

# An Active Approach to Characterizing Dynamic Dependencies for Problem Determination

**Aaron Brown**

Computer Science Division  
University of California at Berkeley

**Gautam Kar, Alexander Keller**

IBM T.J. Watson Research Center

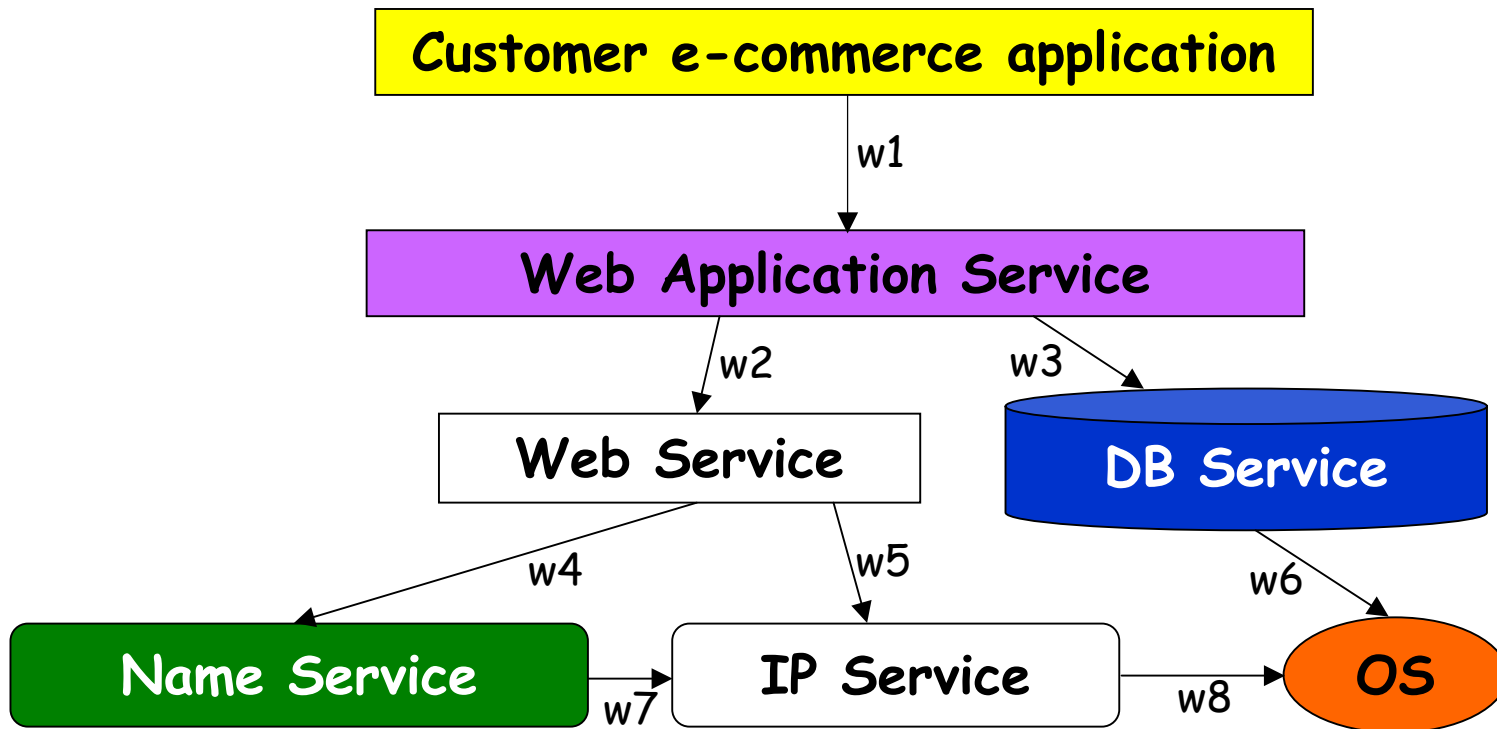
IM 2001, 16 May 2001

# Motivation: problem diagnosis

- Troubleshooting problems is one of the most challenging, time-consuming management tasks
  - symptoms are typically at end-user or SLA level
  - root causes are typically much deeper in system
    - » and often confounded by system complexity
  - must map symptoms to root causes to locate problems!
- *Dependency models* provide an invaluable aid to root-cause analysis
  - capture connections between high- and low-level system components

# Dependency models in a nutshell

- Use a graph (DAG) structure to capture dependencies between system components
  - if failure of **A** affects **B**, then **B** depends on **A**
  - edge weights represent dependency strengths



# Constructing dependency models

- For effective diagnosis, model must capture:
  - static dependencies
  - dynamic runtime dependencies
    - » *e.g.*, dependencies induced by runtime queries
  - distributed dependencies
  - dependency strengths
  - all at the detailed level of individual system resources
- Most existing techniques don't meet these challenges...

