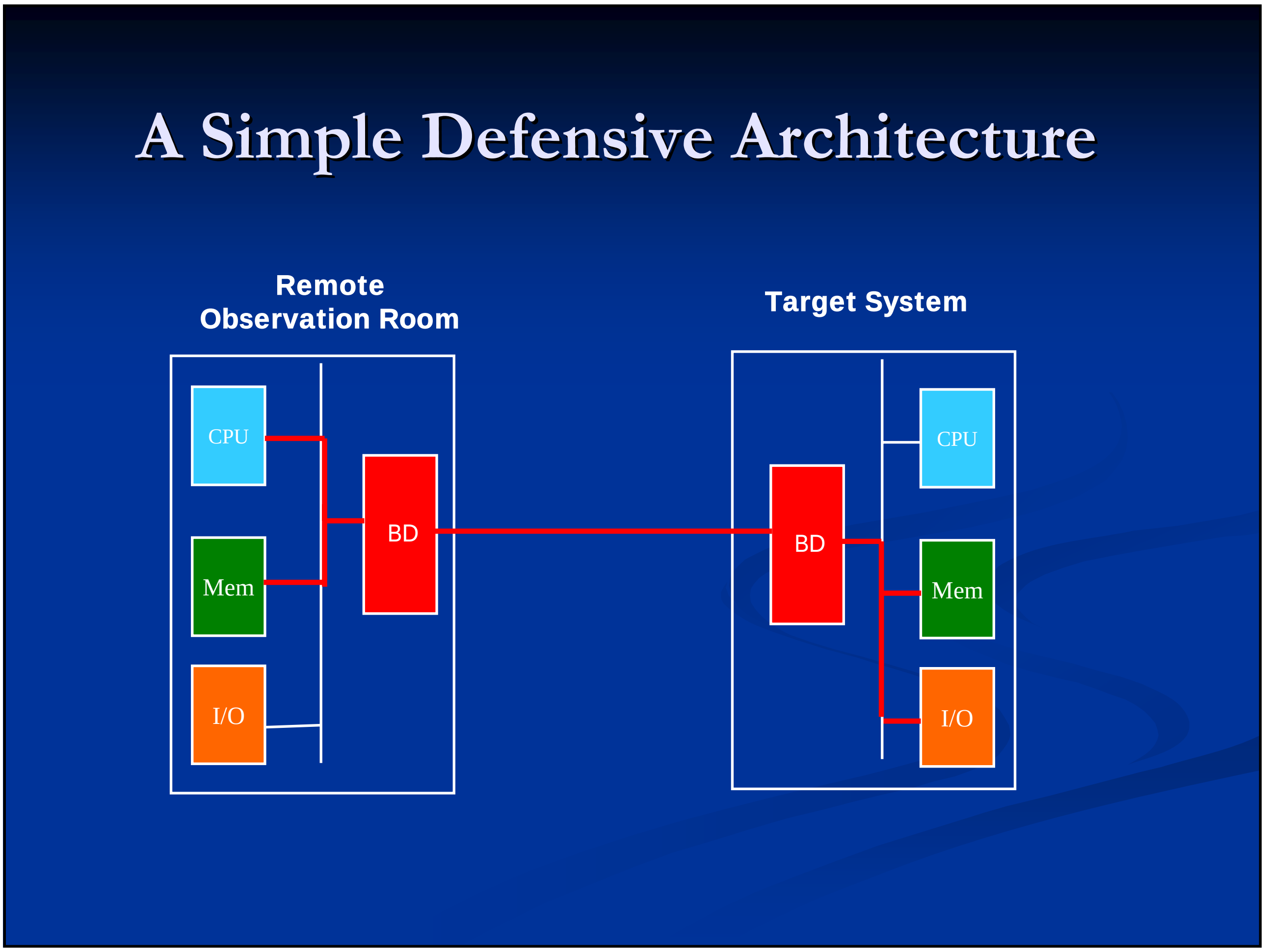
Backdoor: a hidden software or hardware mechanism, usually created for testing and troubleshooting

--American National Standard for Telecommunications

A Hardware Backdoor



- Programmable/intelligent network interface card (I-NIC) connected to a high-speed private network
- Can access computer memory/resources without OS intervention


A Simple Defensive Architecture
Remote
Observation Room
Target System
CPU
Mem
I/O
BD
BD
CPU
Mem
I/O

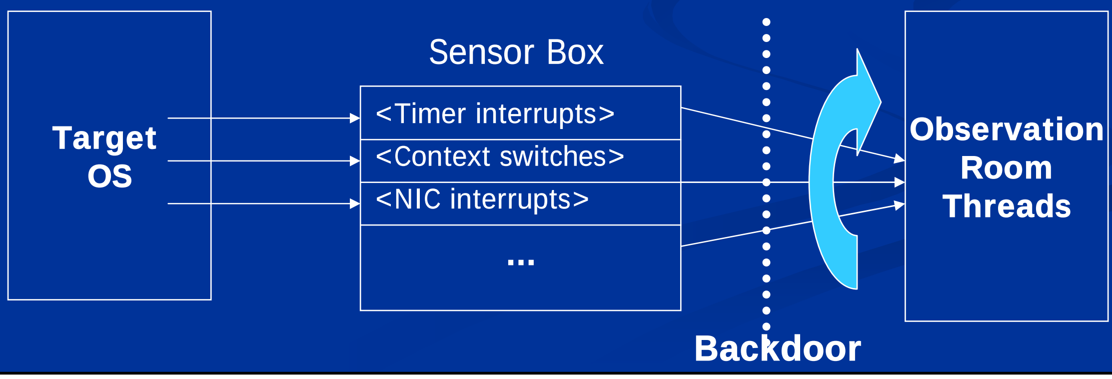
A Sensor Box

- Collection of health indicators (sensors) in the target OS memory used for monitoring
  - <ID, Type, Threshold, Value>

Sensor Type	Threshold
Progress	Update deadline
Level	Max/Min value
Pressure	Max number of events

Failure Detection using Sensor Box

- Progress sensors: number of interrupts
- Target OS updates sensors continuously
- Monitoring thread from the remote observation room read sensors periodically
  - Failure = counter stalled beyond its deadline
- False positive rate vs. detection latency tradeoff



Monitoring and Detection Using Backdoor

Active Remote Observation Rooms

- Repair
  - Identify damaged OS state
  - Modify target OS memory to correct damaged state
- Recovery
  - Continuation box: "essential" OS and application state
  - Extract continuation boxes from the failed system and insert them into healthy systems to resume execution
- Active remote observation rooms are intrusive
  - OS must provide locking for backdoor access and continuation box for remote state extraction
  - Application must cooperate with the backdoor to allow state synchronization

Diagnosis Using Backdoor

Repair Using Backdoor

Case Study: OS State Repairing

- Damaged OS state : resource exhaustion, corrupted data structures, compromised OS, etc.
- Resource exhaustion
  - Attack, overload, system misconfiguration, programming error

Example: A Memory Hog Process

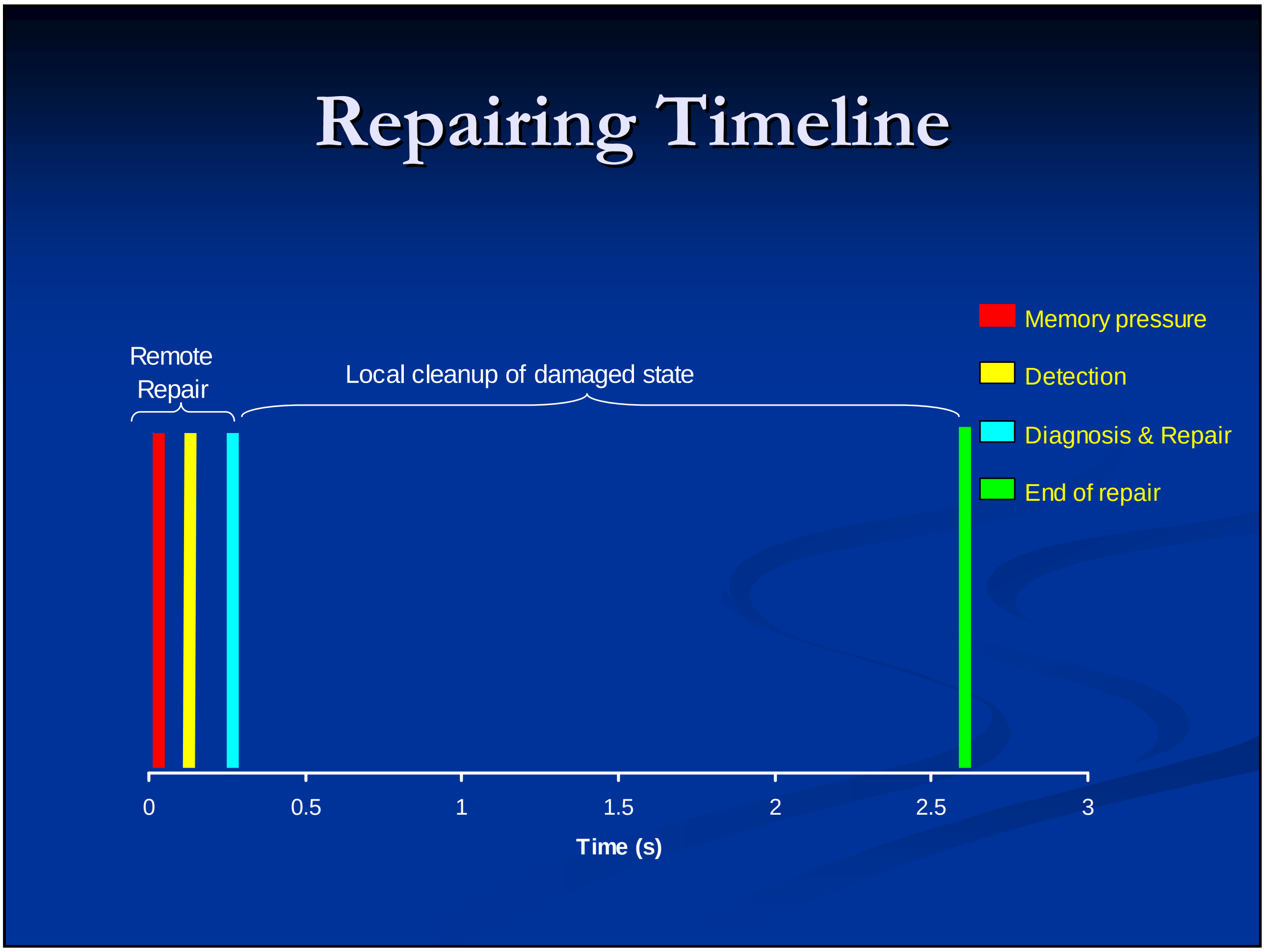
- Program allocates memory in an infinite loop
  - Both memory and swap space will be eventually exhausted
- The computer system freezes
  - Cannot be accessed from console or the network
  - Cannot spawn new processes
  - Cannot handle interrupts
  - Local daemons cannot repair system

