Analytics and Virtualization: Diagnostic Toolkit for the ’10s

Lisa Spainhower

June 26, 2010

Problem Determination Challenges

Problem determination in complex IT enterprises is a major pain point

- Bridge calls and finger-pointing

Not necessarily true that anything is 'broken' and can be 'fixed'

- Incompatibilities
- Configuration mis-matches

Tasks are complex and skill-intensive

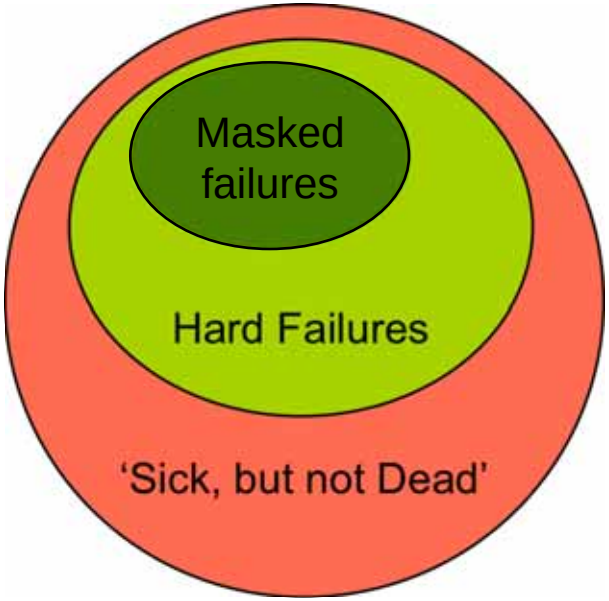
- Many processes, steps, tools
- Often multiple vendors to coordinate

Multiple simultaneous faults are common

- Defects
- Defect plus operational/automation problem
- Main line plus recovery

Problem Determination Challenges

"Systems don't break, they just stop working and we don't know why"



Characteristics of sick, but not dead

Hard for component to detect internally

Probabilistic not deterministic

- Customer view of sick, but not dead
  - 20 % of problems
  - Long duration – **generate 80% of business impact**
  - Hard to diagnose (ghost problems)
    - Every problem is unique
    - Can be triggered by any area of software or hardware
    - Occur infrequently
    - Cause sympathy sickness (creeping failures)
  - Hard to determine what actions to take to recover
- Cause of "sick, but not dead"
  - Review of significant number of incidents has identified the following generic causes
    - Damaged systems
      - Recurring or recursive errors caused by software defects anywhere in the software stack
    - Serialization
      - Priority inversion
      - Classic deadlocks
      - Owner gone
    - Resource exhaustion
      - Physical resources
      - Software resources
    - Indeterminate or unexpected states

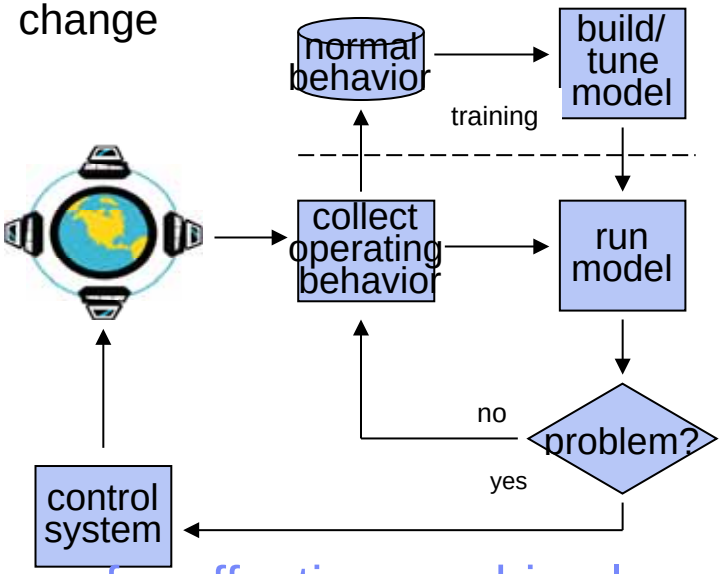
Problem Detection and Determination Procedures

Traditional Practice

- Near-zero resources permitted
  - Reasonable when computing cost >>>people cost
- Instrumentation doesn't help PD
  - Real time behavior statistics used only by 'high priesthood'
- Underlying control principle is a priori understanding of behavior
  - Tuning assumes known set of meters and knobs
- Dependent on thresholds and alarms for problem notification
  - Must be watching the right ones and set them accurately
  - Correlation can be very hard

Machine Learning

- Observe from hypervisor
- Maintenance partition
  - Now makes sense economically
- Observes low-level instrumetation
  - Models obtained from historical data analysis
- Induces meters and knobs
- Models learn what's normal and adapt to change



Virtualization provides out-of-band platform for effective machine learning without agents or modifications to VMs

Why a Changing Role For Virtualization?