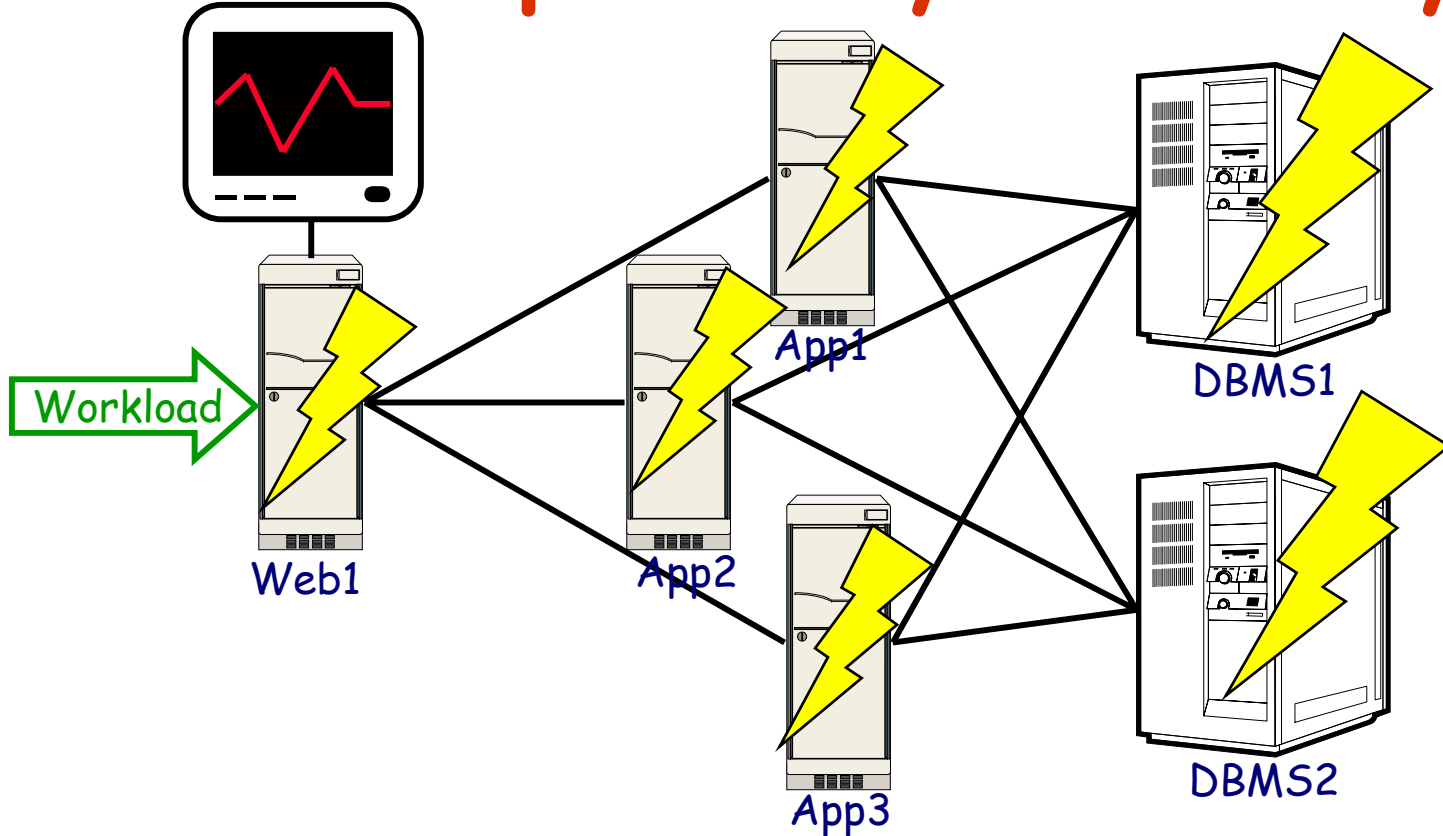
# Outline

- Motivation & background
- **ADD: Active Dependency Discovery**
- Experimental validation of ADD
- Conclusions and future directions

# Discovering dependencies

- **Desired properties of approach**
  - identifies dynamic, runtime dependencies
  - works on distributed systems
  - works with only black-box view of system components
  - provides direct evidence of causality
  - detects dependencies only visible in failure situations
- **These properties inspire an indirect, active approach**
  - **indirect**: no explicit modeling of system
  - **active**: perturb system to elucidate dependencies

# Active Dependency Discovery (ADD)



- 1) Instrument the system and apply workload
- 2) Systematically perturb components
- 3) Measure change in system response
- 4) Construct dependency model from measurement data

# Benefits of ADD

- **Coverage**

- no need to rely on problems occurring naturally, as in passive approaches
- can guarantee coverage by explicitly controlling perturbation

- **Causality**

- causality easy to establish: perturbation is the cause

- **Simplicity**

- no application modeling or modification necessary
- existing endpoint instrumentation may be sufficient
- no complex data mining required
- applied *before* real problems occur



# Drawbacks of ADD

- **Invasiveness**

- can be tricky to do perturbation on production system
- possible solutions:
  - » leverage redundancy if available (*e.g.*, cluster system)
  - » run perturbation during non-production periods (initial system setup or during scheduled downtime)
  - » develop low-grade perturbation techniques

- **Workload-specific**

- extracted models only valid for applied workload
- but, can model components of workload and recombine later

# Outline

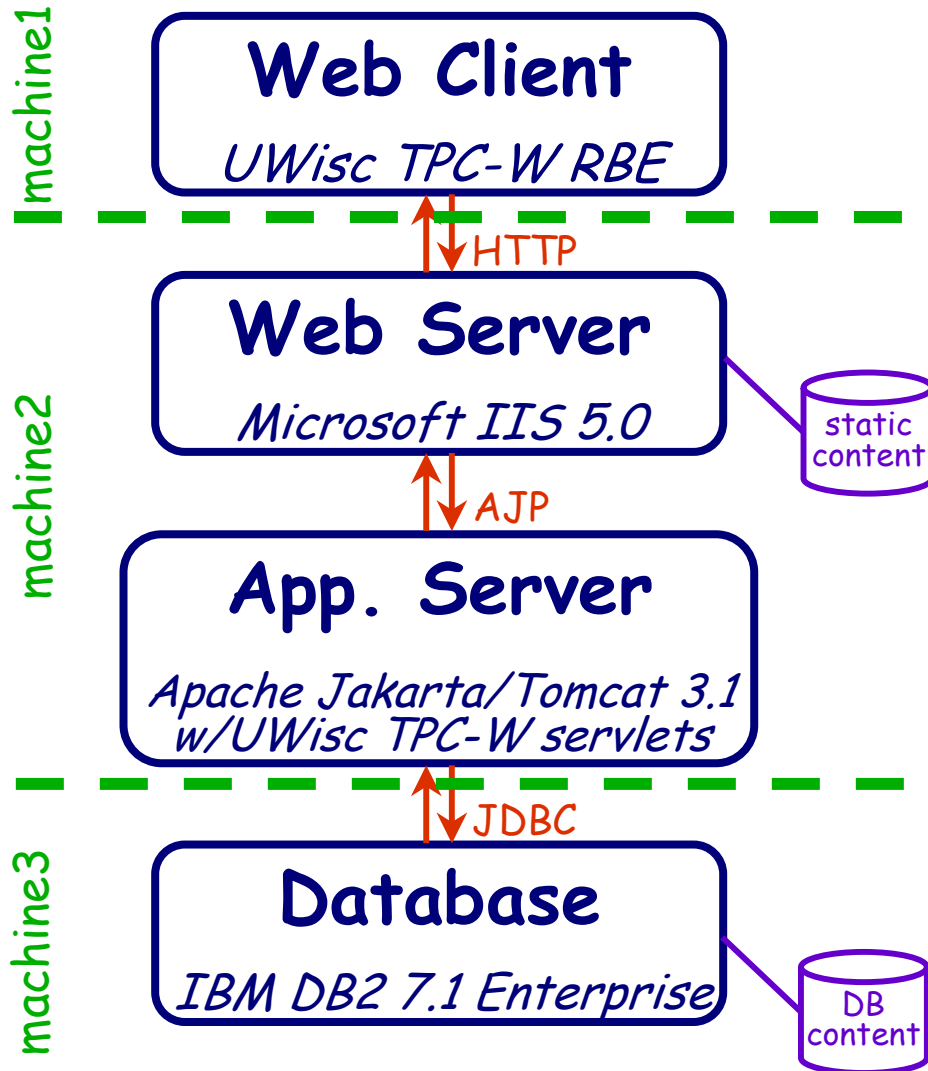
- Motivation & background
- ADD: Active Dependency Discovery
- **Experimental validation of ADD**
  - approach
  - TPC-W testbed environment
  - results
- Conclusions and future directions