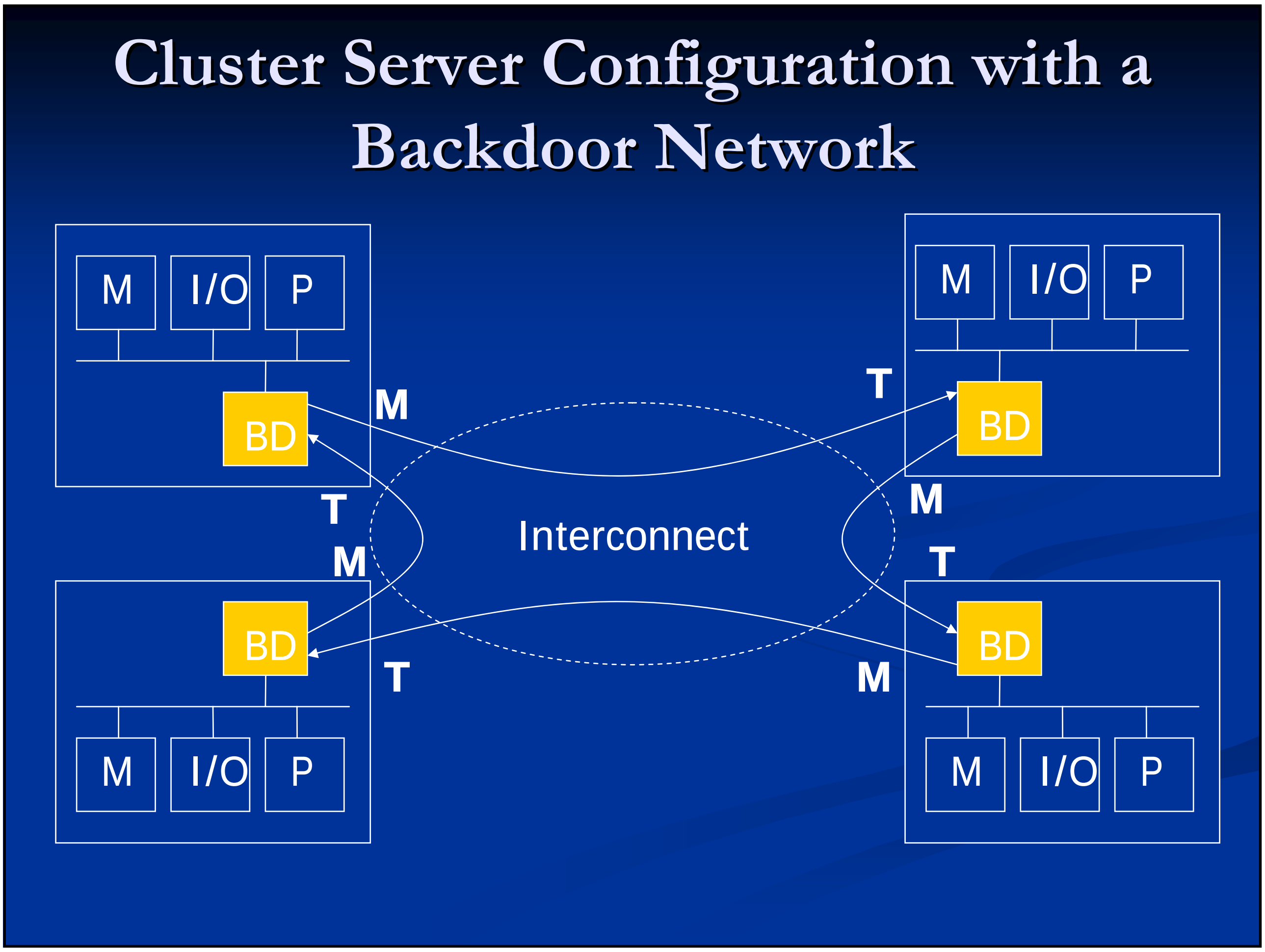
Observation Room Operations

- Monitoring
  - Pressure sensor signals when severe low memory condition is detected
- Diagnosis
  - Target OS externalizes process table and process memory usage statistics
  - Monitoring thread identifies the culprit process
- Repairing
  - Monitoring thread kills culprit by remotely writing a "killing" signal on its signal table