- Intersection of trends
  - √ Rapid adoption of virtualization
  - √ growing customer need for availability, and
  - √ soaring management costs

- Investigation of virtualization as management tool to provide customer benefit
  - Availability is an excellent exemplar but applicability to other domains: security, performance and tuning, power, etc.

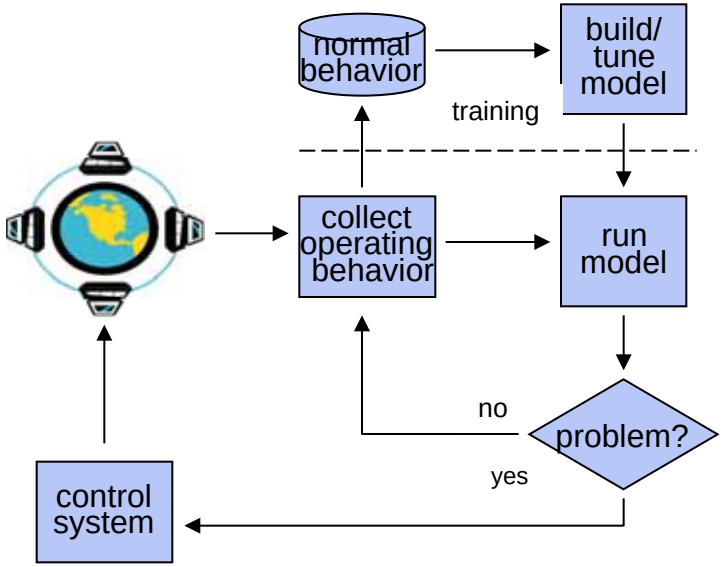
- Availability management has notable benefits
  - Working “outside the OS” generalizes availability management
    - Can manage the availability of VMs, regardless of what is in them
  - A small set of availability management functions can be defined and implemented at the VM level that may satisfy a broad swath of customers
    - Can address management complexity concerns
  - Dedicated “management partition” offers a “place to stand”, i.e. a way of providing availability (and other) functionality while meeting deployment constraints

Virtualization and Machine Learning

Predict and identify system resilience state

- Eliminates downtime whenever possible
- Simplifies complex problem determination

- **Take full advantage of live migration to avoid downtime**
- **Signal/respond to drift from normal**
- **Pinpoint faults to aid problem determination**
- **Fingerprint workload for customized actions**



Example #1

Vigilant: Out-of-band PD for Virtual Machines

Goal: general framework for detection, based on the observed behaviour of the guests

- Application failures
- Guest OS failures
- Configuration bugs
- Resource starvation
- Hardware failures

General concept for identifying component problems

- Build a model of normal machine operation using
  - Data from similar machines, or
  - Data from the same machine over a start-up period
- At runtime, classify each reading into one of the model's states
  - Normal/abnormal
  - Functioning/faulty

General concept for identifying component problems

- If the classified state is problematic, take corrective actions:
  - More resources
  - Migration
  - Save
  - Kill
- If corrective action did not solve the problem, escalate to the next corrective action
  - More complex policies possible

