DBTaint: Cross-Application Information Flow Tracking via Databases

Benjamin Davis
Hao Chen
University of California, Davis

Introduction

- Web services are highly attractive targets
- Over 60% of attacks target Web applications
- Over 80% of vulnerabilities found are in Web applications

(From SANS 2009 Top Cyber Security Risks)

Cross-Site Scripting

<h1>Latest Comment</h1>

<p>

{User Content}

</p>

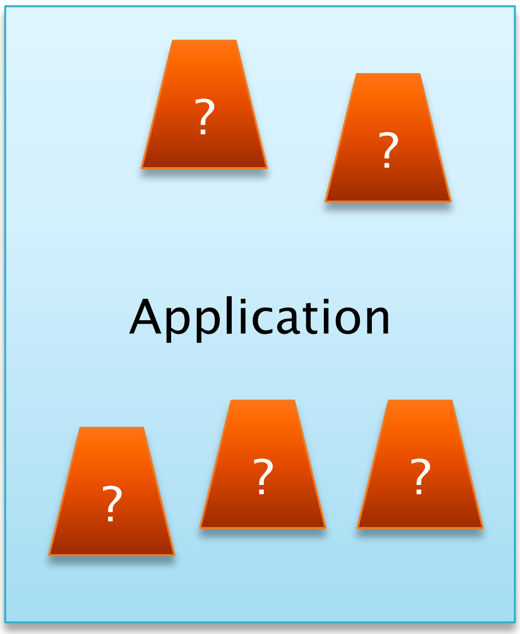
Cross-Site Scripting

<h1>Latest Comment</h1>
<p>
This is great!
</p>

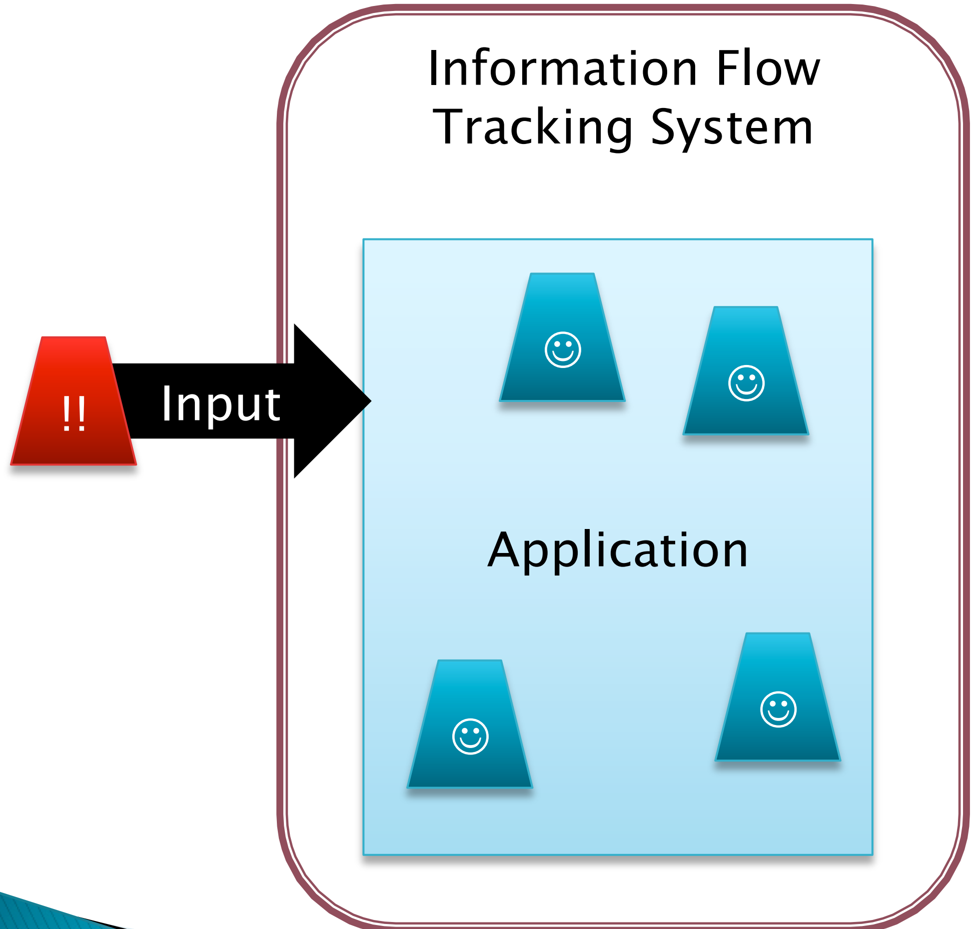
Cross-Site Scripting

```html
<h1>Latest Comment</h1>
<p>
<script>
steal(document.cookie);
</script>
</p>
```