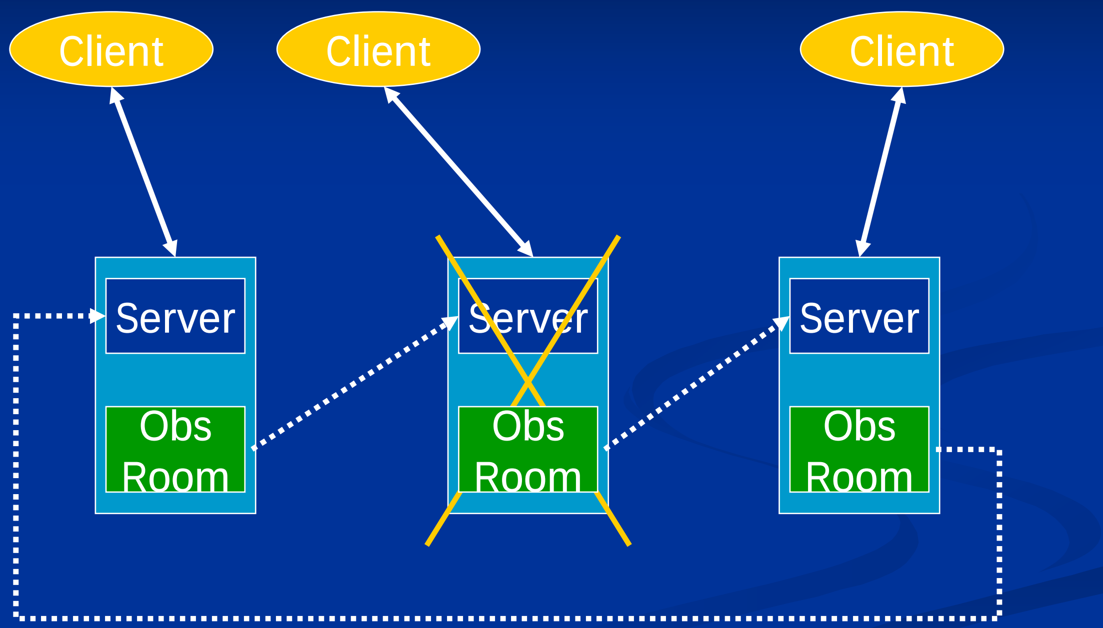
Backdoor Prototype

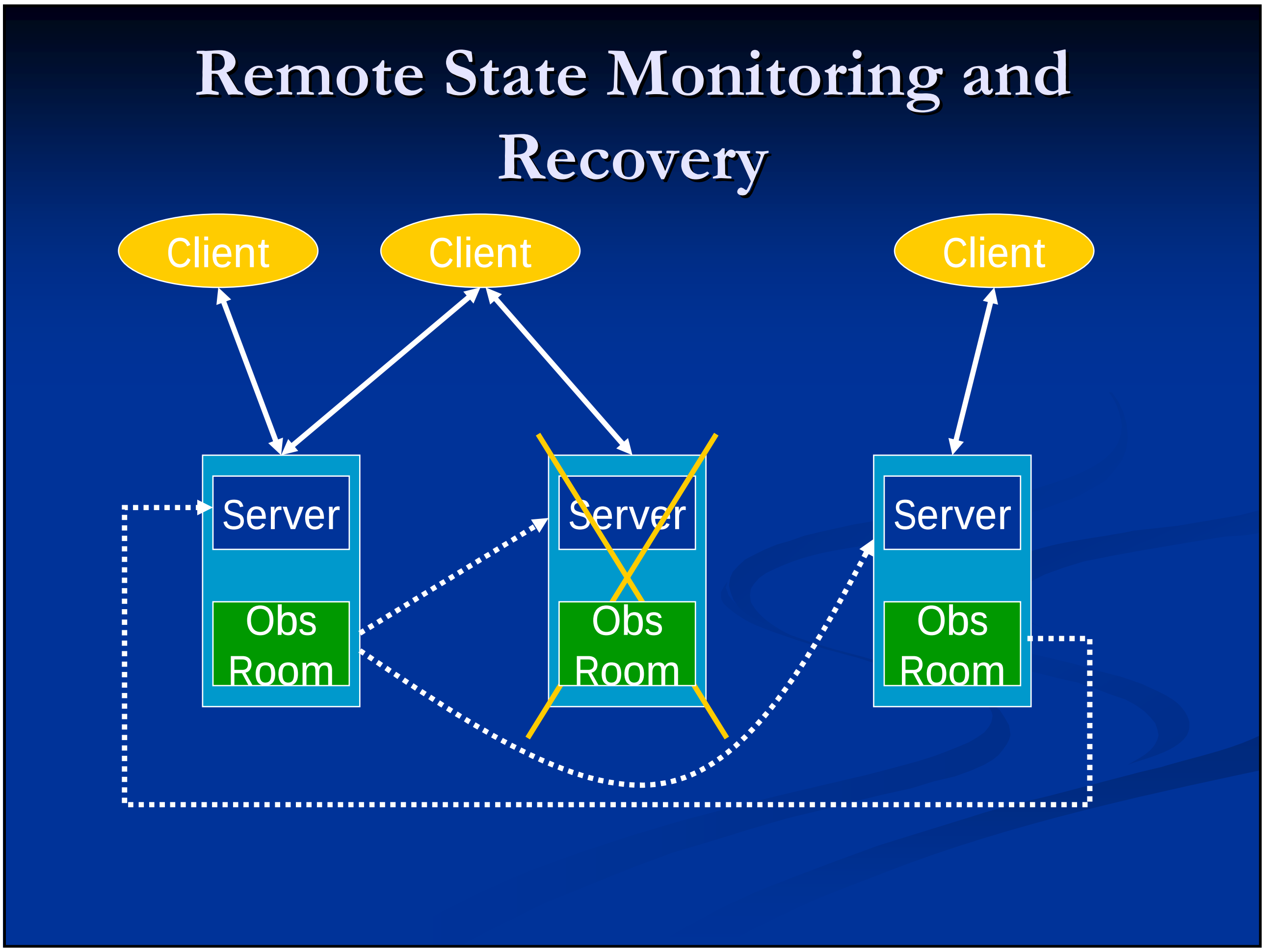
- Implemented on Myrinet LanaiX NIC
  - Modified firmware and low level GM library
- Modified FreeBSD 4.8 kernel
  - Observation room is intrusive for repair/recovery
- Experimental setup
  - Dell Poweredge 2600 servers with 2.4 GHz dual Intel Xeon, 1GB RAM, 2GB swap, Myrinet Lanai X NIC


Repairing Timeline
Remote
Repair
Local cleanup of damaged state
Memory pressure
Detection
Diagnosis & Repair
End of repair
0
0.5
1
1.5
2
2.5
3
Time (s)

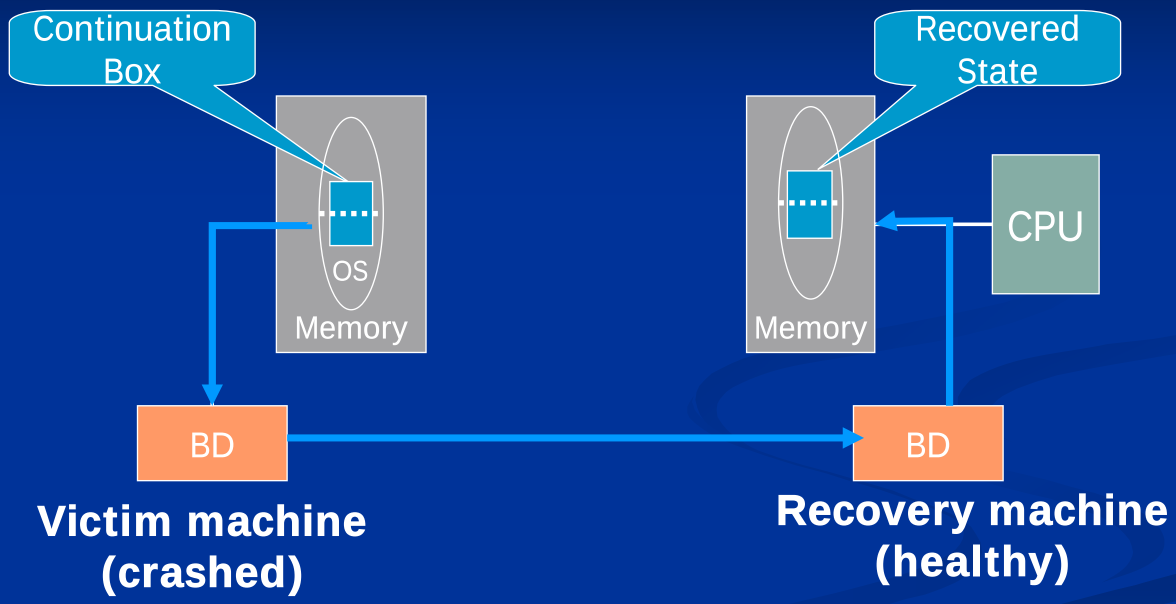

Cluster Server Configuration with a Backdoor Network
M
I/O
P
BD
M
T
T
M
Interconnect
M
T
BD
T
M
M
I/O
P

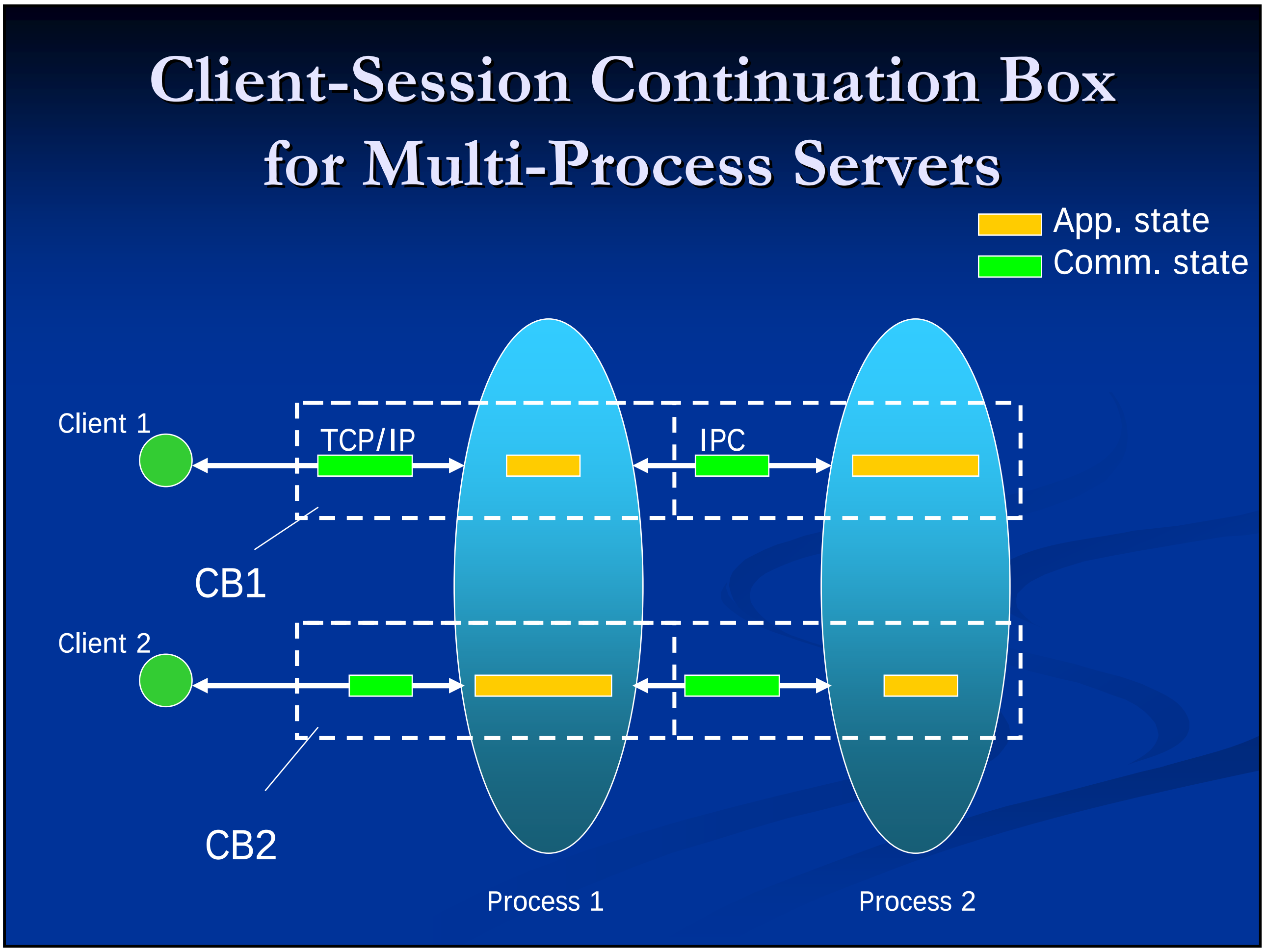
Remote State Monitoring and Recovery




Remote State Monitoring and Recovery
Client
Client
Client
Server
Obs Room
Server
Obs Room
Server
Obs Room