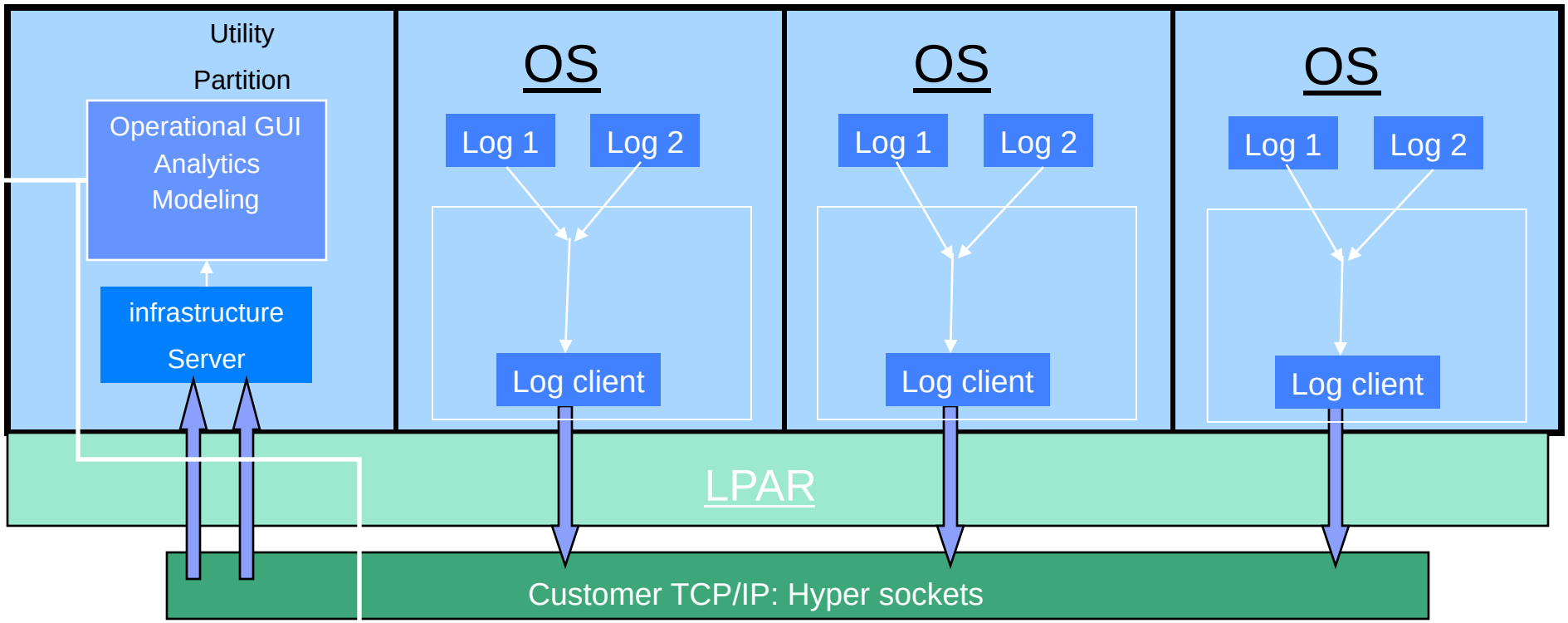
Out-of-band monitoring

- Observe the stream of requests and responses through the hypervisor API:
  - CPU utilization
  - I/O
  - Network traffic
  - Memory usage
  - Swap activity
- Periodically sample to obtain measurement vectors
- Apply machine learning to vectors

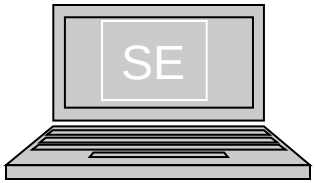
Example #2

Melody: A maintenance partition running machine learning applied to log messages to determine those most likely to indicate a problem.

How does a utility partition work?



Web Server - DOJO XML



Utility partition

- "Agent less" uses existing data from operating system
- Completely managed from the operational GUI
- Results provided by web server for humans and external managers
- Operating system data transported over TCPIP

Melody

- System logs are a major tool for diagnosing problems

  However:

- Millions of text lines per day
  - 250MB – 1.6GB of text
  - Manual sifting – impossible
  - Search is difficult when you don't know what to look for
  - Most log messages are not self explanatory
- System administrators are often not familiar with all components
- Simply recognizing that something is going wrong can be hard