# Validation: e-commerce case study

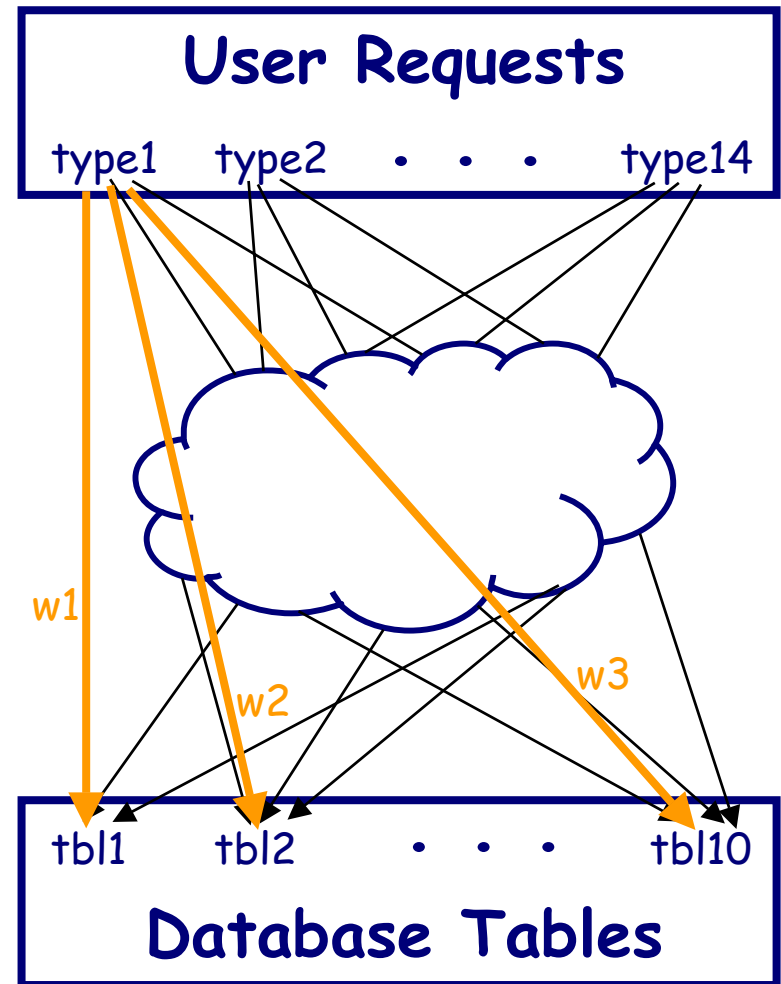
- **Goal: use ADD to discover dependencies in a multi-tier e-commerce environment**
  - using off-the-shelf black-box software
  - in a realistic environment with realistic workload
- **Task: discover dependencies of user web requests on database tables**
  - for each type of user request:
    - » extract dependencies on individual database tables
    - » characterize strengths of those dependencies
    - » hand-verify model against application source code
- **Platform: TPC-W benchmark app & workload**
  - realistic mockup of online bookseller e-commerce site