Continuation Box Extraction

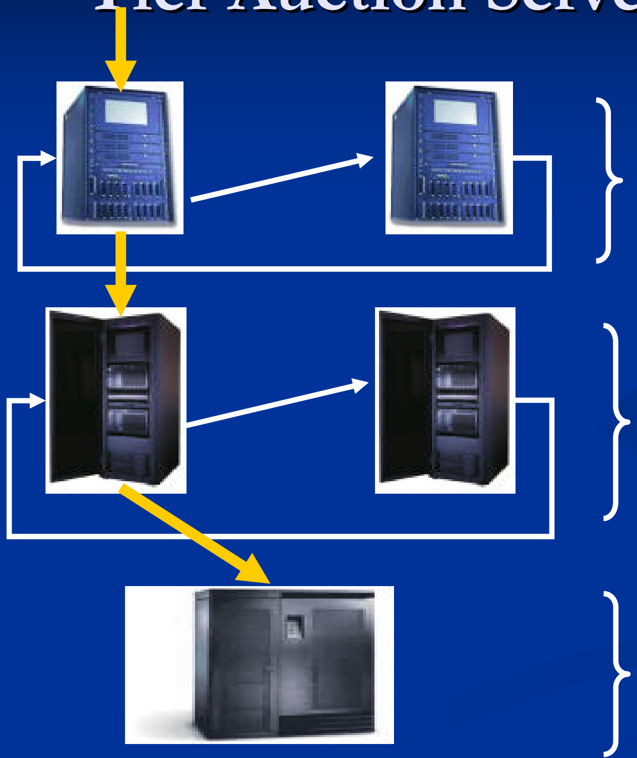



Client-Session Continuation Box
for Multi-Process Servers
App. state
Comm. state
Client 1
TCP/IP
IPC
CB1
Client 2
CB2
Process 1
Process 2

Changes to Make Server Recoverable

```
while (cid = accept()) {
  cbid = create_cb(cid)
  if (import(cbid, &{file_name, offset}) == NULL) {
    receive(cid, file_name)
    offset = 0
  }
  fd=open(file_name)
  seek(fd, offset)
  while (read(fd, block, size) != EOF)  {
    send(cid, block, size)
    offset += size
    export(cbid, {file_name, offset})
  }
}
```

Case Study: Recoverable Multi-Tier Auction Server (RUBiS)



Front-End (FE)

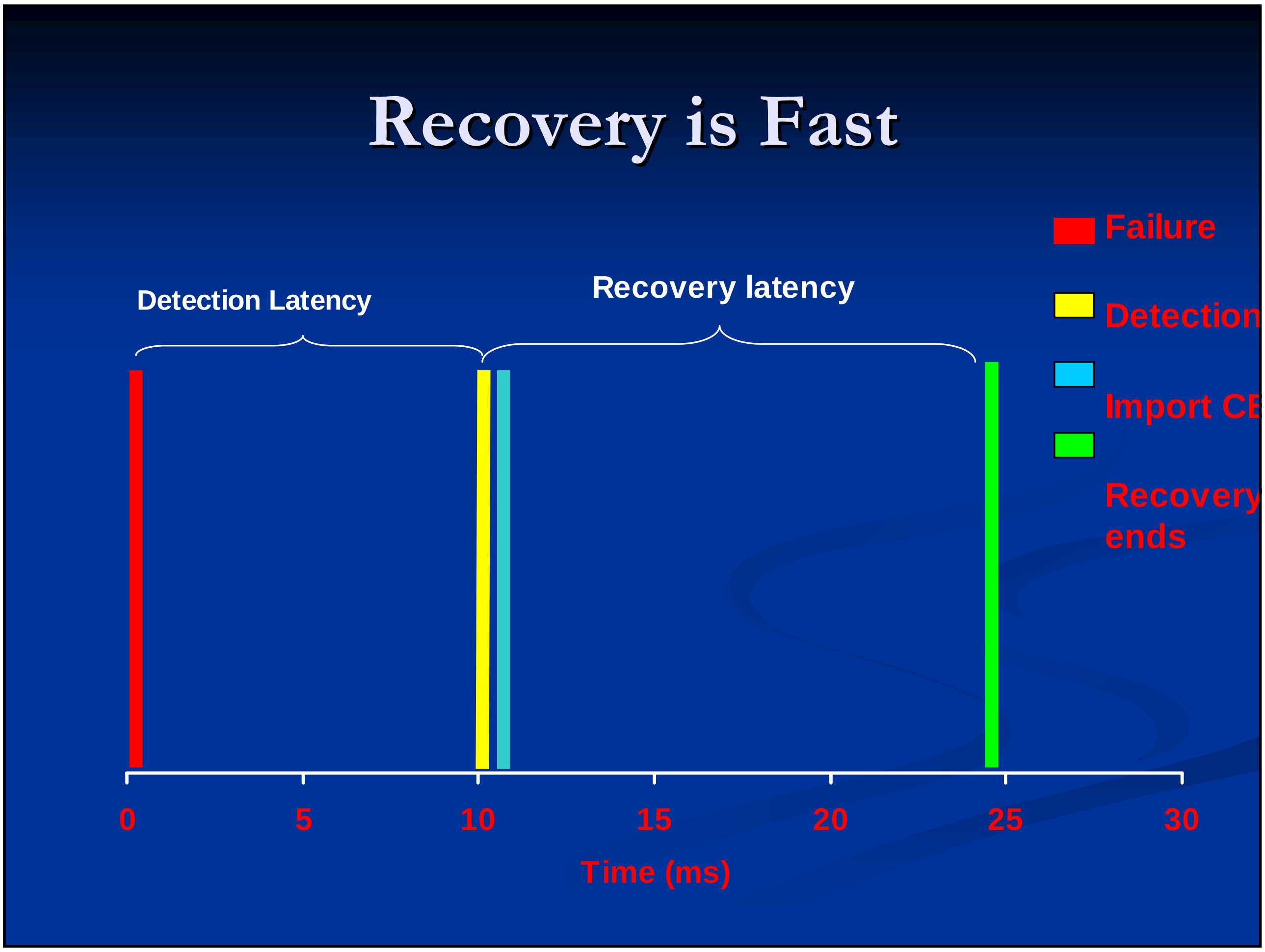
Apache web server

Middle Tier (MT)

JBoss app. server

Back-End

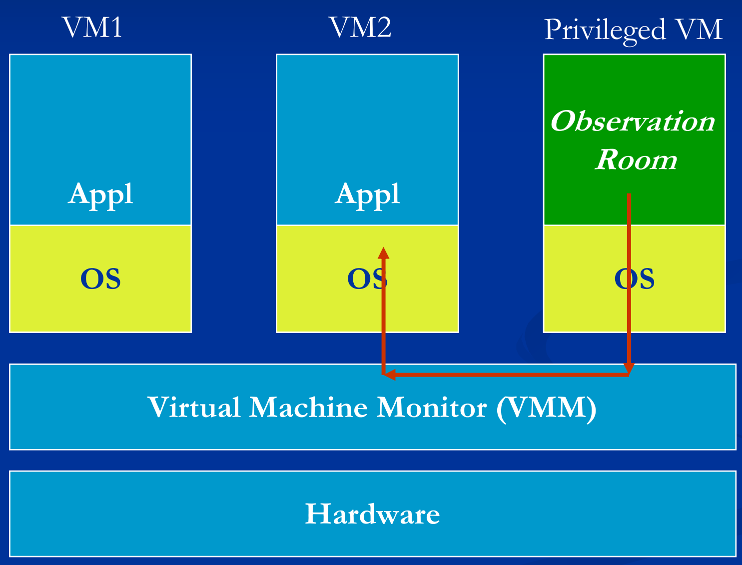
MySQL DB server


Recovery is Fast
Failure
Detection
Import CE
Recovery ends
Detection Latency
Recovery latency
0
5
10
15
20
25
30
Time (ms)

Outline

- Introduction
- Remote Observation Rooms (Backdoors)
- Virtual Observation Rooms (Paladin)
- Network Observation Rooms (FileWall)
- Observation Rooms for MultiCore Processors
- Conclusions

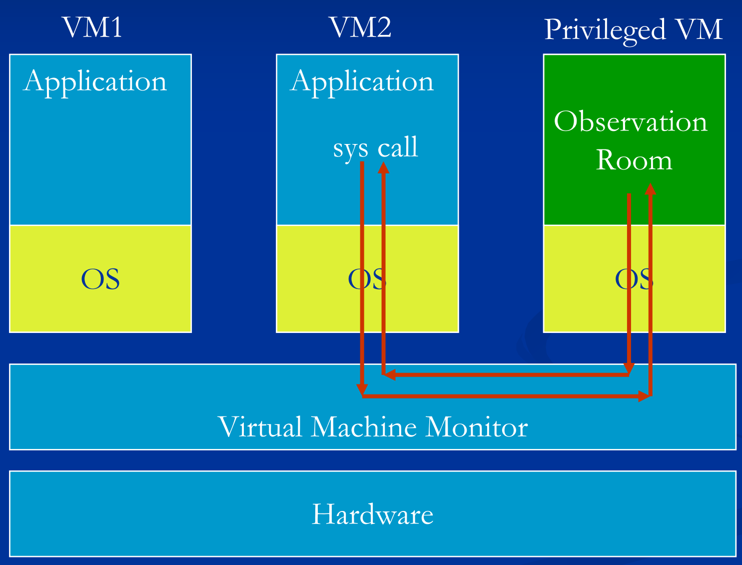
Virtual Observation Rooms



Virtual Observation Rooms (cont'd)

- Complete access to the guest OS and application state
- Virtual backdoors must be provided by the VMM
- Two implementations
  - Asynchronous/Continuous Monitoring: similar to the remote observation rooms
  - Synchronous/Event- Driven: system call interception