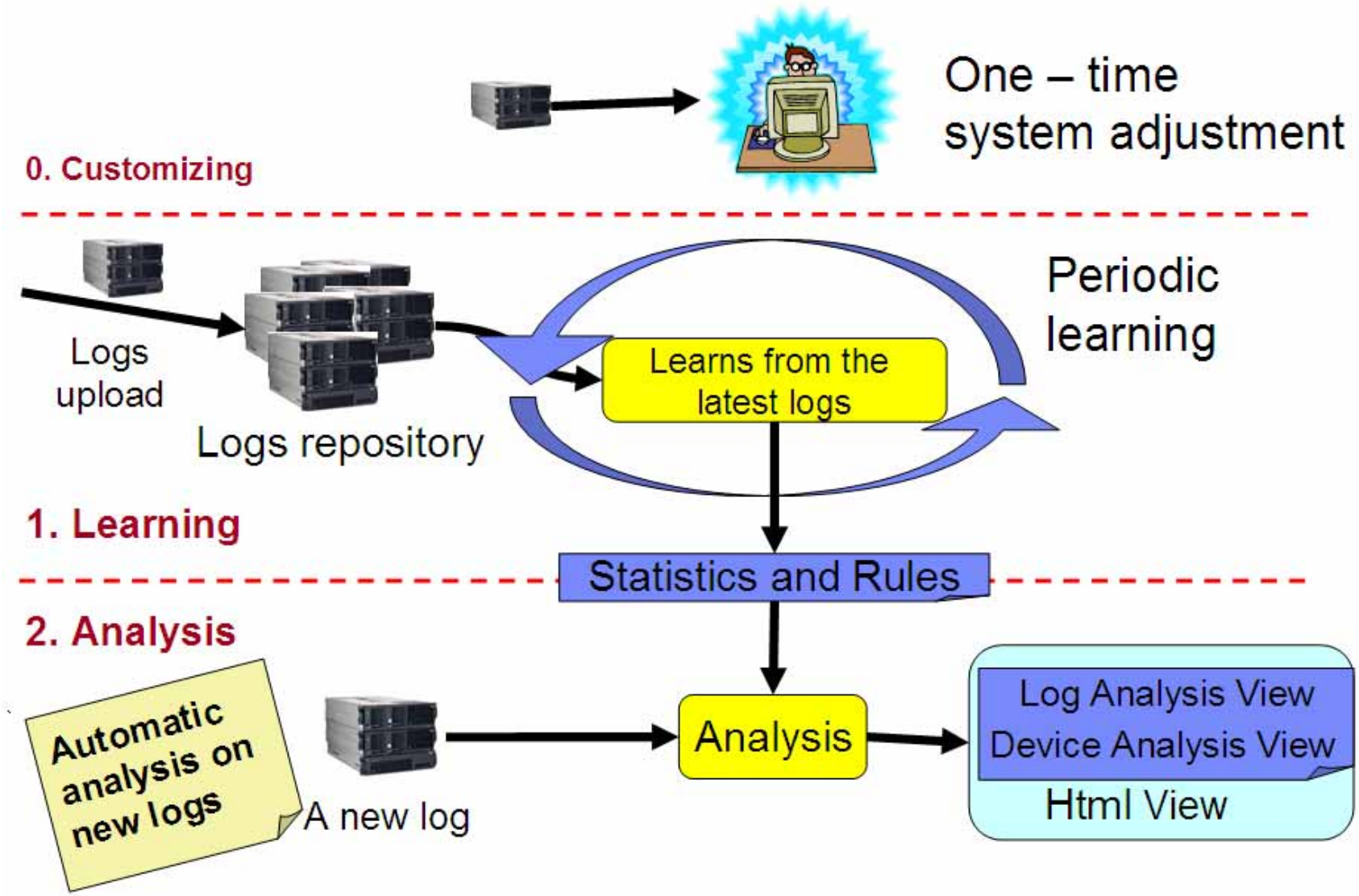
Melody

➔ Need a tool that will

- Point at potential problems
- Make the log more human-readable

- Some challenges
  - No labeled examples: requires unsupervised machine learning
  - Minimal use of domain experts
  - Should complement existing rule-based solutions

Machine Learning Paradigm



Melody

Pre Process

- Collect a training set of (mostly problem-free) logs
- Split logs into manageable parts (time frames)