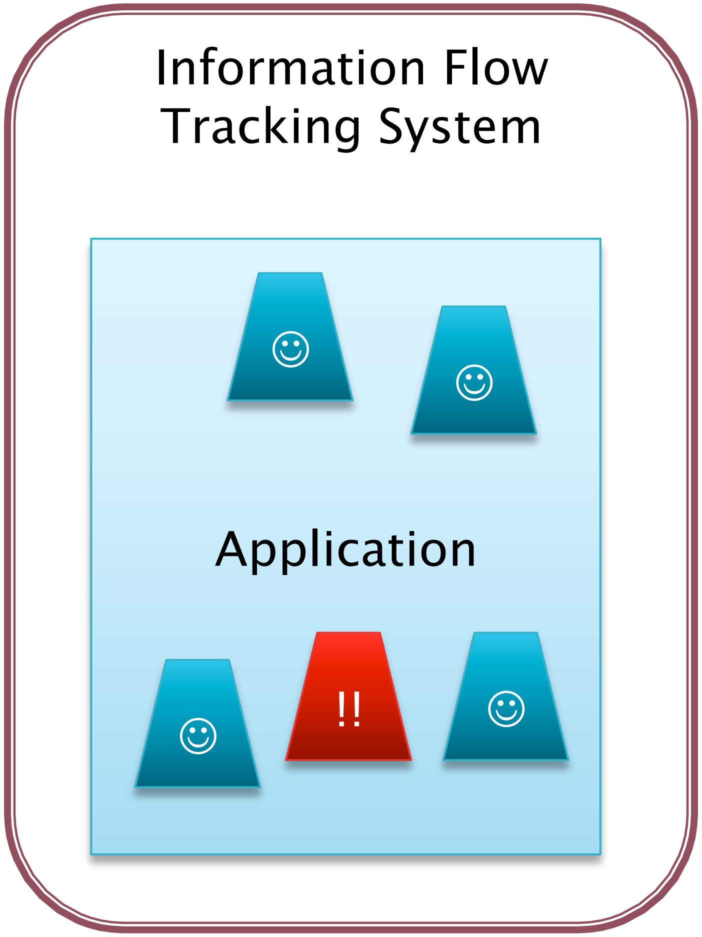
Information Flow Tracking



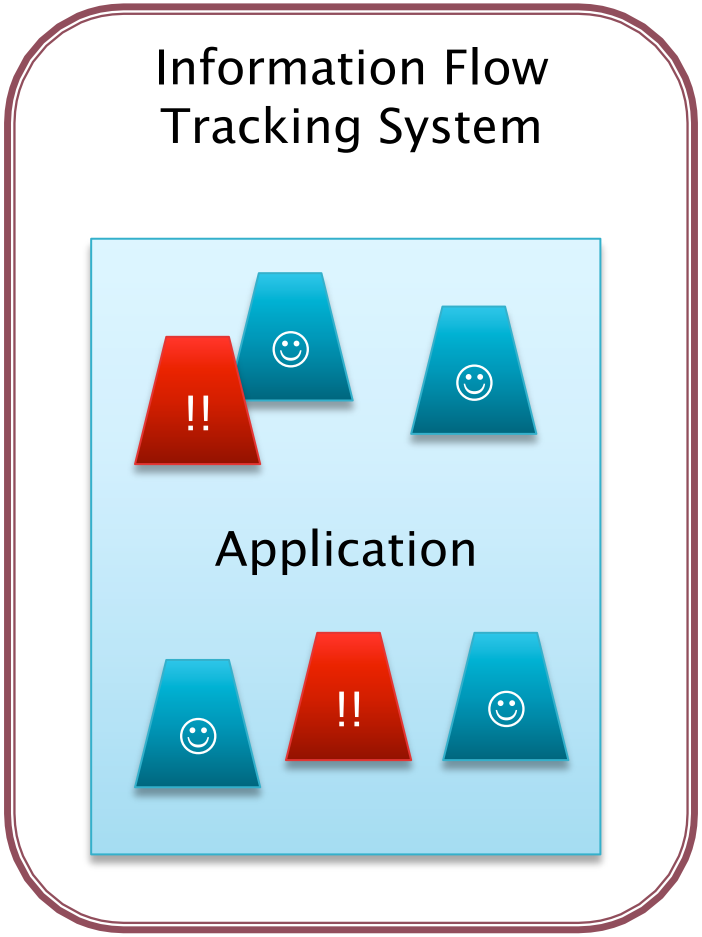
Information Flow Tracking



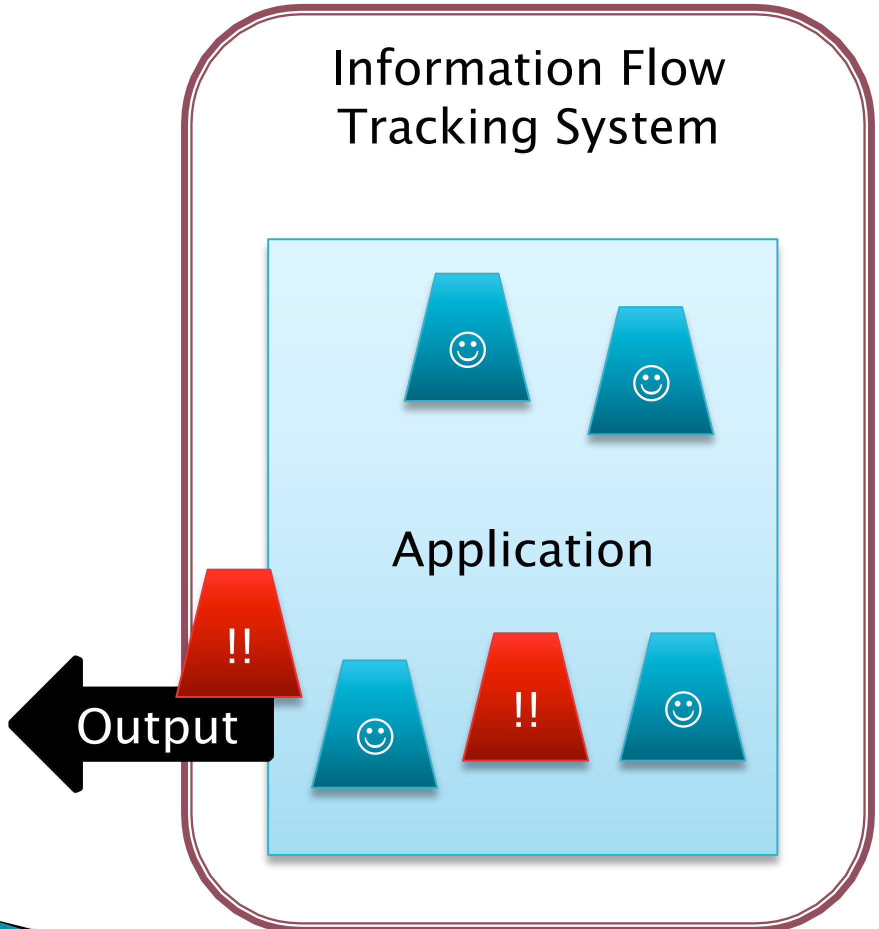
Information Flow Tracking



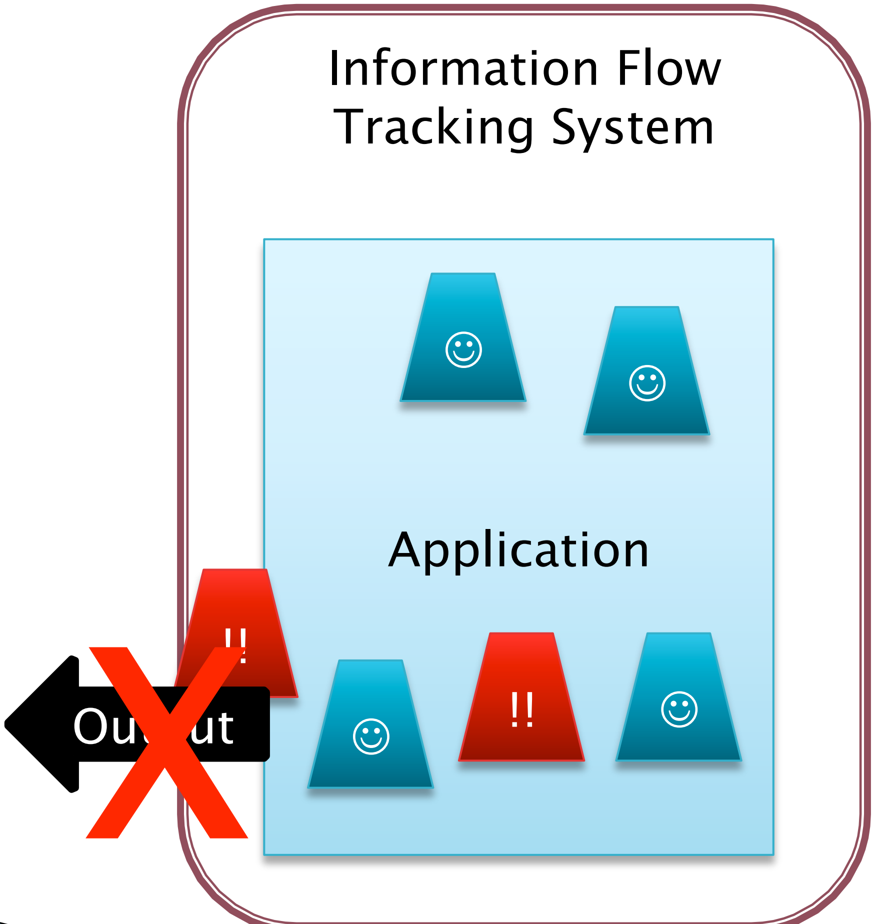
Information Flow Tracking



Information Flow Tracking



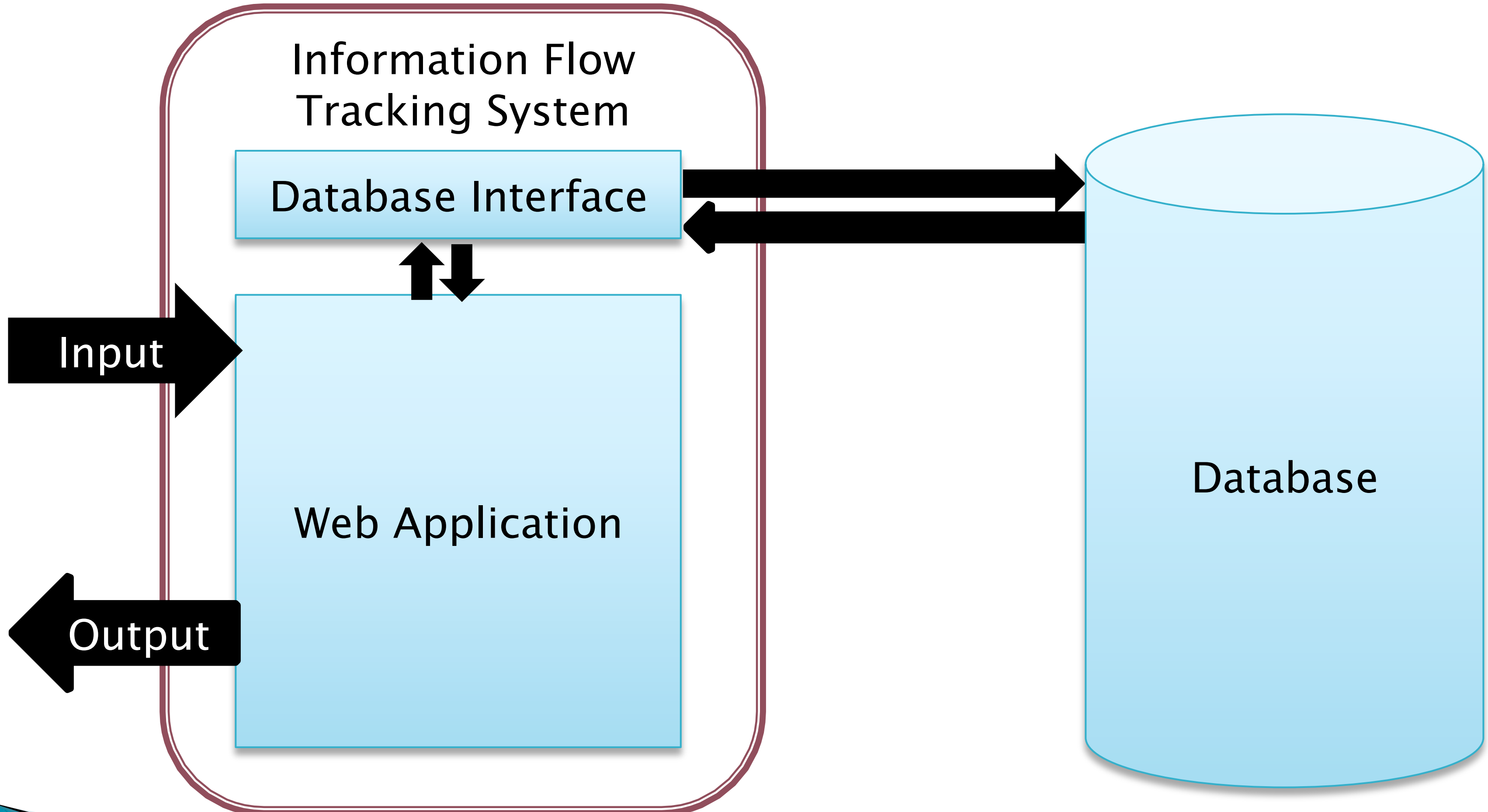
Information Flow Tracking