# TPC-W experimental testbed

## System View



## Dependency View

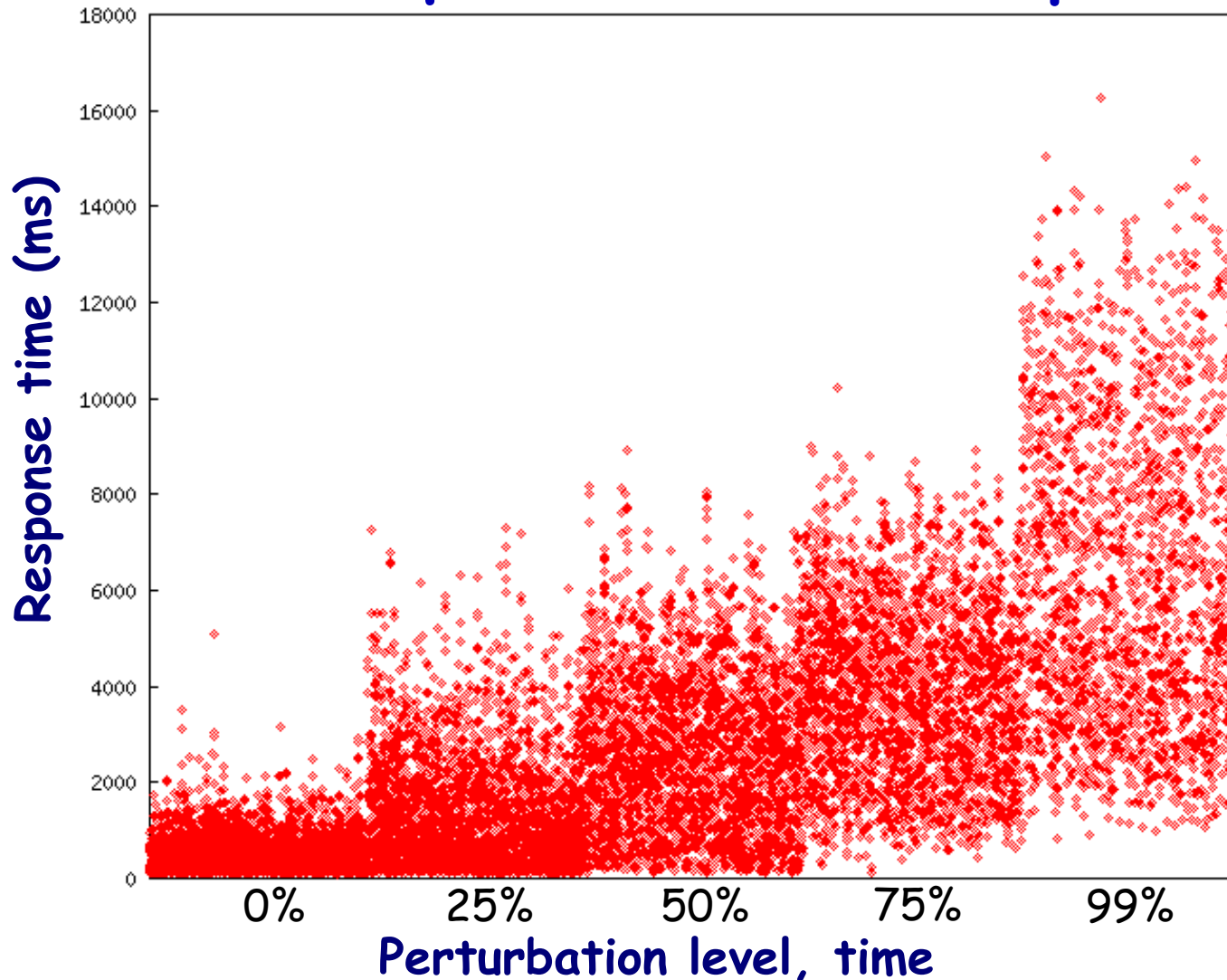


# Perturbation and measurement

- **Perturbation applied to individual DB tables**
  - use DB2's lock manager to exclusive-lock a table
  - configurable "duty cycle" of lock out
    - » queries locked out for first x% of every 4 sec. interval
  - only affects one table; no impact on overall load
  - can simultaneously perturb multiple tables
- **Per-request response time measured by TPC-W front-end user emulator**
  - 14 different types/classes of requests
  - response time is end-to-end, including network delay