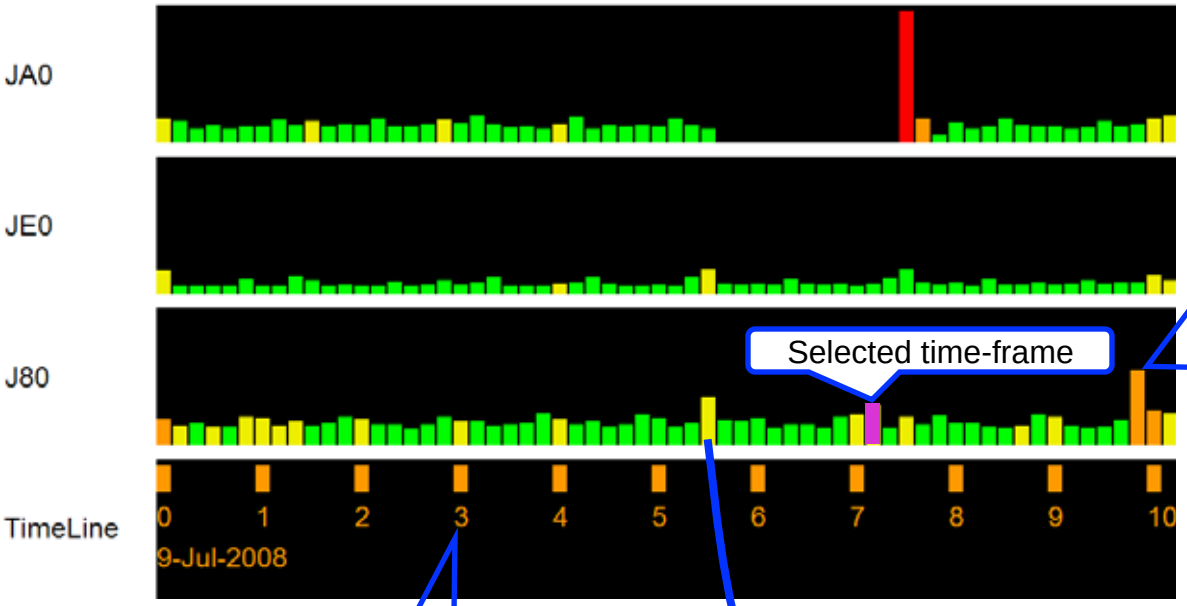
Training:

- Use *unsupervised* machine learning to model normal activity
  - Statistics of message appearances
  - Identify message appearance patterns (clustering)
  - Model periodic activity

Analysis:

- Point out anomalous time frames
- Rank messages within time frames by their anomaly
- Present the intervals in human-readable form

Melody



VM name

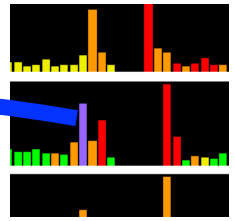
Bar **height**: number of unique message IDs

Hour of day

Bar **color**: **rarity**

Score	Meaning
101	Very rare
95-100	Rare
80-94	A bit unusual
0-79	normal

Melody



Score	Meaning
101	Very rare
95-100	Quite rare, or appears much more than usual
80-94	Appears more than usual
0-79	Appears as normally expected

Message Ids

Select sort method, by: score, component-id

18:00:00/9-Jul-2008 -- 18:10:00/9-Jul-2008

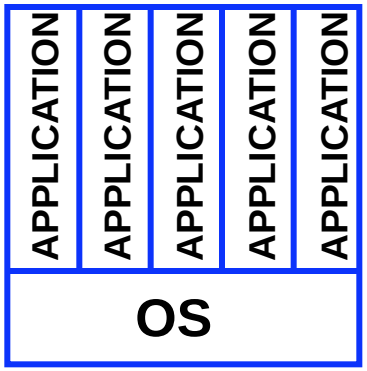
Score	#Appear	Time Line	Component	msg-id	Message
101	1		CICS Transaction Server	DFHTD0001	CICS3TBA AN ABEND (CODE 0C4/AKEX) HAS OCCURRED AT OFFSET X 0A82 IN MODULE DFHTDOC.
94	1			DFHTM1780	CICS3TBA ABEND HAS OCCURRED WHILE PROCESSING PROGRAM DFH0STAT DURING TERMINATION, CODE=APSW.
85	1		MQ Series	CSQX218E	!MQJB0 CSQXLSTT LISTENER NOT STARTED - UNABLE TO BIND, 982 1614 ADDRESS ::, TRPTYPE=TCP INDISP=QMGR, RC=0000045B
68	1		Performance Monitor	GPM066I	RMF DISTRIBUTED DATA SERVER HAS TERMINATED
68	1		--unknown--	HWSL0103I	CLEANUP SUCCESSFUL: CLIENT=WSP10688
65	1		Websphere Application Server for z/OS	BBOJ0099E	MDB PROBLEM: INTERNAL STOP ISSUED FOR MDB LISTENER PORT: 434 DESTINATION: JMS/WP1TRB20STREAMERTOPIC FOR SERVER: WP1CELL/WP1JB2/WP1JB20C/WP1JB20
65	3		MQ Series	CSQ3106E	!MQJB0 CSQ3EC0X - QUEUE MANAGER STOPPED. COMMAND NOT PROCESSED - *
64	1		CICS Transaction Server	DFHTM1752	CICS2FBA PLT - PROGRAM KOCOME00 NOT AVAILABLE.
63	2		CICS Transaction Gateway	CTG6490I	* NORMAL SHUTDOWN OF GATEWAY DAEMON STARTED BY Z/OS OPERATOR
63	2			CTG8239I	RESPONSE RECEIVED FROM CICS TRANSACTION GATEWAY *
63	2			CTG8235I	NORMAL SHUTDOWN WAS REQUESTED