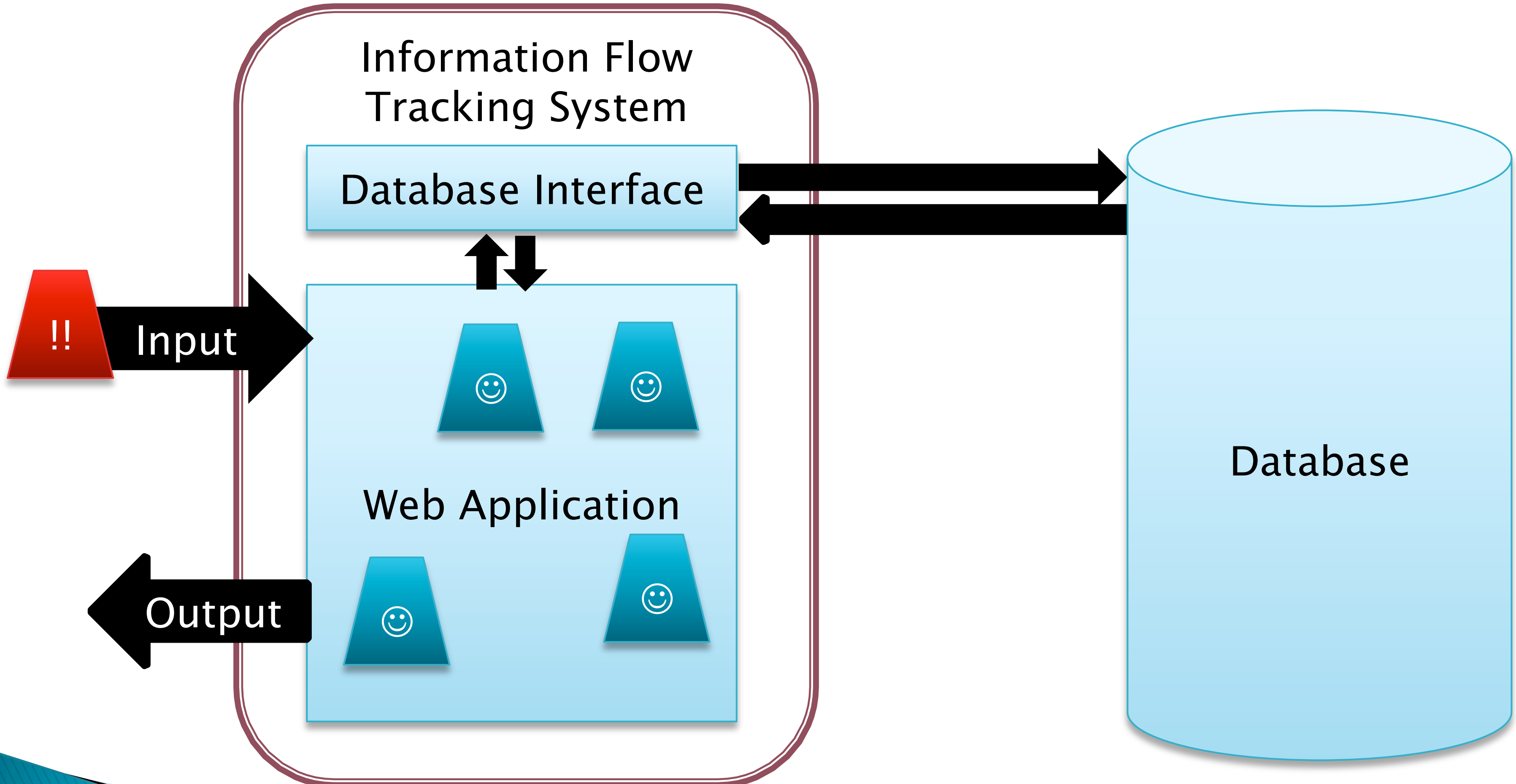
Examples

- Language-based “taint mode”
  - Perl
  - Ruby
- Adding support to language structures
  - Java [Chin, Wagner 09]
  - PHP [Venema]

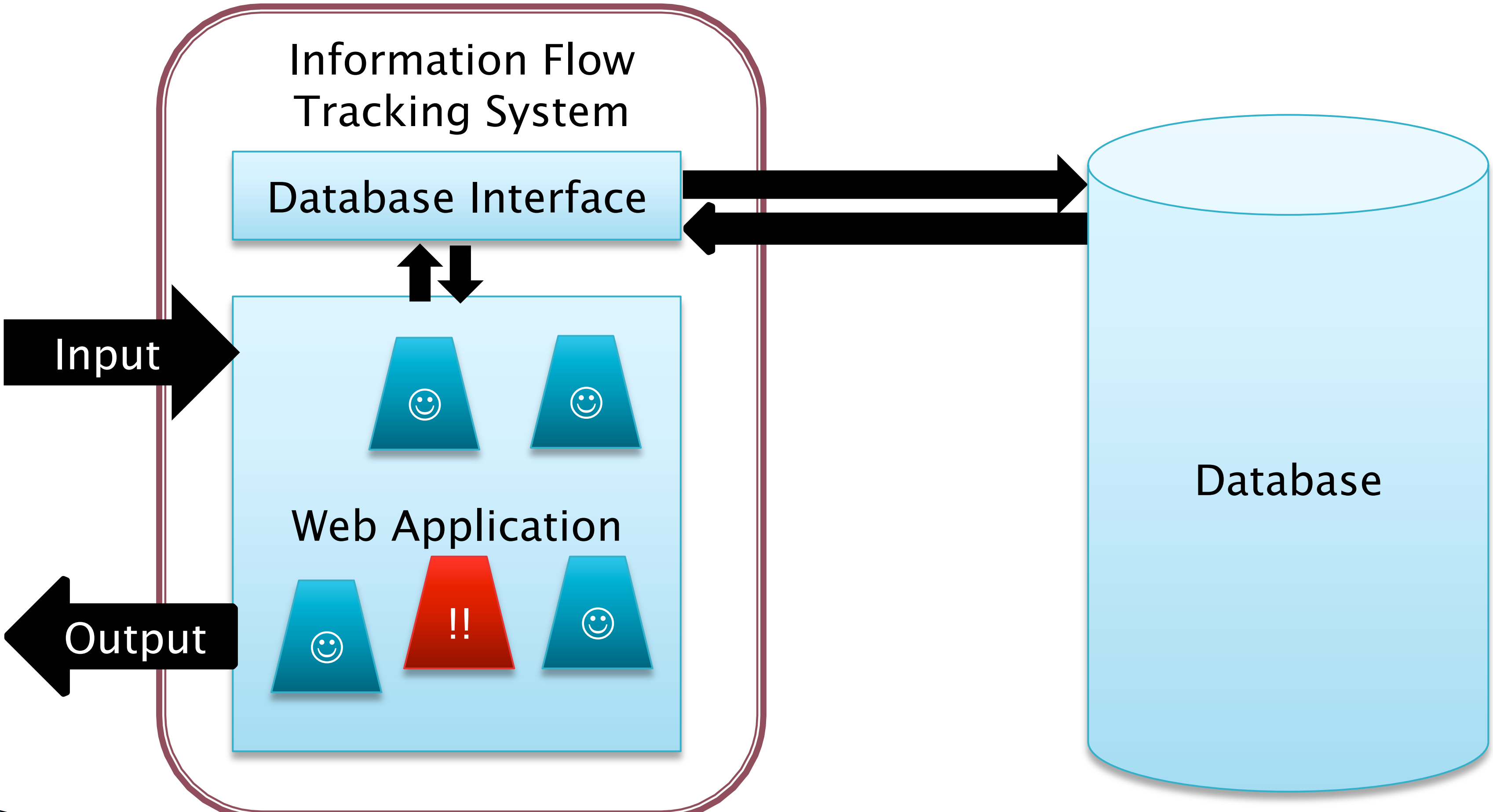
Limitations of Single-Application Systems



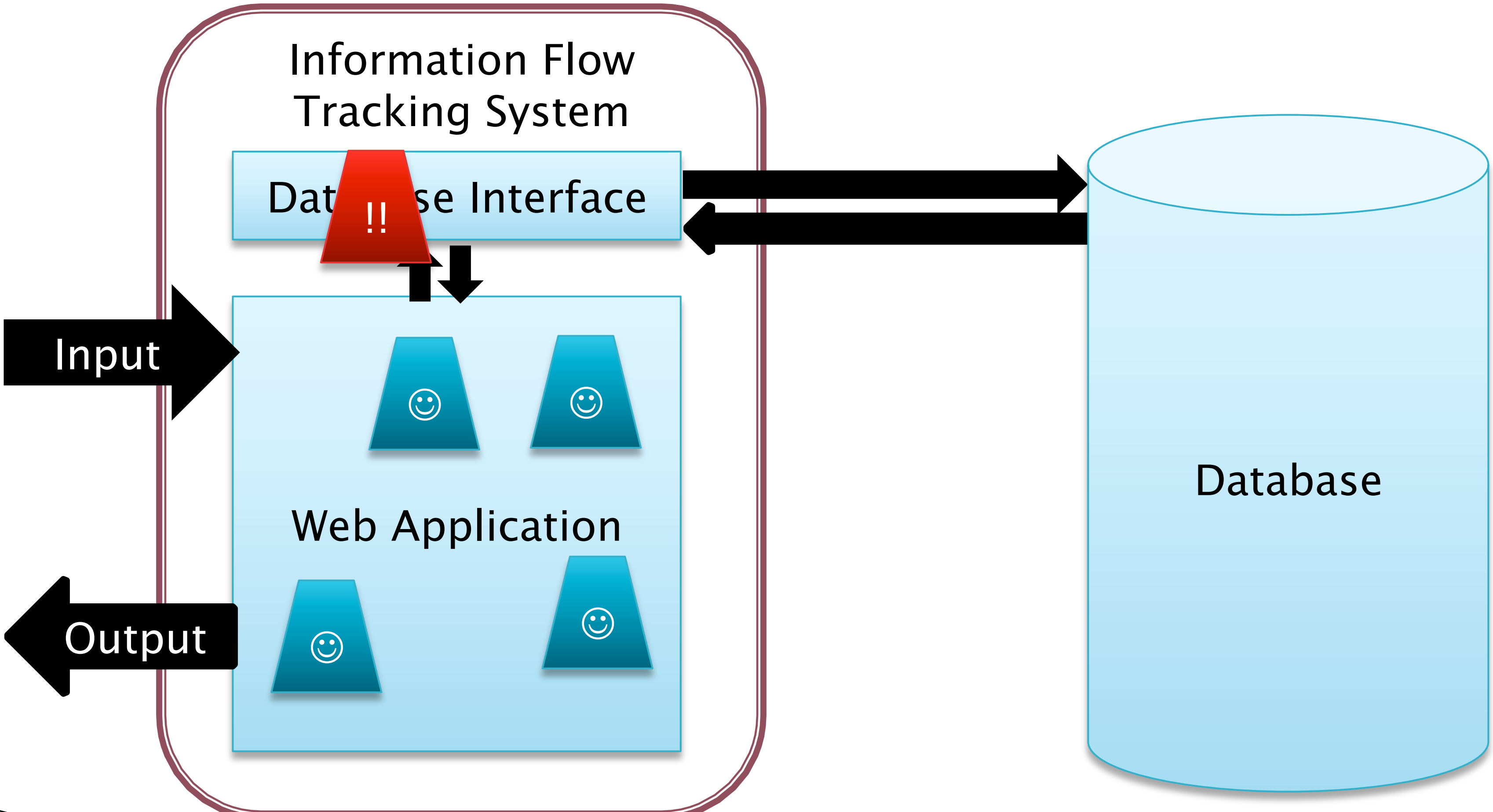
Limitations of Single-Application Systems