Synchronous Virtual Observation Rooms

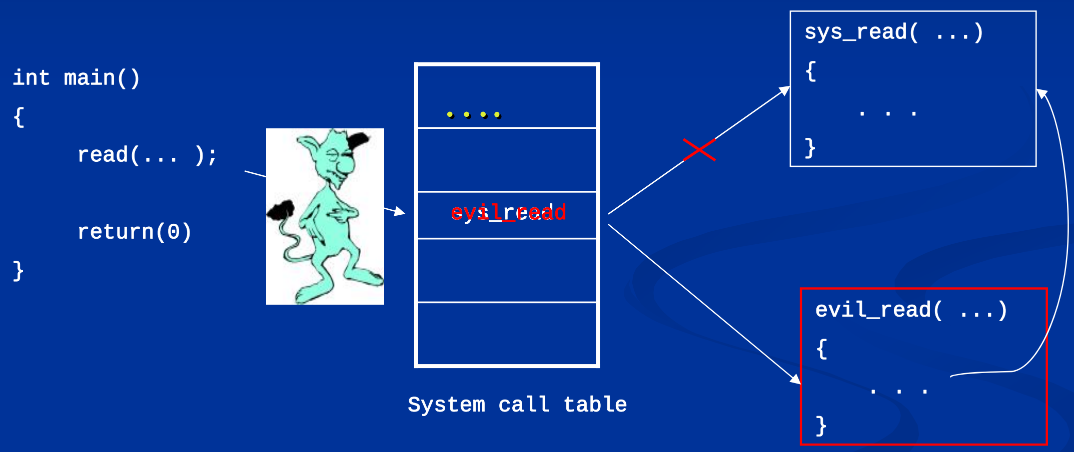


Case Study: Rootkit Detection

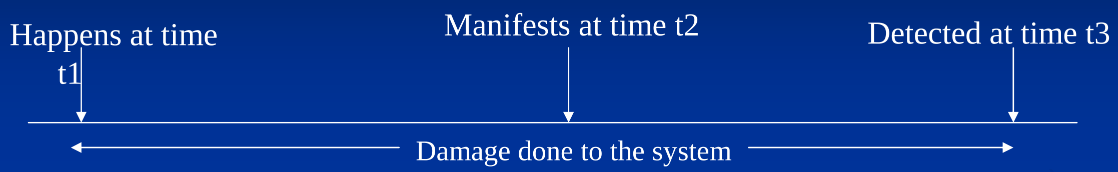
- Rootkit: collection of tools used by the attack to hold root privileges on the compromised system.
- Rootkit hiding mechanisms:
  - Replace system binaries like *ps* and *netstat*
  - Replace shared libraries
  - Replace entries in system call table
  - Replace entries in interrupt descriptor table (IDT)
  - Replace kernel text.
- Synchronous virtual observation room tasks:
  - Detect intrusion
  - Contain damage without restarting the system

Rootkit Example

- System call hijacking



Intrusion Timeline



- Damage done to the system from t1-t3 needs to be discovered and undone – typically done manually
- Ideally intrusion should be detected at t1(Prevention)
  - Easier for known attacks
  - Hard for new/unknown attacks
- Intrusion Detection Systems (IDS)
  - Move t3 closer to t2
  - Ideally move t3 close to t1

Observation Room for Intrusion detection

- Define ***protected zones***
  - In memory
  - On file system
- **Detect** attempted illegal access to protected zones.
- **Track** dependencies between processes and between files and processes
- **Contain** damage in progress using dependency information

Protected Zones

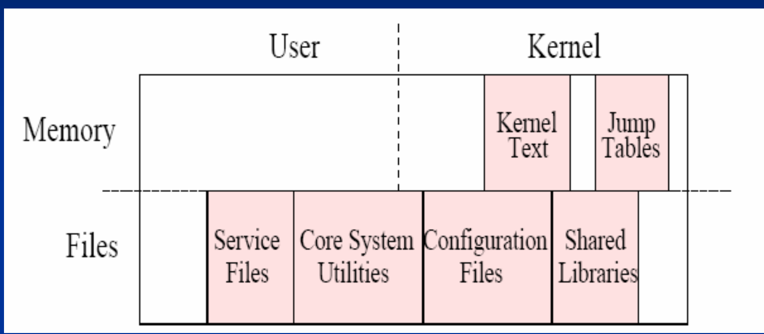


Fig 1: Protected Zones

```
/bin          RD_X
/sbin         RD_X
/boot         RD_X
/usr/bin      RD_X
/usr/sbin     RD_X
/etc/passwd   RD_ONLY
```

Fig 2: Sample policy file (protected files)

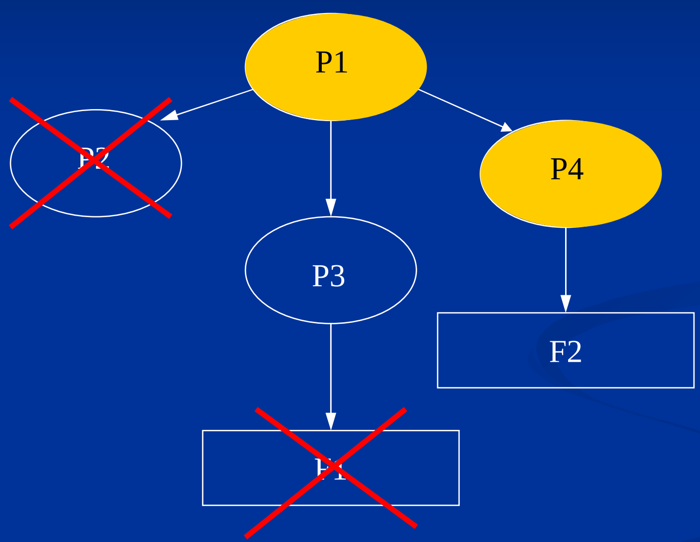
```
System call table
Interrupt descriptor table
Kernel text
```

Fig 3: Protected Memory Zones

Track Dependencies

- **Infer dependencies**
  - Parent-child relationships between processes
  - Dependencies between files and processes
- **Store dependencies**
  - Dependency tree stored in a database
  - Dependency tree must be kept small to allow fast response

Dependency Rules and Tree



P1 creates P2

P2 exits

P1 creates P3

P1 creates P4

P3 creates F1

P4 creates F2

F1 is deleted

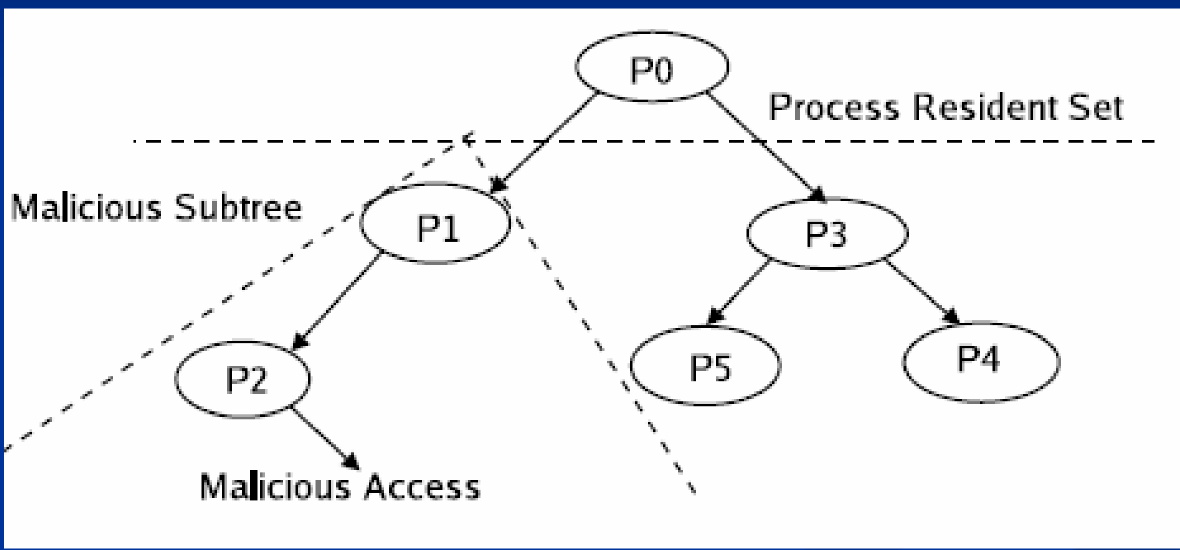
P4 exits

P1 exits

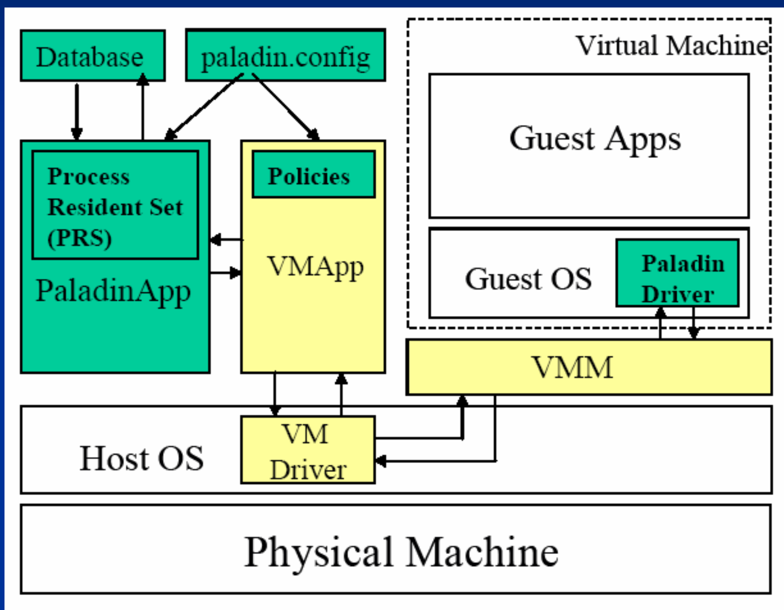
Automatic Containment

- Use dependency tree to locate the largest possible malicious subtree that includes the process which performed the malicious access
- Kill all processes from the malicious subtree to stop the ongoing damage
- Many challenges