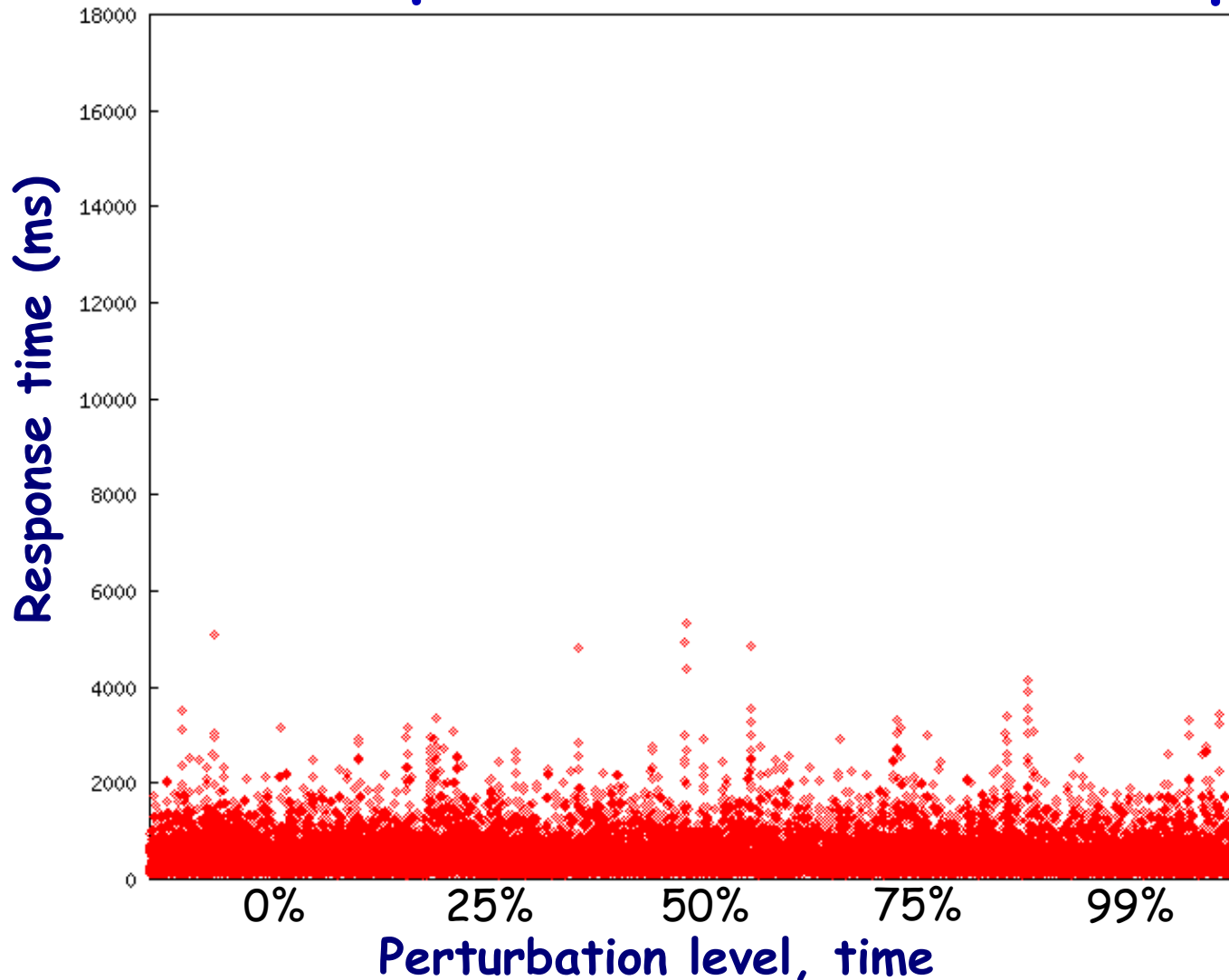
# Raw perturbation results

- Ex: Search request, ITEM table perturbed



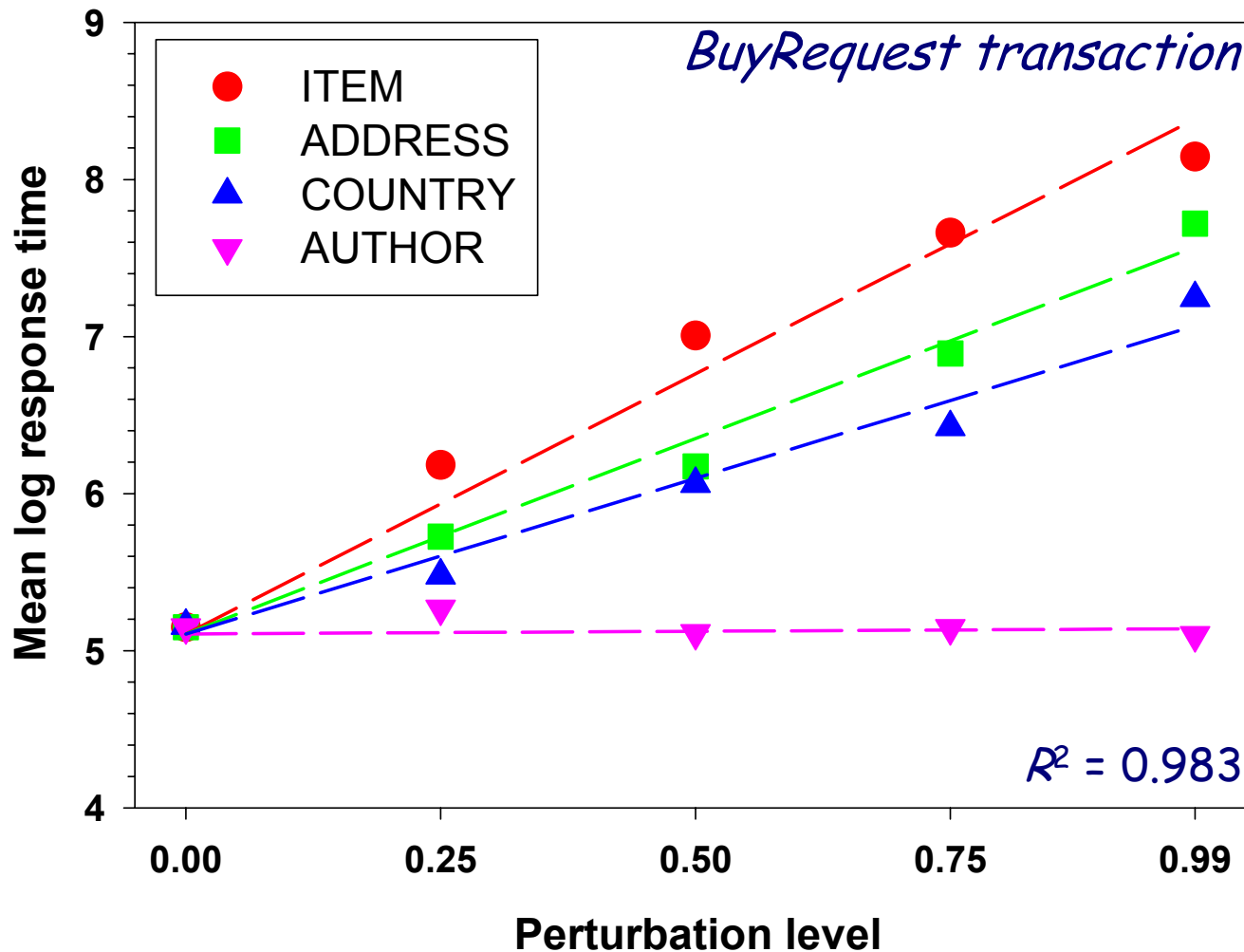
# Raw perturbation results (2)

- Ex: Search request, `CC_XACTS` table perturbed



# Applying a linear model

- Linear regression on mean of log of data
  - statistically positive slope gives dependency strength





# Summary of results

- **Modeling correctly identified 41 of 42 true dependencies at 95% confidence level**
  - compare to 140 potential dependencies (!)
  - one false negative most likely due to insufficient data
  - caveat: some glitches due to unmodeled interactions
    - » manifested as small negative dependency strengths
    - » solution: improve model or simply discard negative strengths

# Summary of results (2)

- Tabular representation of full dependency set:

Request Table	admncnf	admreq	bestsell	buyconf	buyreq	custreg	home	newprod	ordrdisp	ordering	proddet	srchreq	srchres	shopcart
ADDRESS				X	X				X					
AUTHOR	X	X	X					X			X		X	
CC_XACTS				X					X					
COUNTRY				X	X				X					
CUSTOMER				X	X	X			X					
ITEM	X	X	X	X	X		X	X	X		X	X	X	X
ORDER_LINE	-		X	X					X					
ORDERS	X		X	X					X					
SHOP_CART														X
SHOP_CART_L				X	X									X