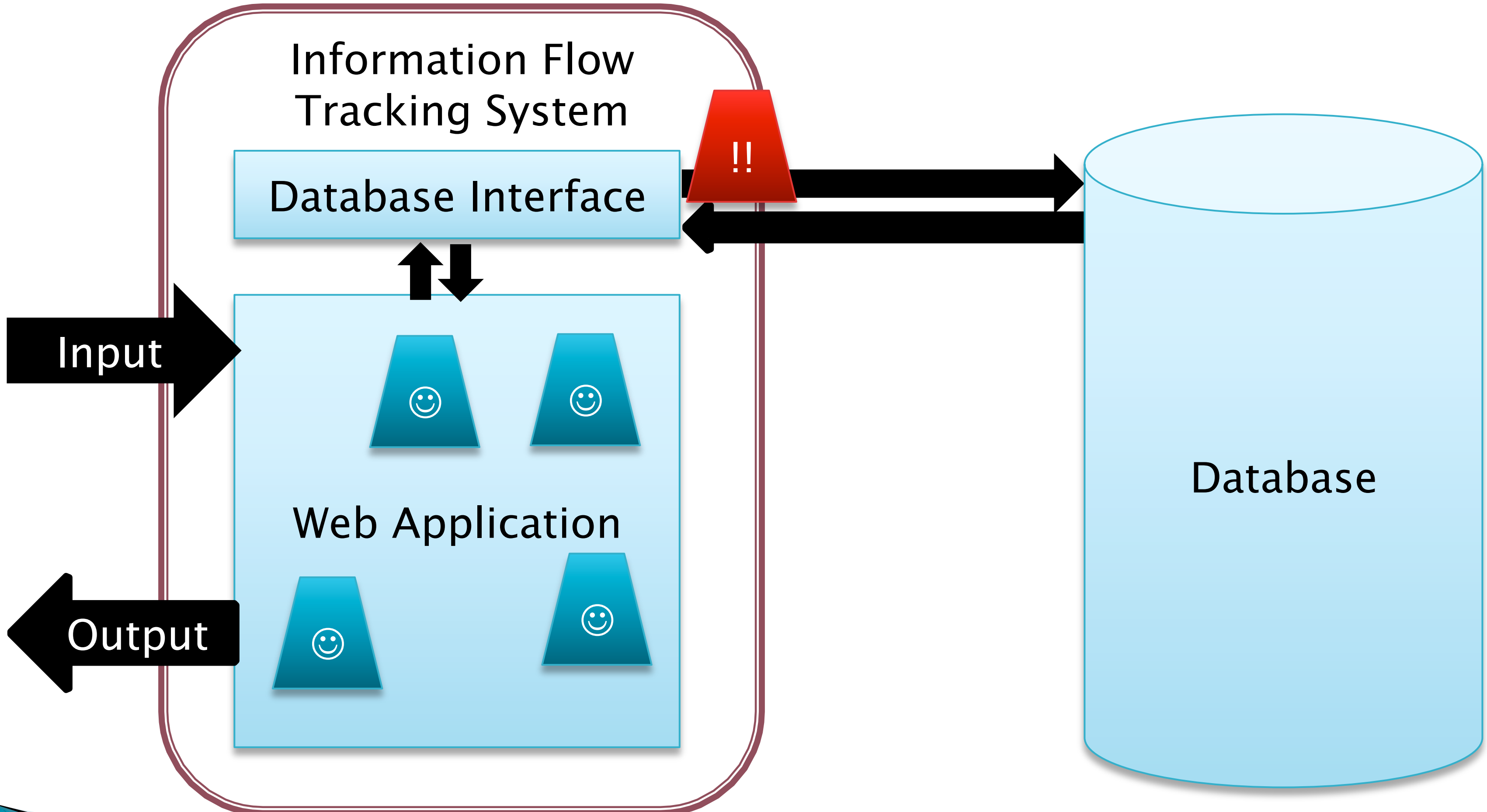
Limitations of Single-Application Systems



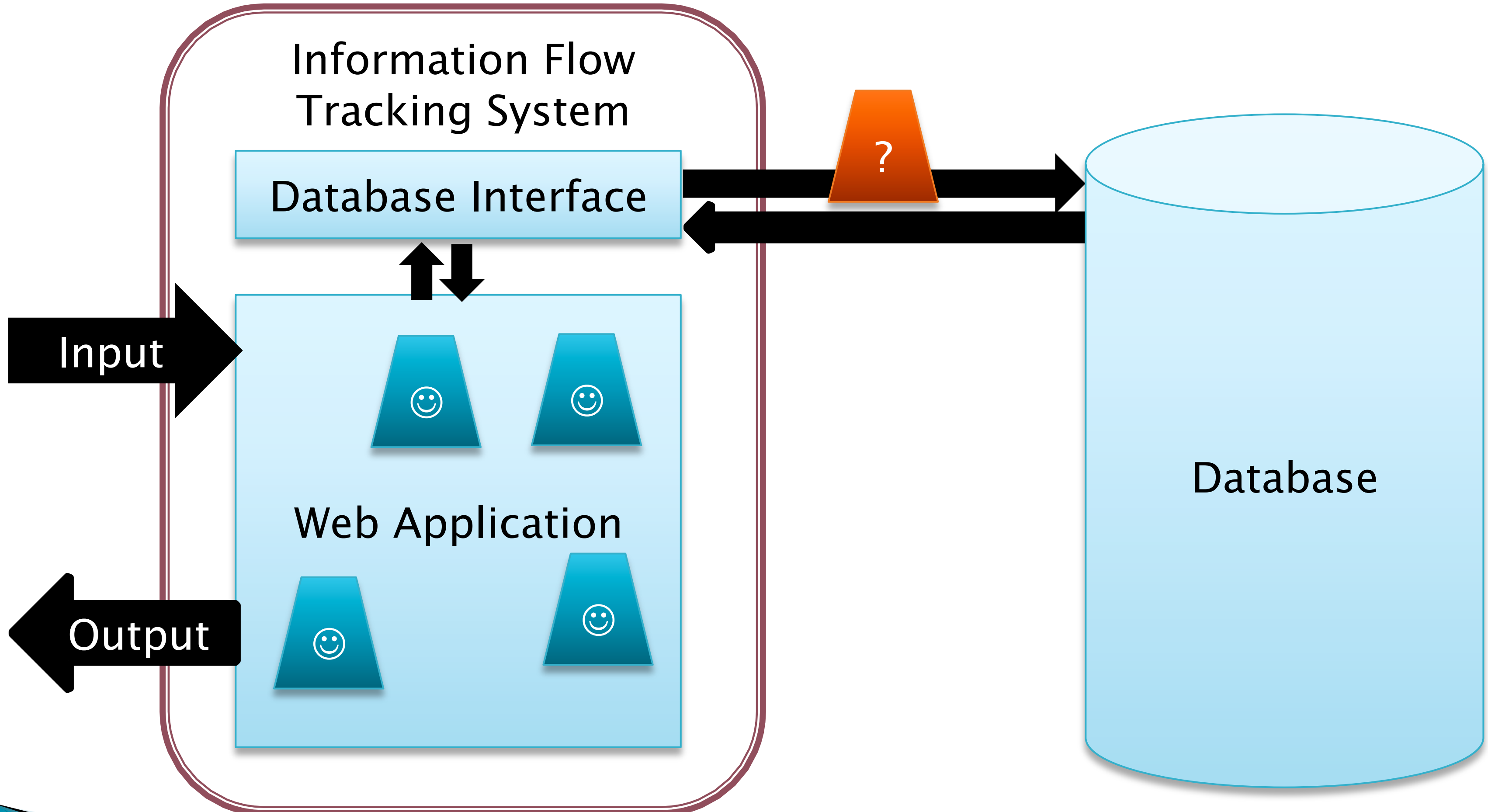
Limitations of Single-Application Systems



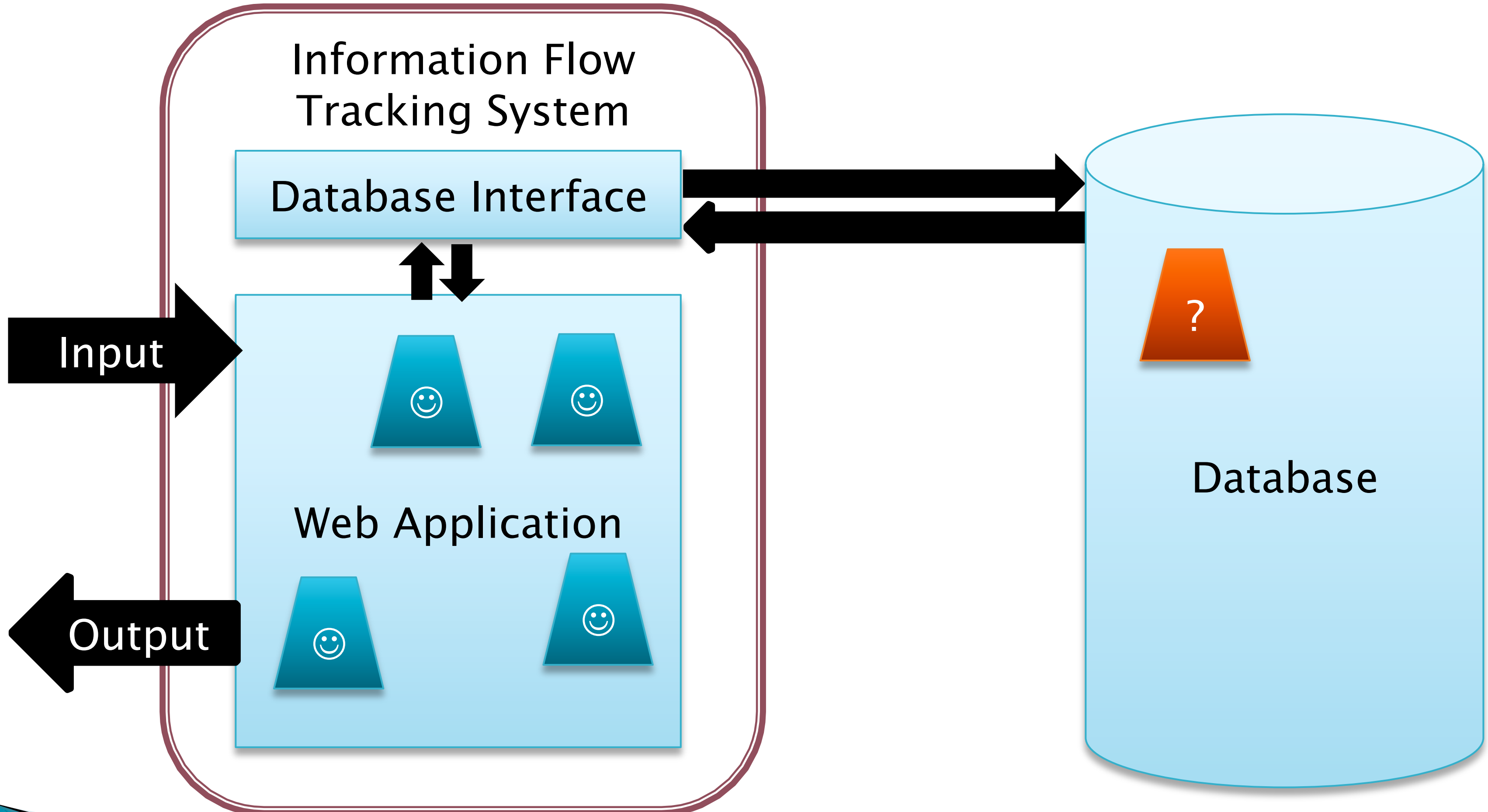
Limitations of Single-Application Systems