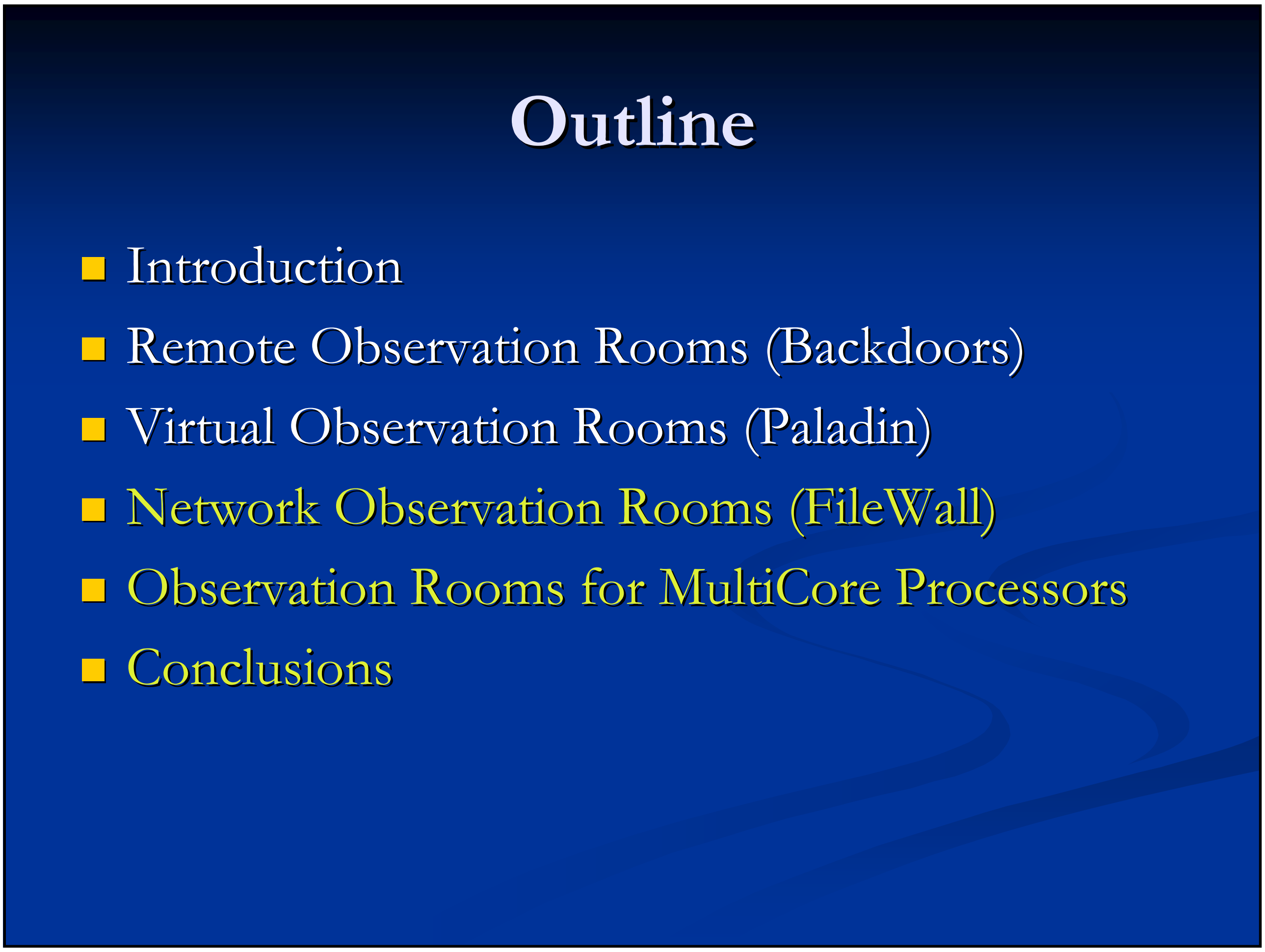
Containment Algorithm



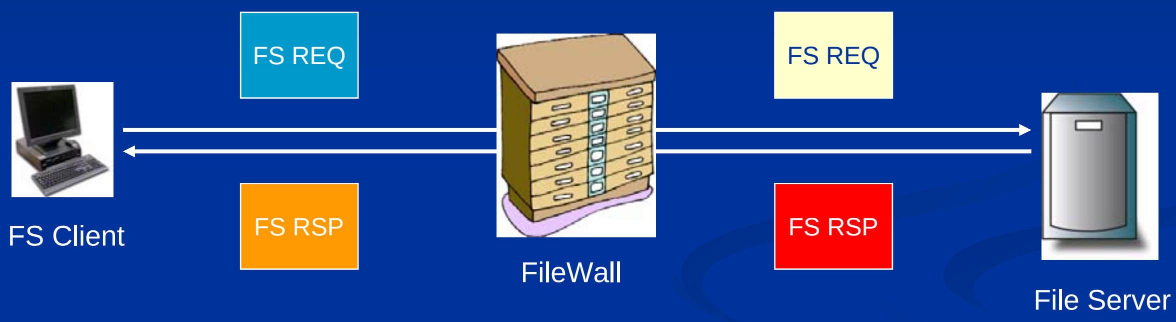
Prototype (Paladin*) on VMware



- Successfully tested against 27 rootkits available for Linux

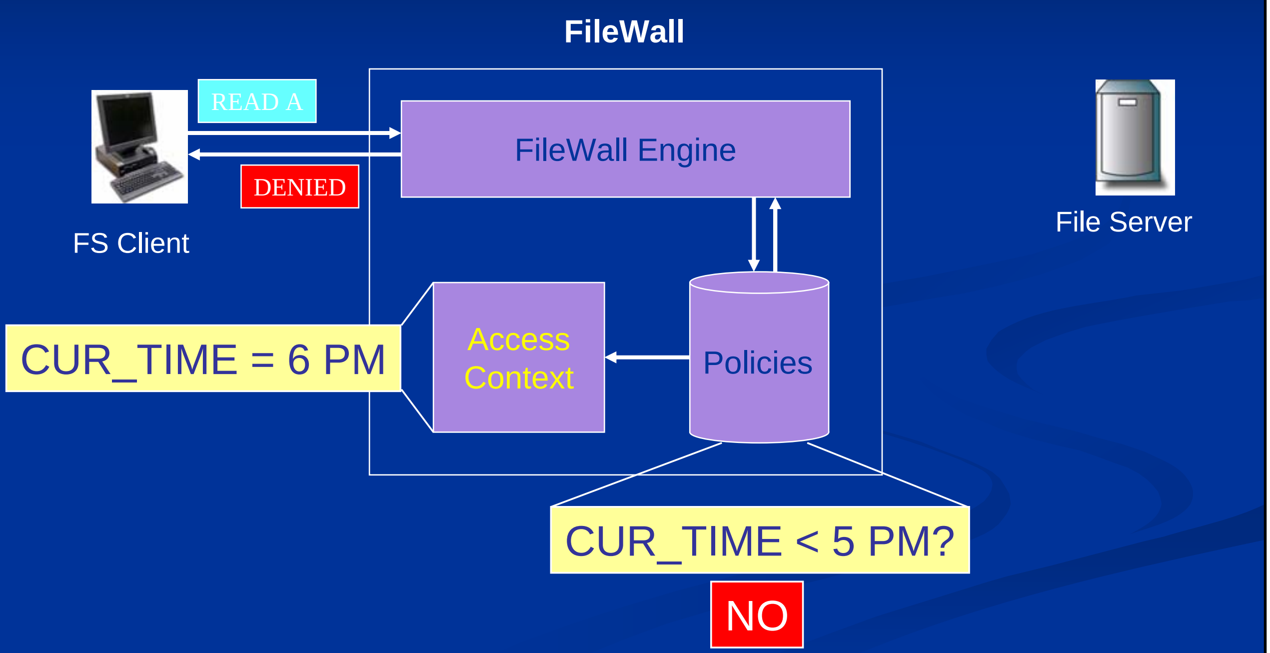

Outline
Introduction
Remote Observation Rooms (Backdoors)
Virtual Observation Rooms (Paladin)
Network Observation Rooms (FileWall)
Observation Rooms for MultiCore Processors
Conclusions

FileWall: A Network Observation Room for Network File Systems

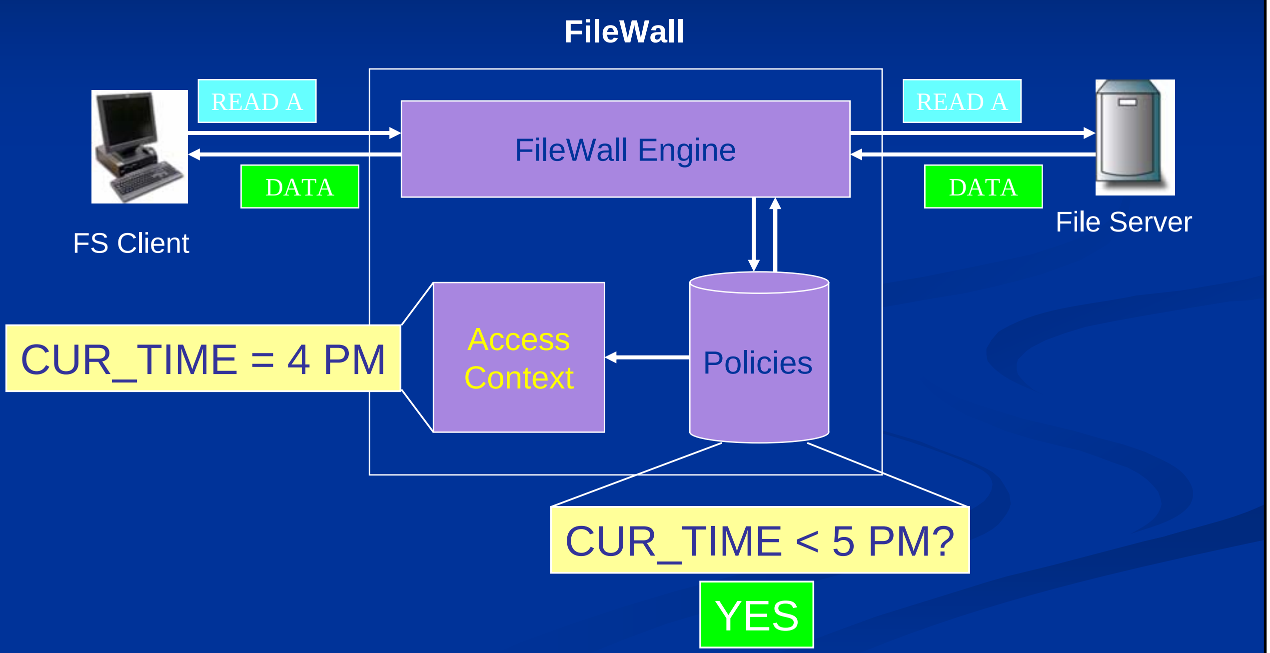


- File system accesses translated into messages
- Passive FileWall: file system message monitoring
- Active FileWall
  - Perform message transformations
  - Implement file access policies and other file system extensions