Strengths: X = (0,1]    X = (1,2]    X = (2,3]    X = (3,4]

# Using dependencies for diagnosis

- **When a problem occurs:**
  - 1) identify faulty request
    - » from problem report, SLA violation, test requests, ...
  - 2) select the appropriate column in dependency table
  - 3) select the rows representing dependencies
    - » this is the set of potential root causes
  - 4) investigate potential root causes, starting with those of highest weight

# Using dependencies for diagnosis (2)

- **Can extend approach to multiple system levels**
  - compute one dependency matrix per level
  - iterate levels from user symptoms to culprit resource
- **This process may not uniquely identify problem**
  - but can narrow down the culprits via combinations
    - » isolating the effects of individual tables
    - » e.g., SHOP\_CART\_L "=" orderdisp - buyconf
  - not all tables can be uniquely isolated
    - » but could do so by adding synthetic test requests?
    - » ideal is to build a *basis* for the whole-system dependency matrix

# Outline

- Motivation & background
- ADD: Active Dependency Discovery
- Experimental validation of ADD
- **Conclusions and future directions**

# Conclusions and future directions

- **Dependency models help problem determination**
- **ADD effectively discovers dependency models**
  - approach is uniquely positioned in the design space
    - » active, indirect approach finds dynamic, distributed dependencies; works on black-box systems
  - initial experimental results are promising
    - » very good success on TPC-W experiments
- **Future directions**
  - techniques to integrate ADD into production systems
  - investigation of end-to-end vs. layer-by-layer tradeoffs
  - using dependency models for other management tasks
    - » impact analysis, performance optimization, ...

# An Active Approach to Characterizing Dynamic Dependencies for Problem Determination

*For more information:*

abrown@cs.berkeley.edu

{gkar,alexk}@us.ibm.com

<http://www.research.ibm.com/sysman>

End



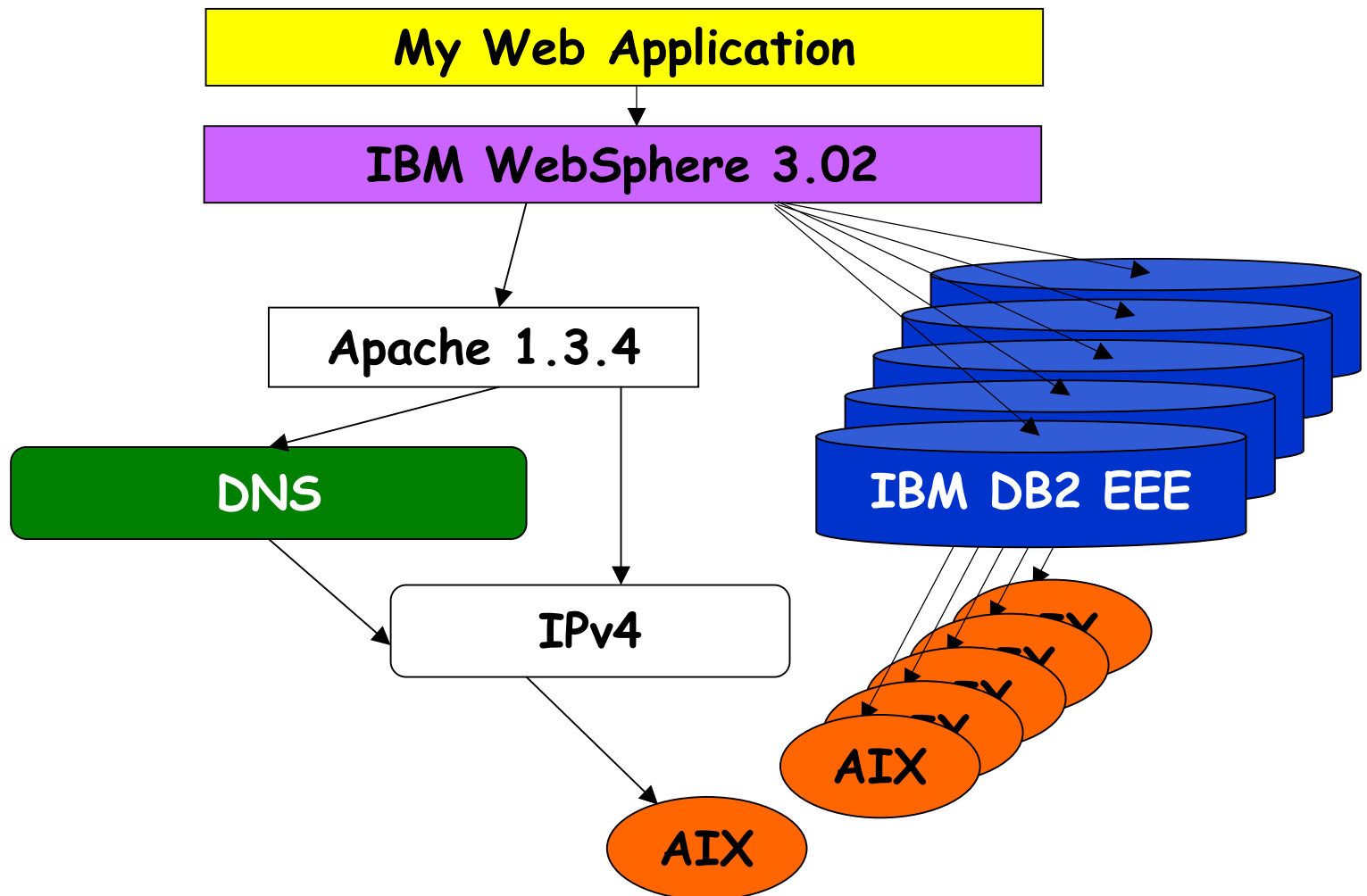
# Backup slides

# Dependencies & root-cause analysis

- There are good algorithms for root-cause analysis using dependency data
  - event correlation [Yemini96, Choi99, Gruschke98, ...]
  - systematic probing via graph-traversal [Kätker95]
- But...they assume dependencies are identified manually!
  - impractical in modern systems at any interesting level of detail
  - need automatic discovery of fine-grained dependency models to solve practical problems