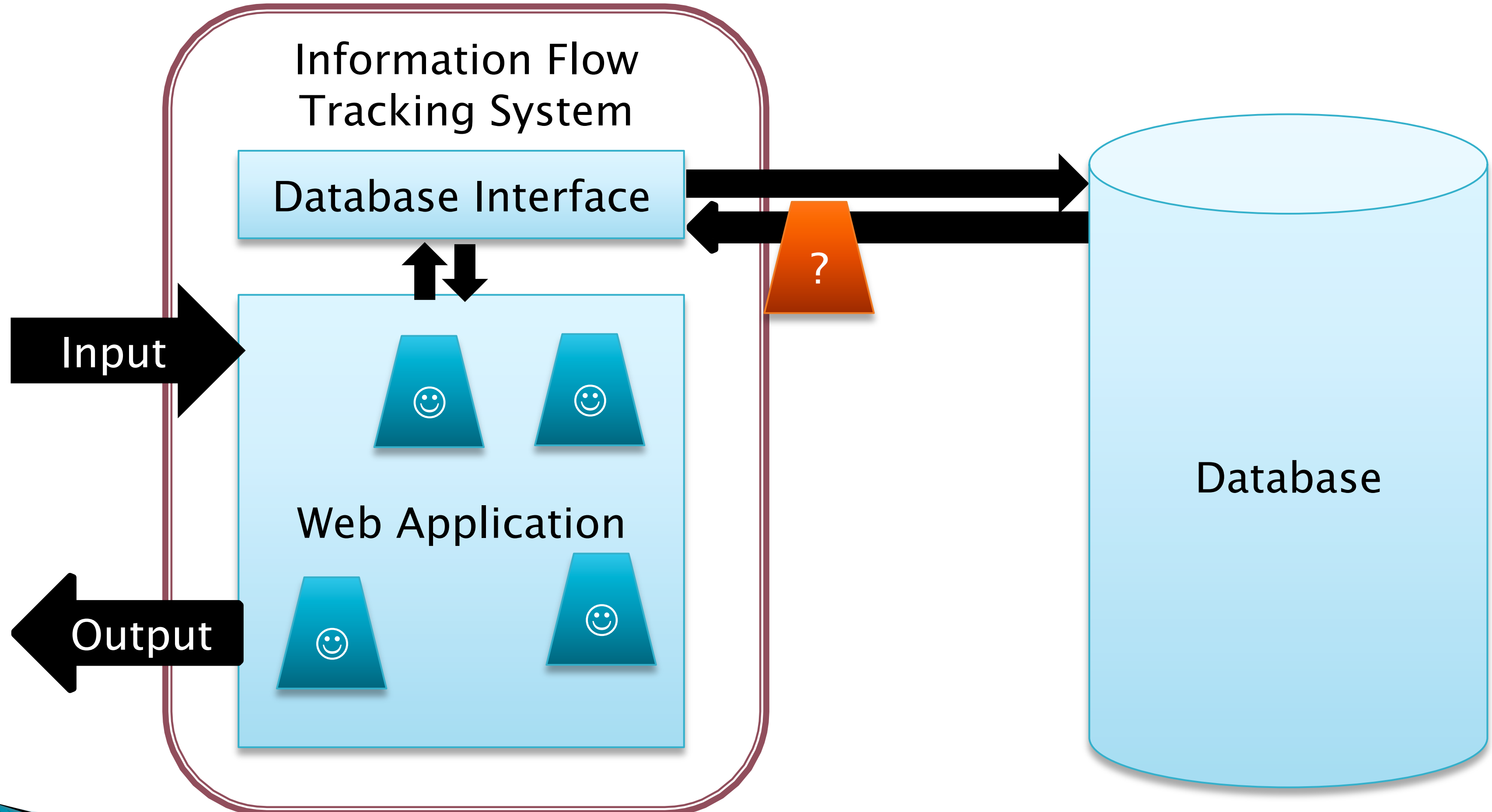
Limitations of Single-Application Systems



Limitations of Single-Application Systems



Limitations of Single-Application Systems



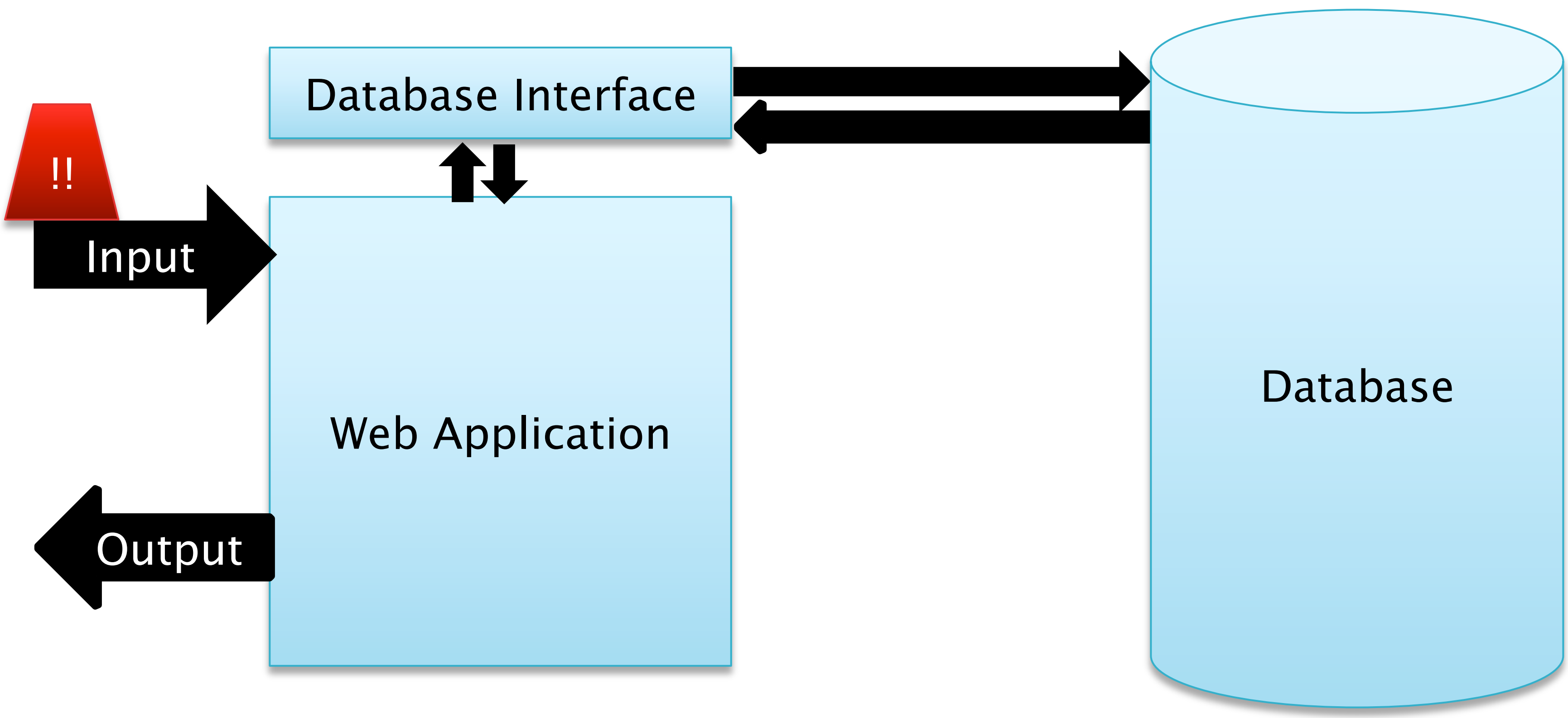
Limitations of Single-Application Systems

- What if you have multiple applications?
- How to treat data from the database?
  - All tainted -> false positives
  - All untainted -> false negatives
  - Require manual annotation?
  - Application-specific decisions?

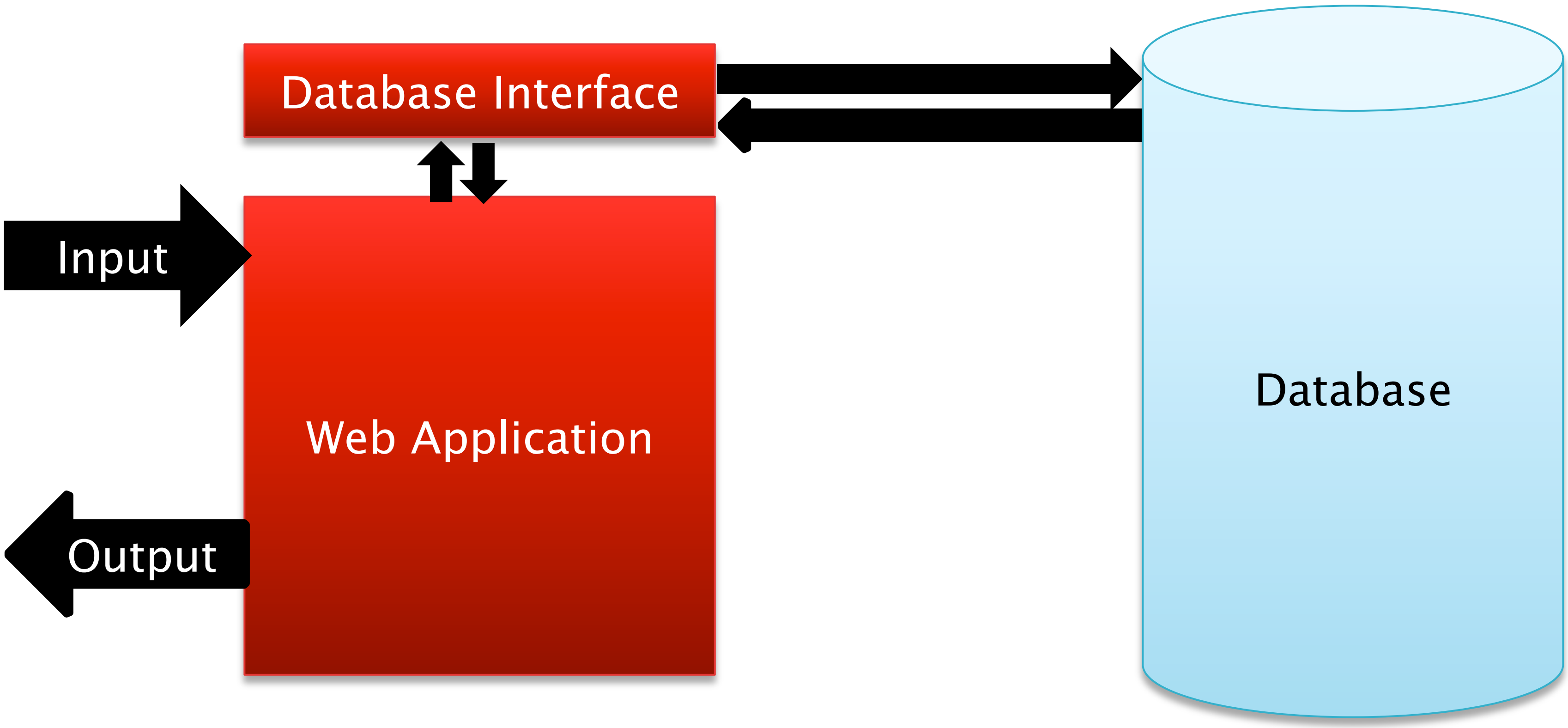
System-Wide Approaches

- Taint tracking through the entire system
  - [Asbestos, 05]
  - [HiStar, 06]
- Implemented in
  - Hardware
  - OS
  - VMM/emulator