- Sorting options:
  rarity, critical words, component, message ID, message clusters

Example #3

Transplay: Record and replay of an application container

Used to debug difficult software bugs

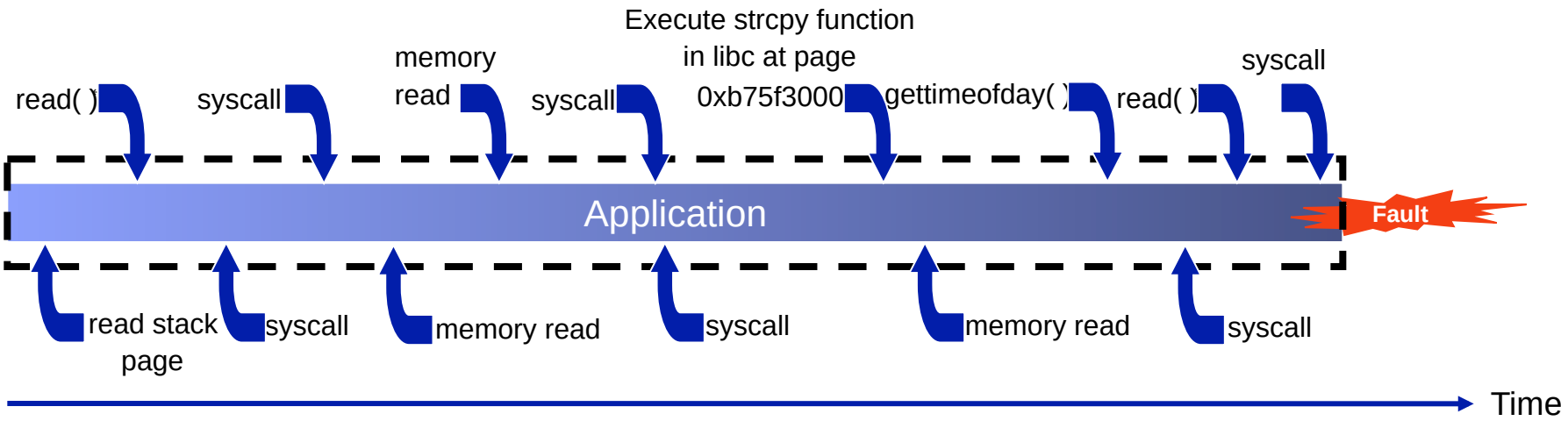


Record and Transplay

- Transplay is an integrated tool that
  - ***Efficiently*** records hard-to-reproduce bugs as they occur in production
  - Replays the same execution in a completely ***different environment***, potentially running a ***different operating system***
- Transplay prototype is able to record bugs in Linux applications and replay them on Microsoft Windows
- No application modifications needed