# A motivating example...

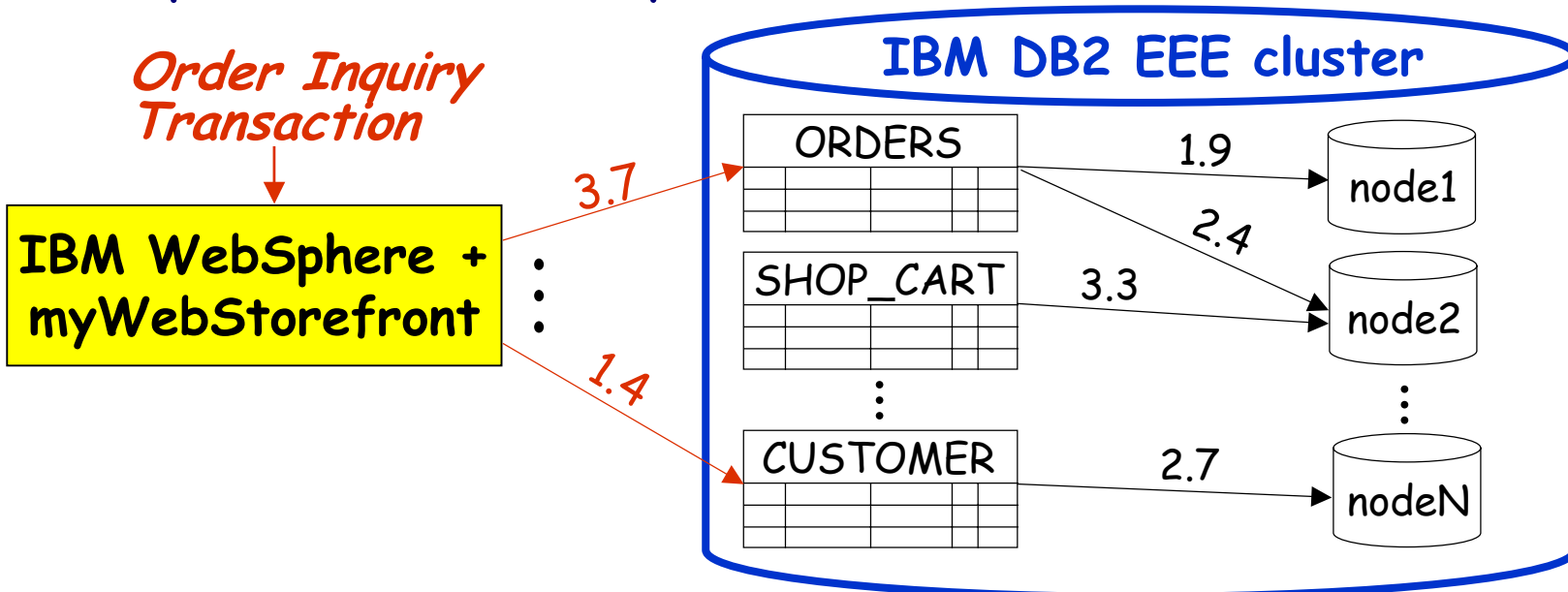
- E-commerce system with cluster database



*This level of detail is called a "structural" model*

# What's really needed?

- **Dynamic, operational dependency graphs**
  - based on runtime behavior, not static analysis
  - computed for each type of user transaction/action
    - » each transaction's graph is a subgraph of the overall system dependency graph
  - dependencies weighted by "strength" and parameterized by workload



# How is this useful?

- **Helps restrict search space for root cause of a problem**
  - presence/absence of operational dependencies tells you where you must look
  - dependency strengths may optimize search
  - in most cases, cannot completely identify root cause
- **Aids in system optimization**
  - dependency strengths reflect balance of system
- **Supports “impact analysis”**
  - strength of dependency is a direct measure of failure impact of a particular component

# Dependency discovery: approaches

- **Direct**

- relies on human to analytically compute dependencies
  - » from app-specific knowledge, configuration files, ...
- impractical for realistic systems

- **Indirect**

- based on instrumentation and monitoring
- correlates observed failures/degradations across components
- typically *passive*
  - » no perturbation to system beyond instrumentation
- examples: data mining, event correlation, neural-net dependency discovery, MPP bottleneck detection

# Challenges of an indirect approach

## 1) Causality

- most indirect approaches identify only correlation

## 2) Coverage

- passive approaches only find dependencies that are activated while the system is monitored
- can miss important dependencies that only appear in rare failure modes
  - » but these are often the most important dependencies!