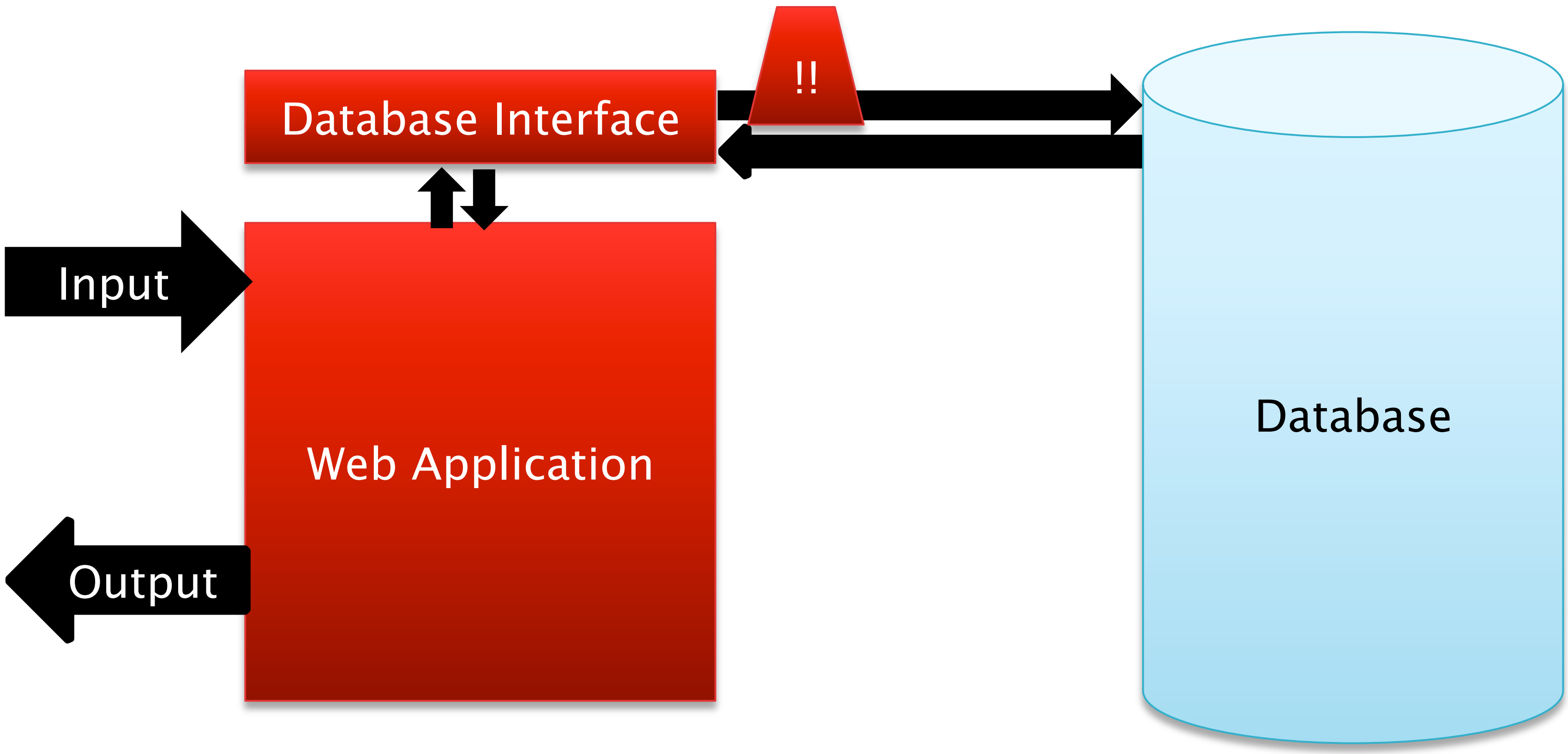
Process-level System-Wide Tracking



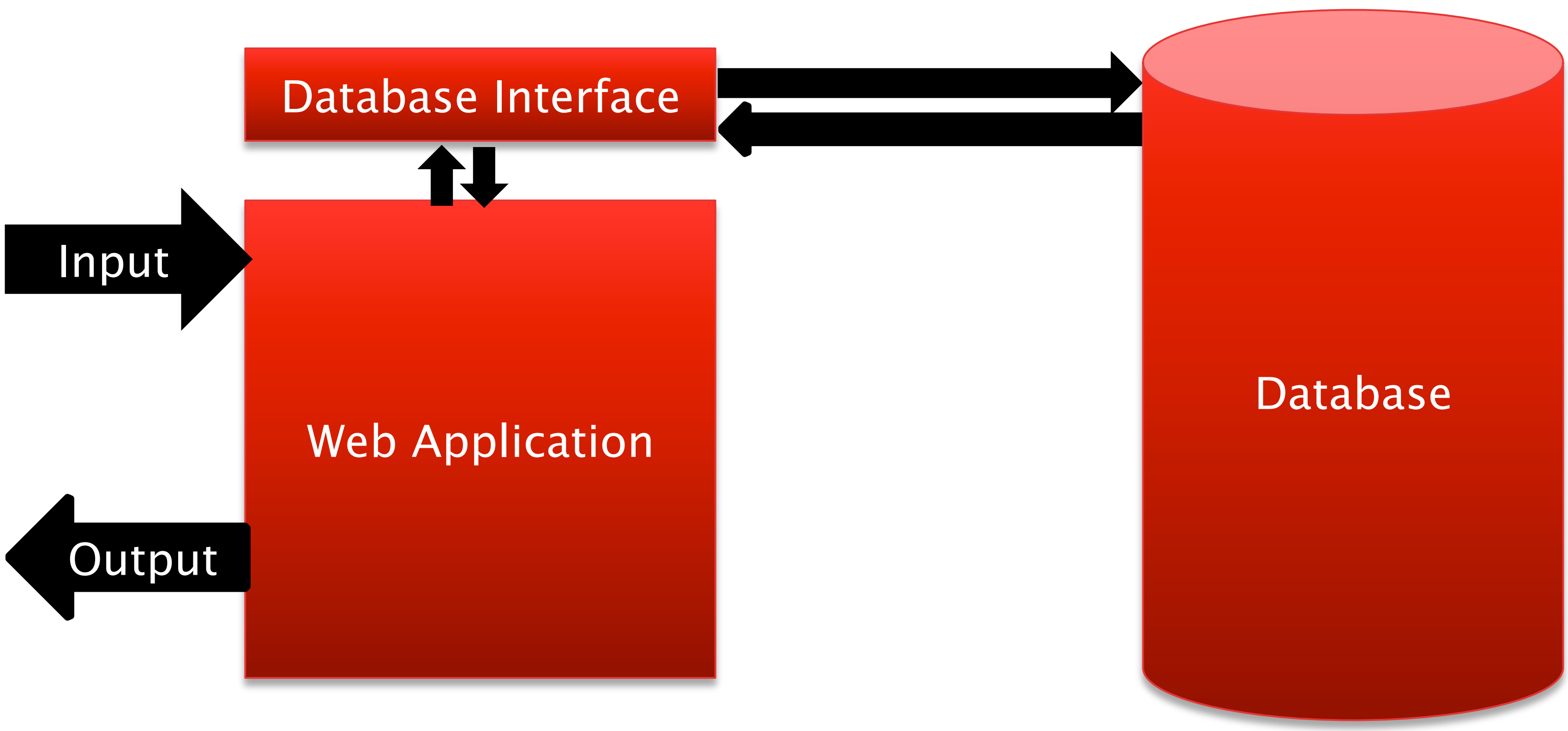
Process-level System-Wide Tracking



Process-level System-Wide Tracking



Process-level System-Wide Tracking



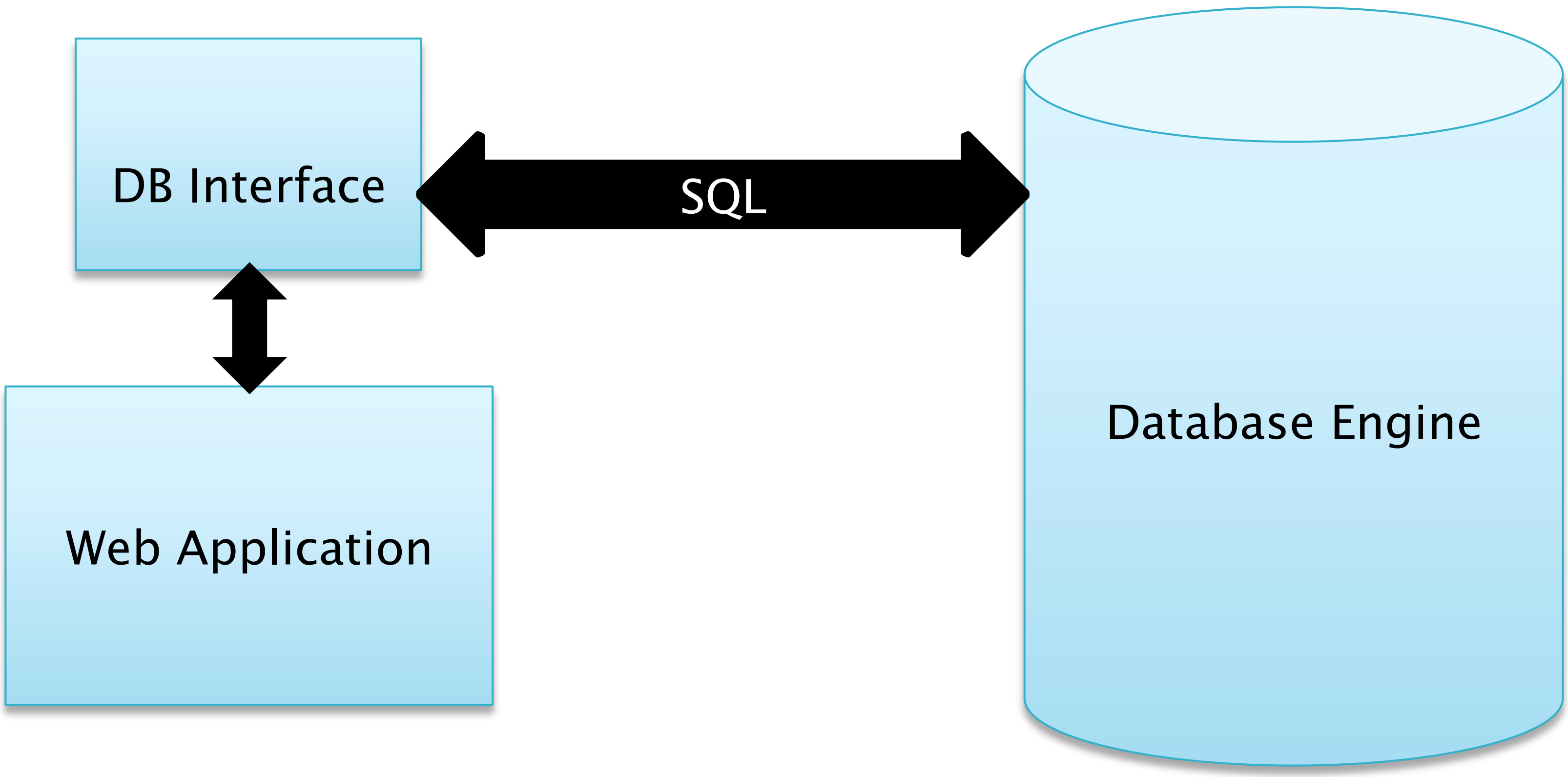
Low-level System-Wide Systems

- Low level/fine granularity
  - Hardware mechanism [Suh, Lee, Devadas 04]
  - Minos [Crandall, Chong, 04]
- Lacks high-level database semantics
  - Aggregate functions
  - Comparisons, SELECT DISTINCT

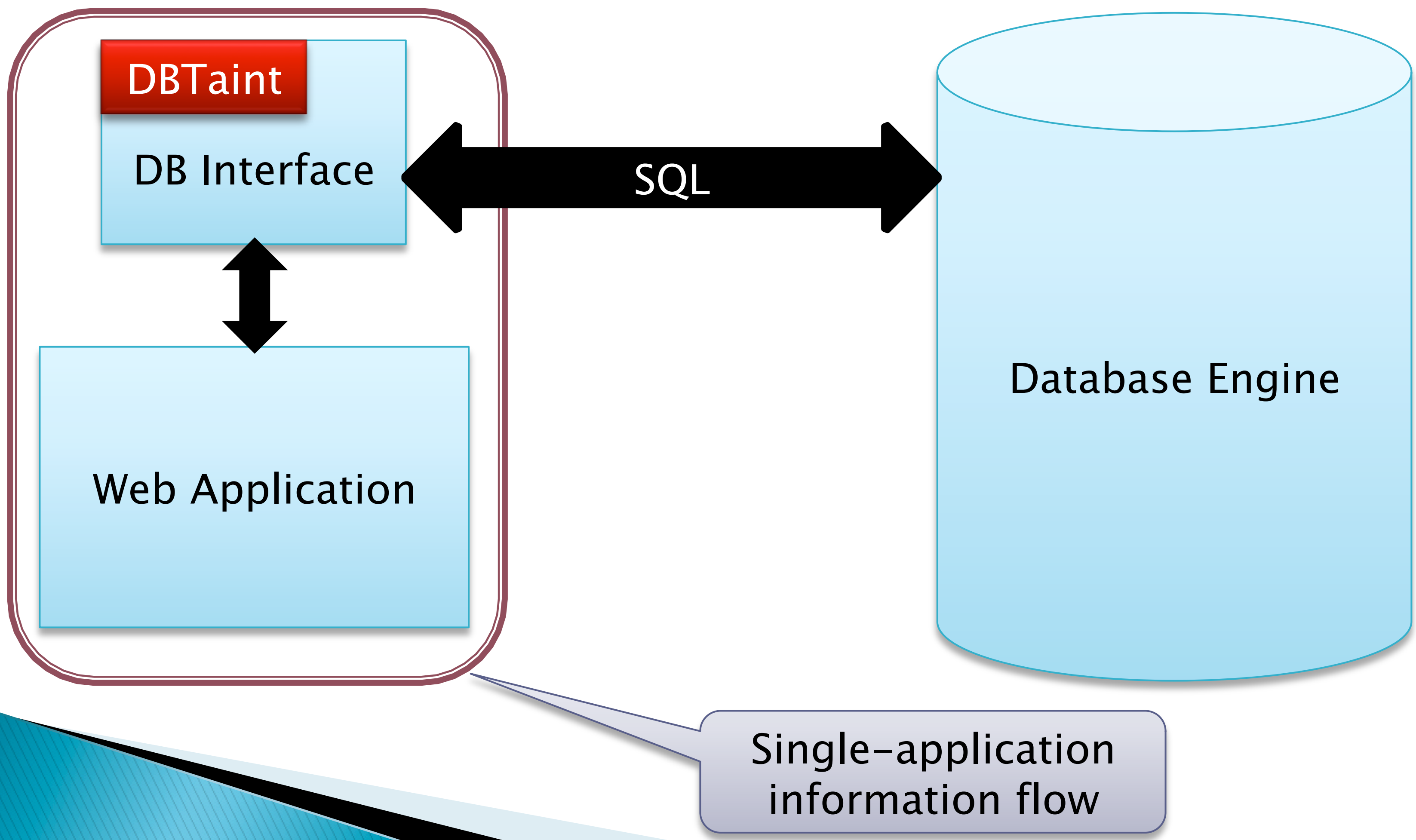
DBTaint Design Goals

- End-to-end taint tracking
  - Across Web applications and databases
- Leverage existing single-application information flow tracking engines
- Compatible with existing Web services
  - Require no changes to Web applications
- Taint propagation through database functions

Web Service



DBTaint Web Service



Information Flow in the Database

- Store taint data in database composite types
  - Tuple of form: (<value>, <taint_value>)
- Store/retrieve taint values via SQL
  - No additional mechanisms needed in the database
  - No change to underlying database data structures

Id	Status
19	‘closed’
27	‘open’
32	‘pending’

Before DBTaint

Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

With DBTaint

Information Flow in the Database

- Create functions that operate on composite types
  - Comparison operators (=, !=, <, …)
  - Arithmetic operations (+, −, …)
  - Text operations (upper, lower, …)
  - Aggregate functions (MAX, MIN, SUM, …)
- Functions implemented in SQL
  - CREATE FUNCTION
  - CREATE OPERATOR
  - CREATE AGGREGATE

Database Taint Behavior

- Arithmetic operations

$$(4, 0) + (5, 1) = \boxed{(9, ?)}$$