Application Recording

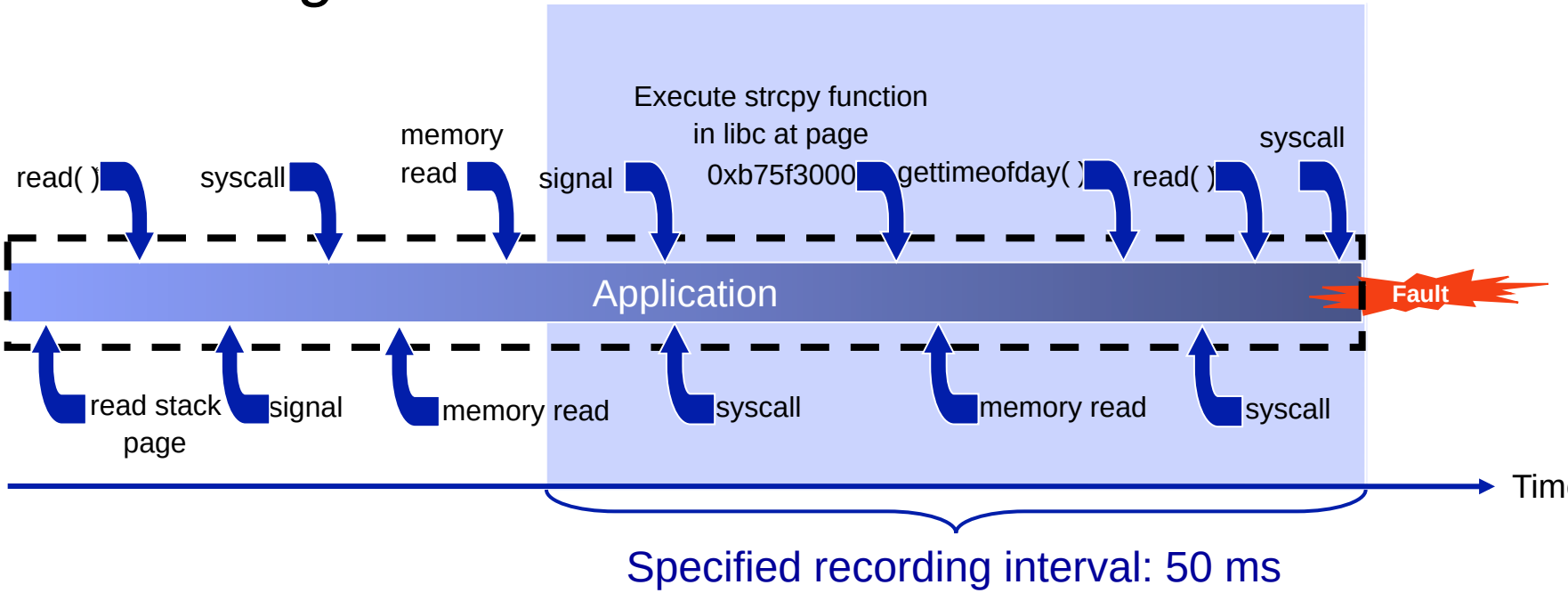
- Application is treated as a black-box and all “inputs” that enter the application are recorded



- Inputs include
  - Data read from files, network sockets etc.
  - Parts of application binaries accessed by the application
  - Data returned by the OS via system calls
  - Memory pages read by the application

Partial Checkpointing

- **Checkpoint**: *Complete intermediate state of a running application at a point of its execution*
- **Partial checkpoint**: *Partial state of an application necessary to replay its execution for a specified recording interval*



Partial Checkpointing (cont)

- `start` and `stop` primitives control partial checkpointing
  - Arbitrary periods of application's execution can be recorded for later reexecution
  - Start**:** Record the processor context, start monitoring the application
  - Stop**:** Save the accumulated log data

- A partial checkpoint consists of:
  - Processor context at the beginning of the recording interval
  - Memory pages accessed by the application
  - Results of system calls made by the application
  - Meta data necessary for deterministic reexecution: interleaved shared memory accesses, signals

Replay Across Operating Systems

- Partial checkpoint is self-contained
  - All data needed by the application comes from the log; hence underlying operating system doesn't matter
- Application's interface with the underlying OS is virtualized
  - Enables an unmodified Linux binary to run on Microsoft Windows
- Use binary instrumentation (Pin) to transparently
  - Intercept Linux system calls on Windows
  - Resolve memory conflicts by transposing memory references

Replay

- Application’s address space partially reconstructed
  - Large portions of application's address space left empty before transferring control to the application
- Control transferred to the application by loading the processor state
- Application’s requests are satisfied by replaying the precomputed results observed during recording