Example: Temporal Access Control using the FileWall



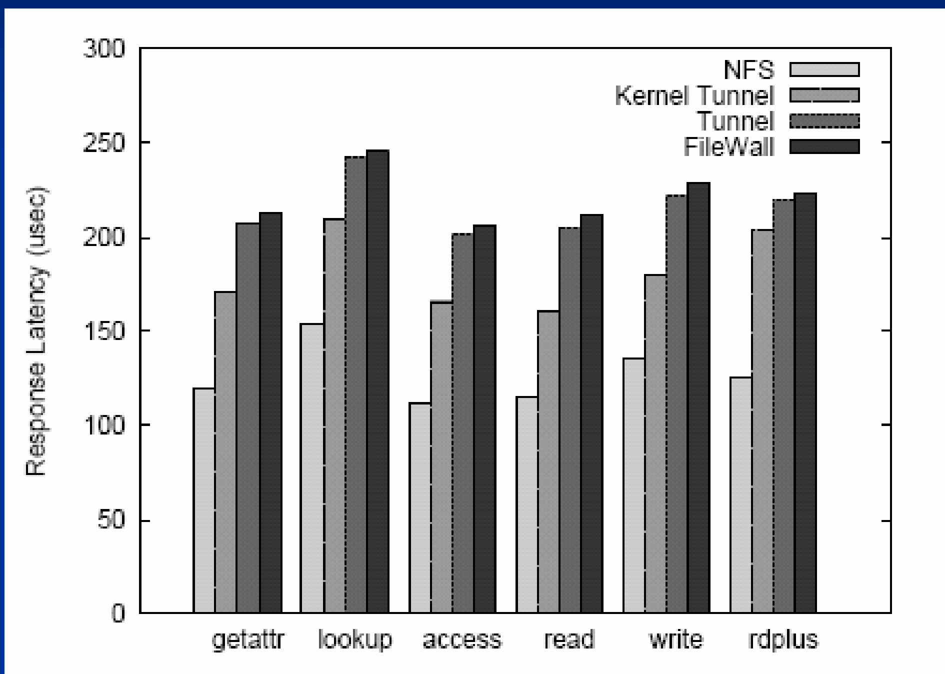
Example: Temporal Access Control using the FileWall



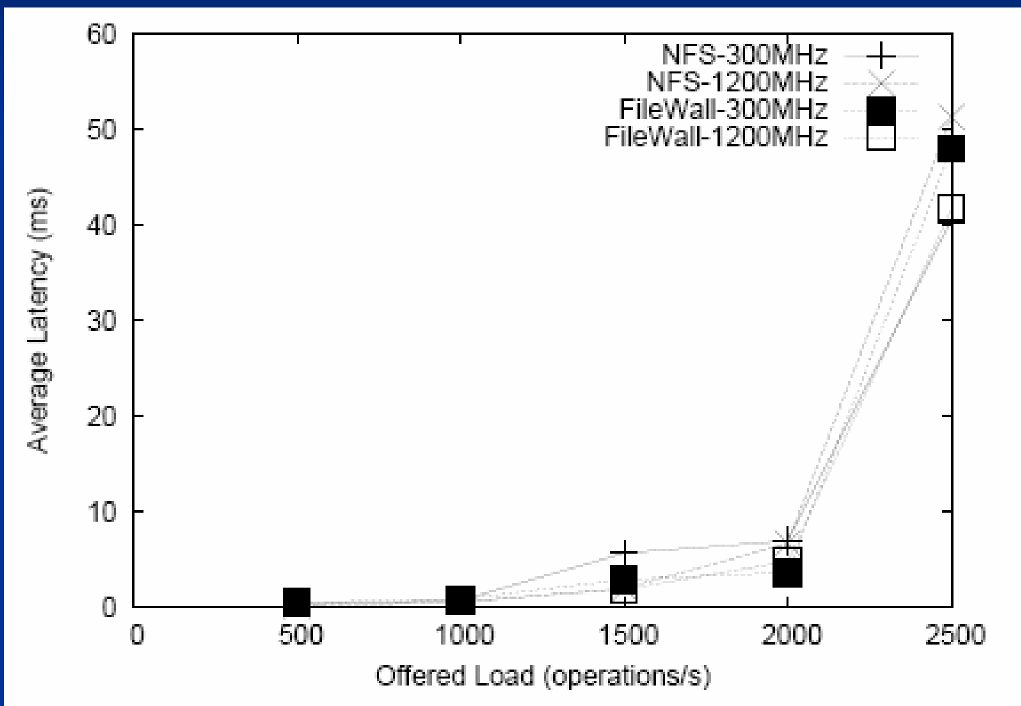
FileWall Prototype

- FileWall
  - Click Modular Router
  - NFS over UDP
- Policies
  - Statistics Monitoring
  - Temporal Access Control
  - File Handle Security
  - Client Transparent Failover

Interposition Overheads



FStress Performance



Outline

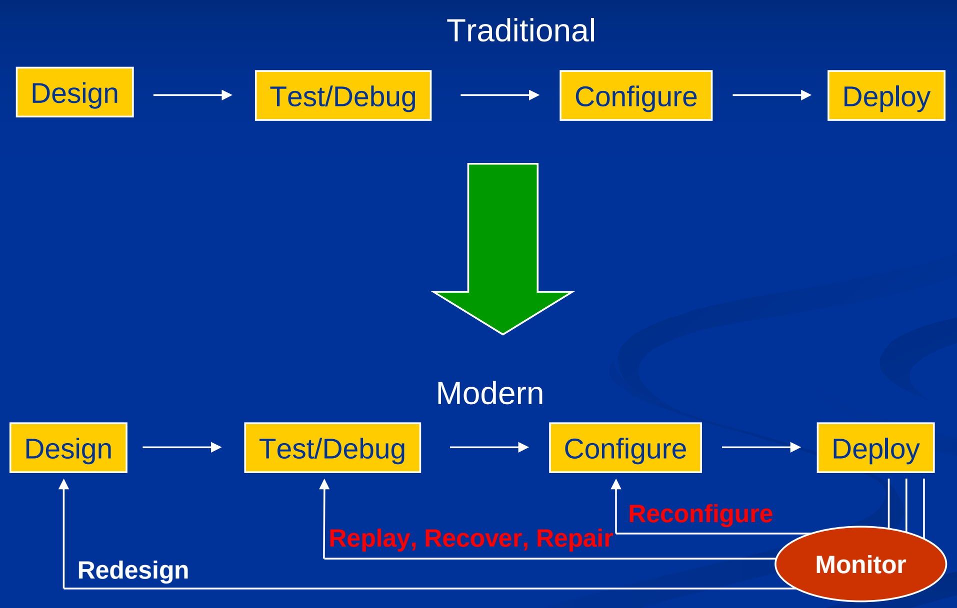
- Introduction
- Remote Observation Rooms (Backdoors)
- Virtual Observation Rooms (Paladin)
- Network Observation Rooms (FileWall)
- Observation Rooms for MultiCore Processors
- Conclusions