Database Taint Behavior

- Arithmetic operations

$$\underbrace{\boxed{(4, 0)}}_{\text{untainted}} + \underbrace{\boxed{(5, 1)}}_{\text{tainted}} = \boxed{(9, ?)}$$

Database Taint Behavior

- Arithmetic operations

$$\boxed{(4,0)} + \boxed{(5,1)} = \boxed{(9,1)}$$

untainted

tainted

tainted

Database Taint Behavior

- MAX

$$\{(2, 0), (3, 1), (5, 0)\} = \boxed{(5, ?)}$$

Database Taint Behavior

- MAX

$$\{\boxed{(2,\ 0)},\ \boxed{(3,\ 1)},\ \boxed{(5,\ 0)}\} = \boxed{(5,\ ?)}$$

untainted tainted untainted

Taint Philosophy

- Untainted: trusted source
  - Web application defaults
  - Values generated entirely by the Web application
- Tainted: from untrusted source, or unknown
  - User input

- Explicit information flow
- Database returns untainted value only if database has received that value untainted

Database Taint Behavior

- MAX

$$\{\underbrace{\boxed{(2, 0)}}_{\text{untainted}}, \underbrace{\boxed{(3, 1)}}_{\text{tainted}}, \underbrace{\boxed{(5, 0)}}_{\text{untainted}}\} = \boxed{(5, ?)}$$

Database Taint Behavior

- MAX

$$\{\underbrace{\boxed{(2,\ 0)}}_{\text{untainted}},\ \underbrace{\boxed{(3,\ 1)}}_{\text{tainted}},\ \underbrace{\boxed{(5,\ 0)}}_{\text{untainted}}\} = \underbrace{\boxed{(5,\ 0)}}_{\text{untainted}}$$

Database Taint Behavior

- Equality

$$(3, 0) \stackrel{?}{=} (3, 1)$$

untainted

tainted

Database Taint Behavior

- Equality

$$3 == 3$$

Database Taint Behavior

- Equality

$$(3, 0) == (3, 1)$$

untainted

tainted

- Adopt notion of backwards-compatibility [Chin, Wagner 09]

Database Taint Behavior

- MAX

$$\{\underbrace{\boxed{(5, 1)}}_{\text{tainted}}, \underbrace{\boxed{(5, 0)}}_{\text{untainted}}\} = \boxed{(5, ?)}$$