## • Solution: an *active* indirect approach

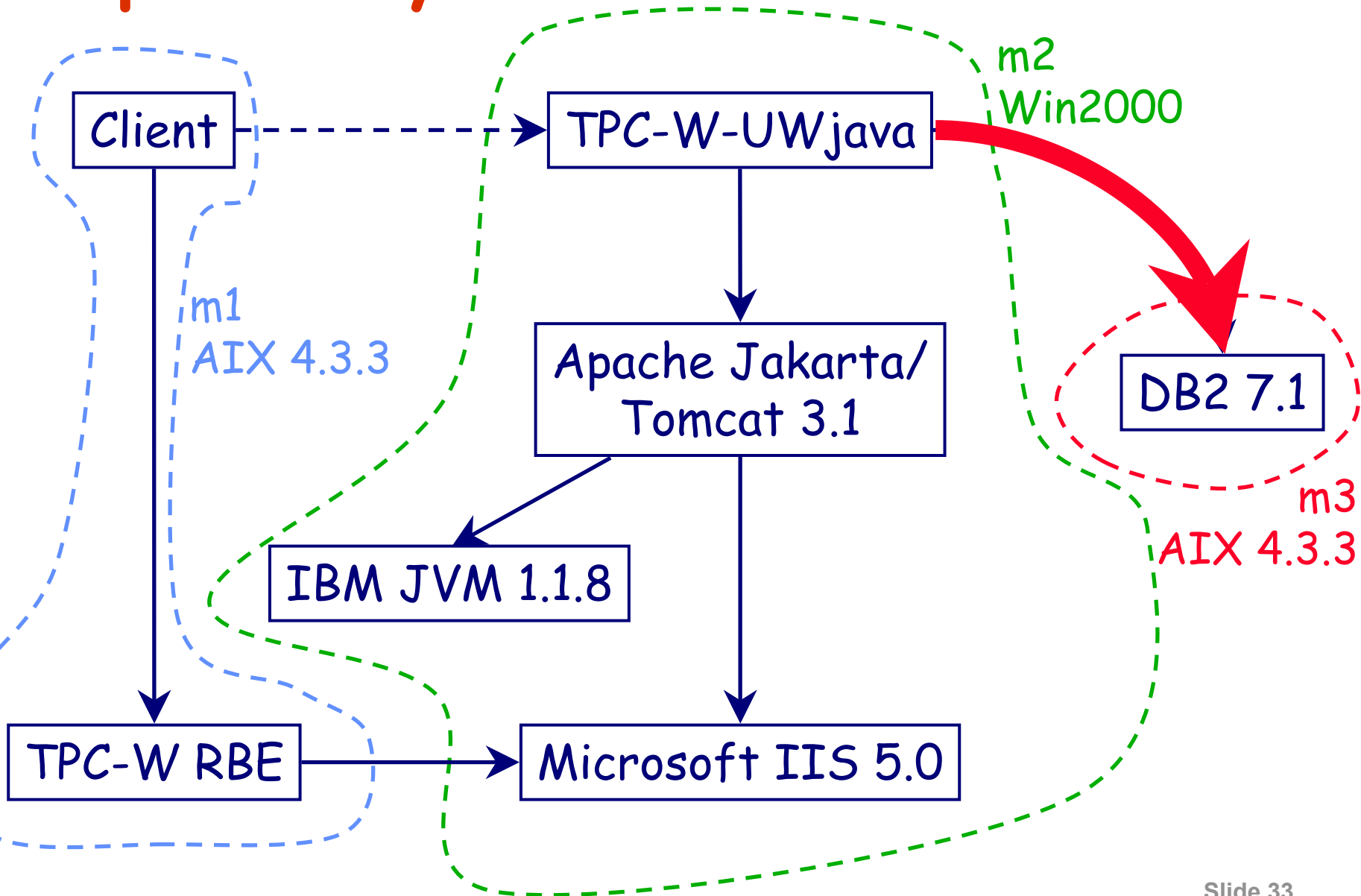
- directly perturb the system, establishing causality and increasing coverage

# Testbed web application

- **TPC-W web commerce application**
  - standardized TPC benchmark
  - simulates activities of a "business-oriented transactional web server"
  - implements storefront of an Internet book seller
  - includes user sessions, shopping carts, browsing, search, online ordering, "best sellers", ...
  - includes workload specification and generator
    - » fully parameterized
    - » standard mixes to simulate users that are mostly-browsing, mostly-ordering, or shopping (mix)
  - implementation in Java from University of Wisconsin



# Dependency view: TPC-W testbed



# Experiment details

- **Workload**

- 90 simulated users
- TPC-W standard "shopping" mix
- an average of 11.8 unperturbed transactions/sec
- servers not saturated by this workload

- **Perturbation**

- only one table perturbed at a time
- 0%, 25%, 50%, 75%, 99% levels for each table
- 30 minutes of perturbation at each level

# Limitations of the test case

- **Constant workload**
  - can't parameterize dependencies by workload
- **Independent table perturbation**
  - can't include interaction terms in model
- **End-to-end performance metric**
  - OK here since we're only looking at one level of system
  - assumes perturbations don't have additional effects beyond the database
  - if the dependency is not manifested in performance, it won't be detected
- *None of these limitations are inherent*

# Modeling details

- **Simple first-order linear model:**

- assumes constant effects, independence, and linearity of perturbation (under transform of  $m$ )
- let  $m_i$  be some metric for transaction type  $i$
- let  $\mu_i$  be the mean non-perturbed value of  $m_i$
- let  $P_j$  be the level of perturbation of system element  $j$
- then:

$$r_i = \mu_i + \sum_j (\alpha_j P_j) + \varepsilon$$

- the  $\alpha_j$ 's are fit to the data, and represent the effects of perturbation of the components  $j$ 
  - »  $\alpha_j$  characterizes the strength of  $m_i$ 's dependency on  $j$

# Model details

- Fit a first-order linear model:

$$r_i = \mu_i + \sum_j (\alpha_j P_j) + \varepsilon$$

- Estimated effects ( $\alpha_j$ ) for buy request txn:

ITEM:	3.31 ± .26	SHOP_CART:	0.06 ± .26
ADDRESS:	2.49 ± .26	CC_XACTS:	0.06 ± .26
CUSTOMER:	2.41 ± .26	AUTHOR:	0.03 ± .26
SHOP_CART_LINE:	2.35 ± .26	ORDER_LINE:	0.003 ± .26
COUNTRY:	1.98 ± .26	ORDER:	-0.02 ± .26

- Despite simplicity, models fit well

- $R^2$  ranges from .906 to .996, with mean .973
- there are clearly higher-order effects present
  - » especially noticeable in significant *negative* effects
  - » but first-order effects dominate

# Existing approaches

- **Most popular approaches are passive**
  - event collection and data mining
  - neural-network-based dependency discovery
  - performance bottleneck detection in parallel programs
  - network fault detection
  - nuclear power plant problem diagnosis
- **Passive approaches have two main weaknesses:**
  - hard to differentiate *correlation* and *causation*
  - hard to get coverage of all problem/failure cases
- **Active approaches limited to postmortems**

# A less-linear result

- Not nearly as linear, but linear model still sufficient
- Example data: order confirmation transaction