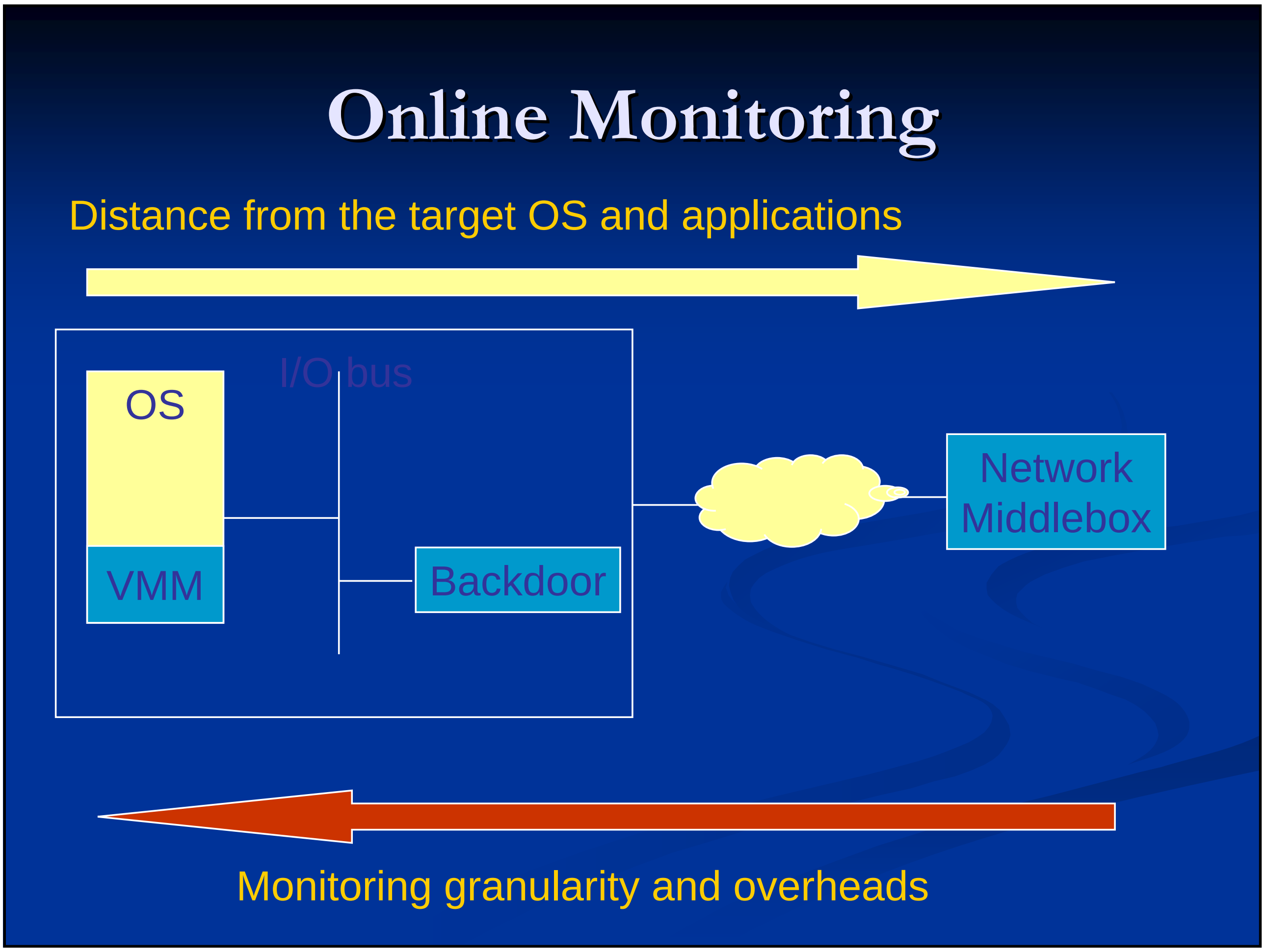
Continuous Monitoring

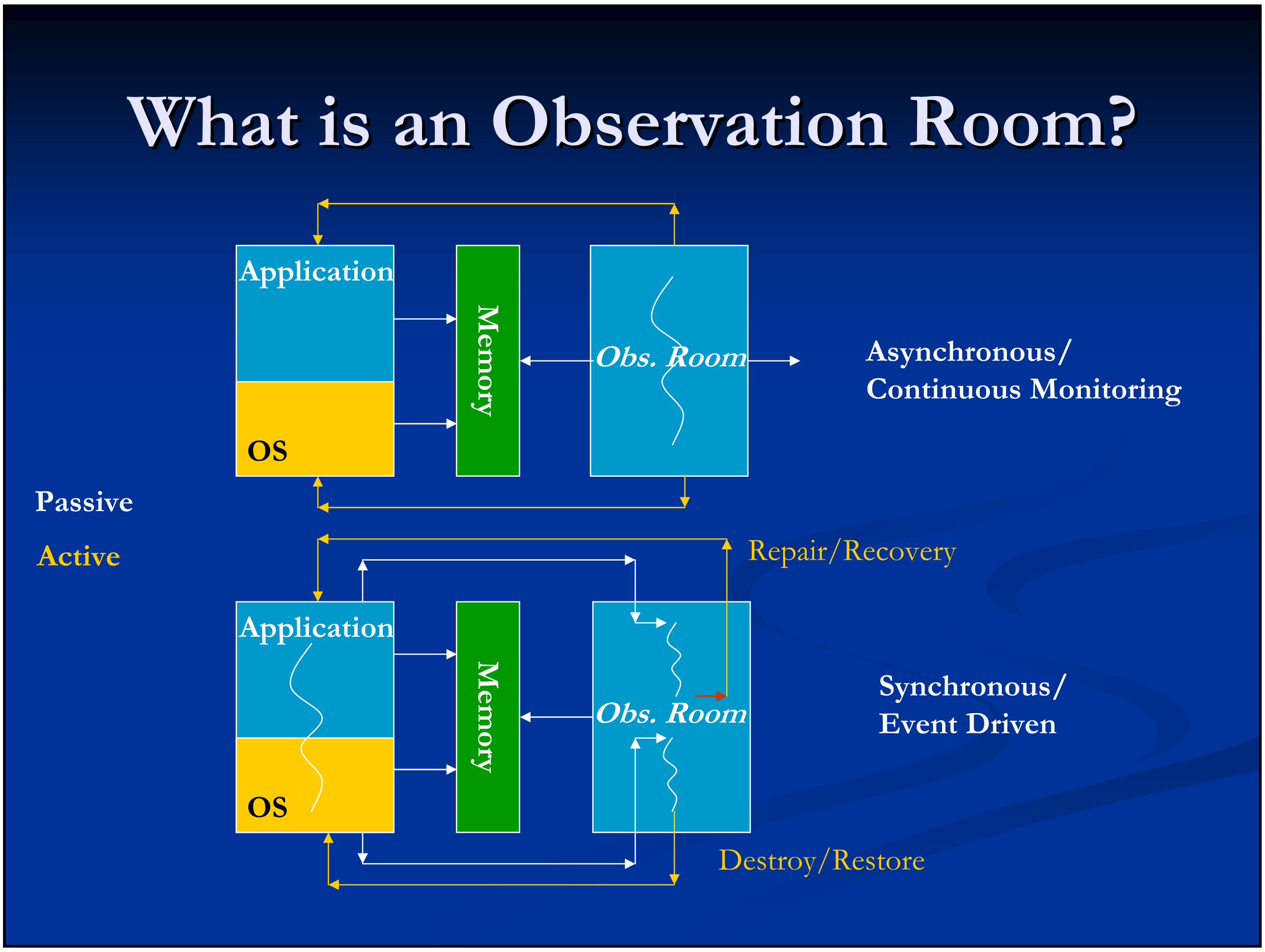
- MultiCore Architectures: an opportunity to trade performance for reliability
  - Observation room hosted on dedicated core
  - Continuous monitoring
- Application level monitoring
  - Application threads (application cores)
  - Monitoring thread (observation room core)
- Monitoring thread
  - Shares the address space and is co-scheduled with the application threads
  - Its local data is protected from the application threads
  - Checks value-based invariants: (e.g. return addresses)
- Asynchronous or synchronous monitoring

Conclusions

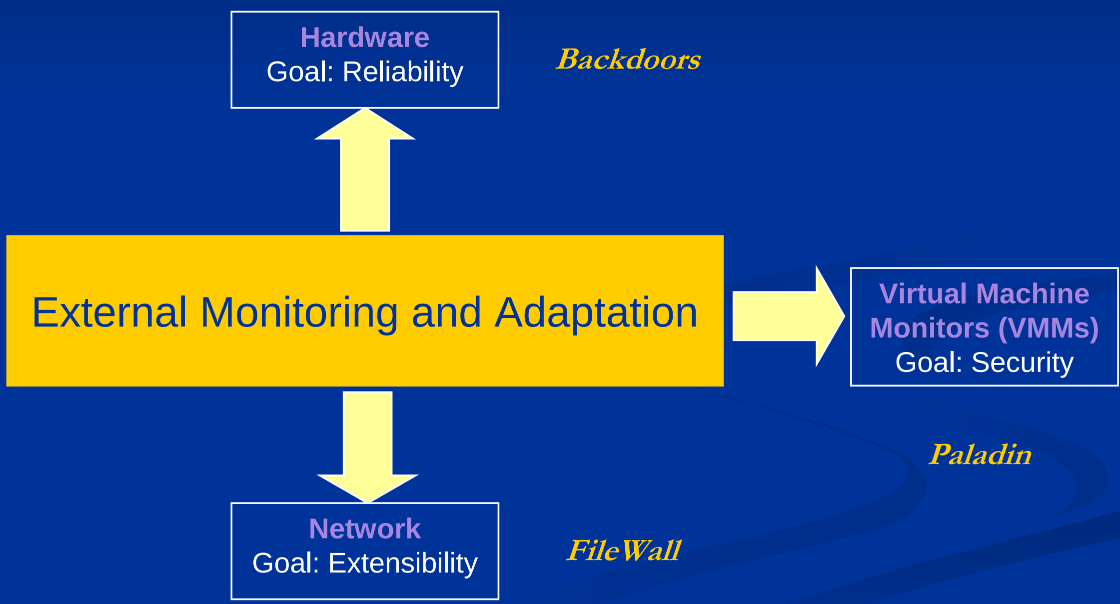
Online Monitoring Has a Role in Software Lifecycle




Online Monitoring
Distance from the target OS and applications
OS
VMM
I/O bus
Backdoor
Network
Middlebox
Monitoring granularity and overheads


What is an Observation Room?
Application
OS
Memory
Obs. Room
Asynchronous/
Continuous Monitoring
Passive
Active
Repair/Recovery
Application
OS
Memory
Obs. Room
Synchronous/
Event Driven
Destroy/Restore

Our Observation Rooms



More Conclusions

- Observation rooms can be placed at various distances from the target: remote system, network, virtual machine
- Consume resources; trade some performance for possibly extra reliability and security
- Provide system survivability, complementary to existing reliability solutions
- Limitations: false positives and false negatives
- Big challenge: observation room programmability
- Big opportunity: multicore processor architectures

Thank You!

http://discolab.rutgers.edu