Database Taint Behavior

- MAX

$$\{5, 5\} = 5$$

Database Taint Behavior

- MAX

$$\{5, 5\} = 5$$

OR

Database Taint Behavior

- MAX

$$\{\boxed{(5, 1)}, \boxed{(5, 0)}\} = \boxed{(5, ?)}$$

OR

Database Taint Behavior

- MAX

$$\{\underbrace{\boxed{(5, 1)}}_{\text{tainted}}, \underbrace{\boxed{(5, 0)}}_{\text{untainted}}\} = \underbrace{\boxed{(5, 0)}}_{\text{untainted}}$$

- When possible, prefer to return untainted values

Web Service Information Flow

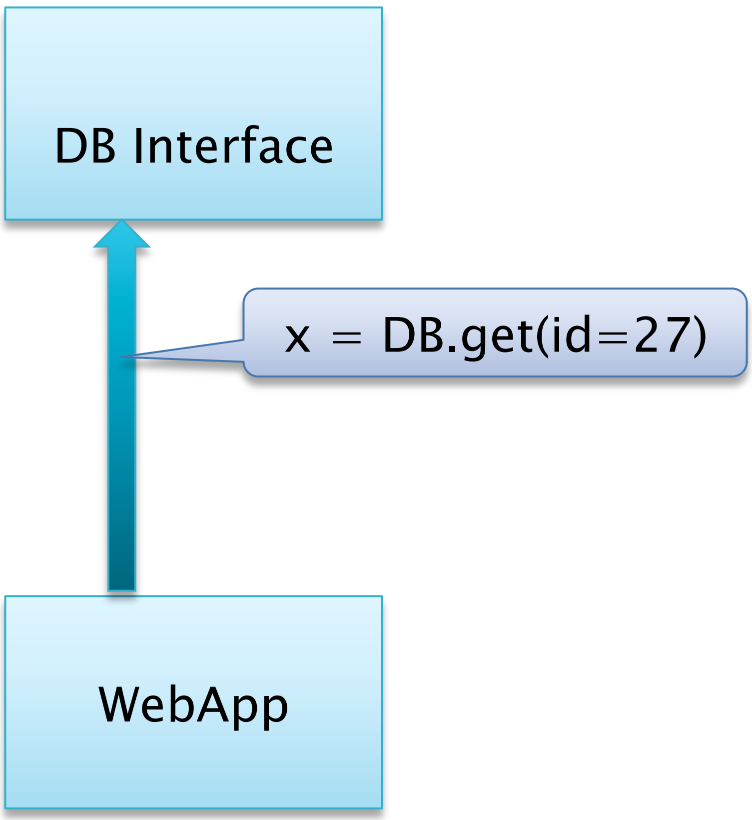
DB Interface

Database Table

Id	Status
19	‘closed’
27	‘open’
32	‘pending’

WebApp

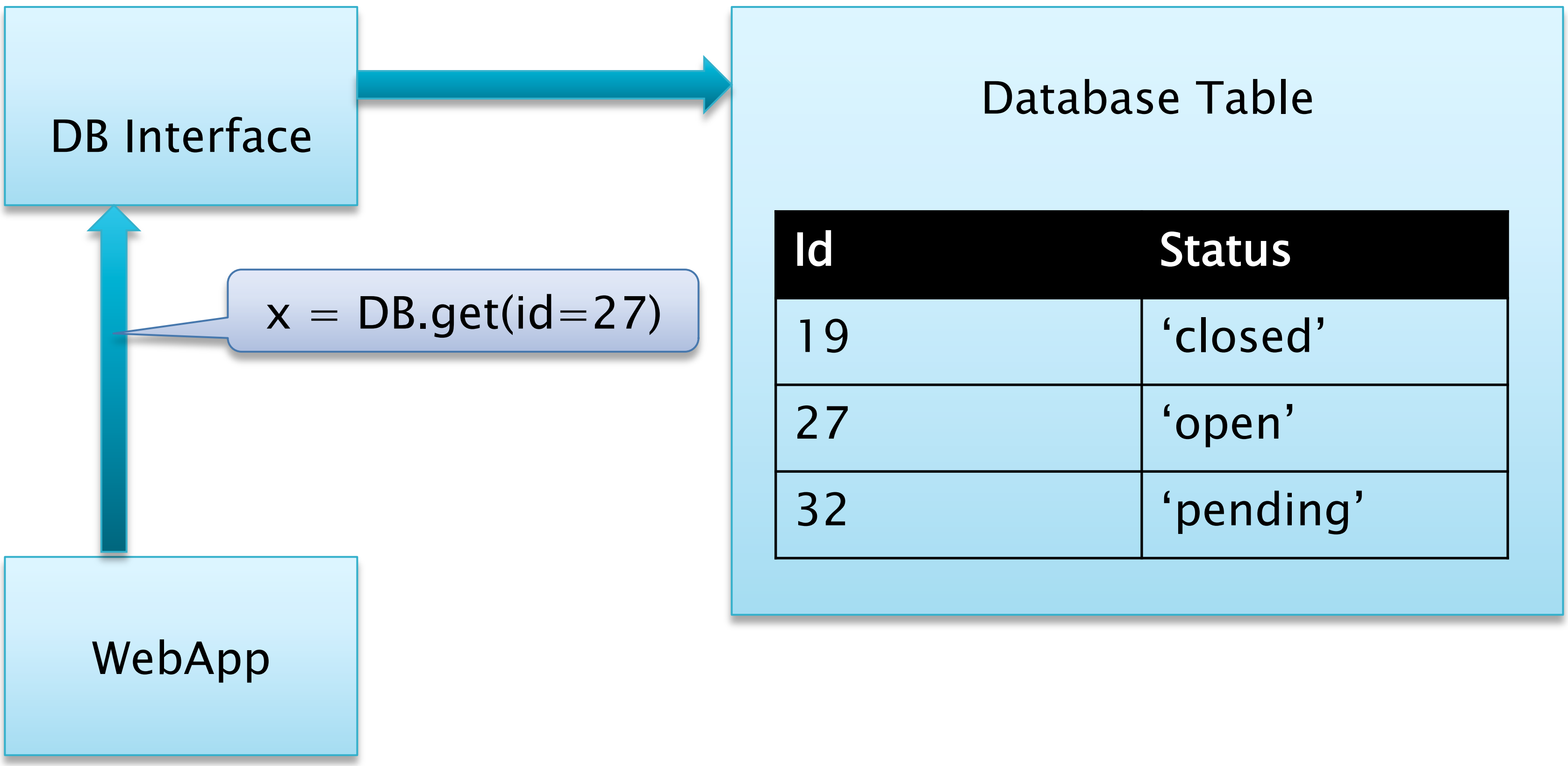
Web Service Information Flow



Database Table

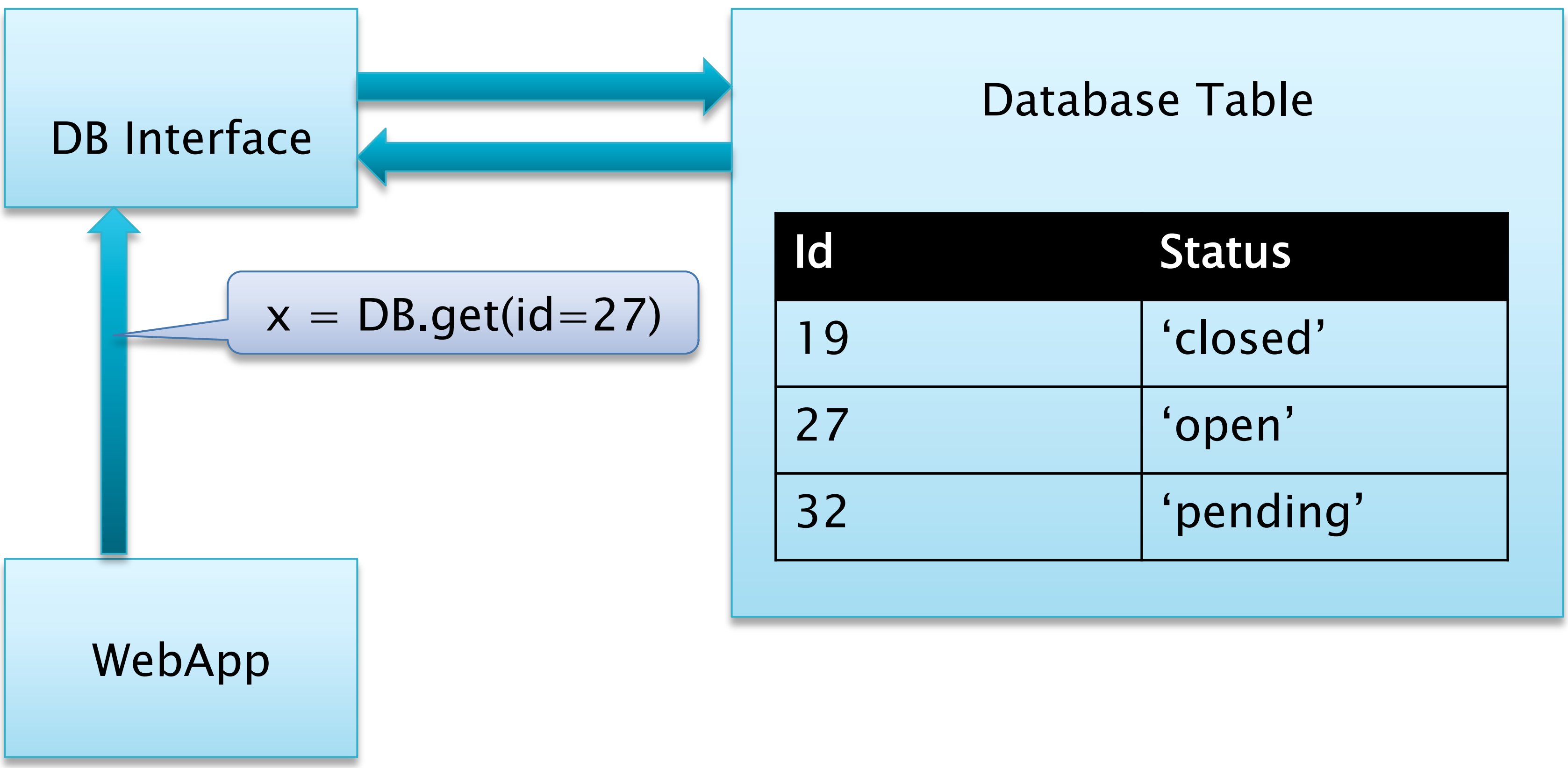
Id	Status
19	‘closed’
27	‘open’
32	‘pending’

Web Service Information Flow

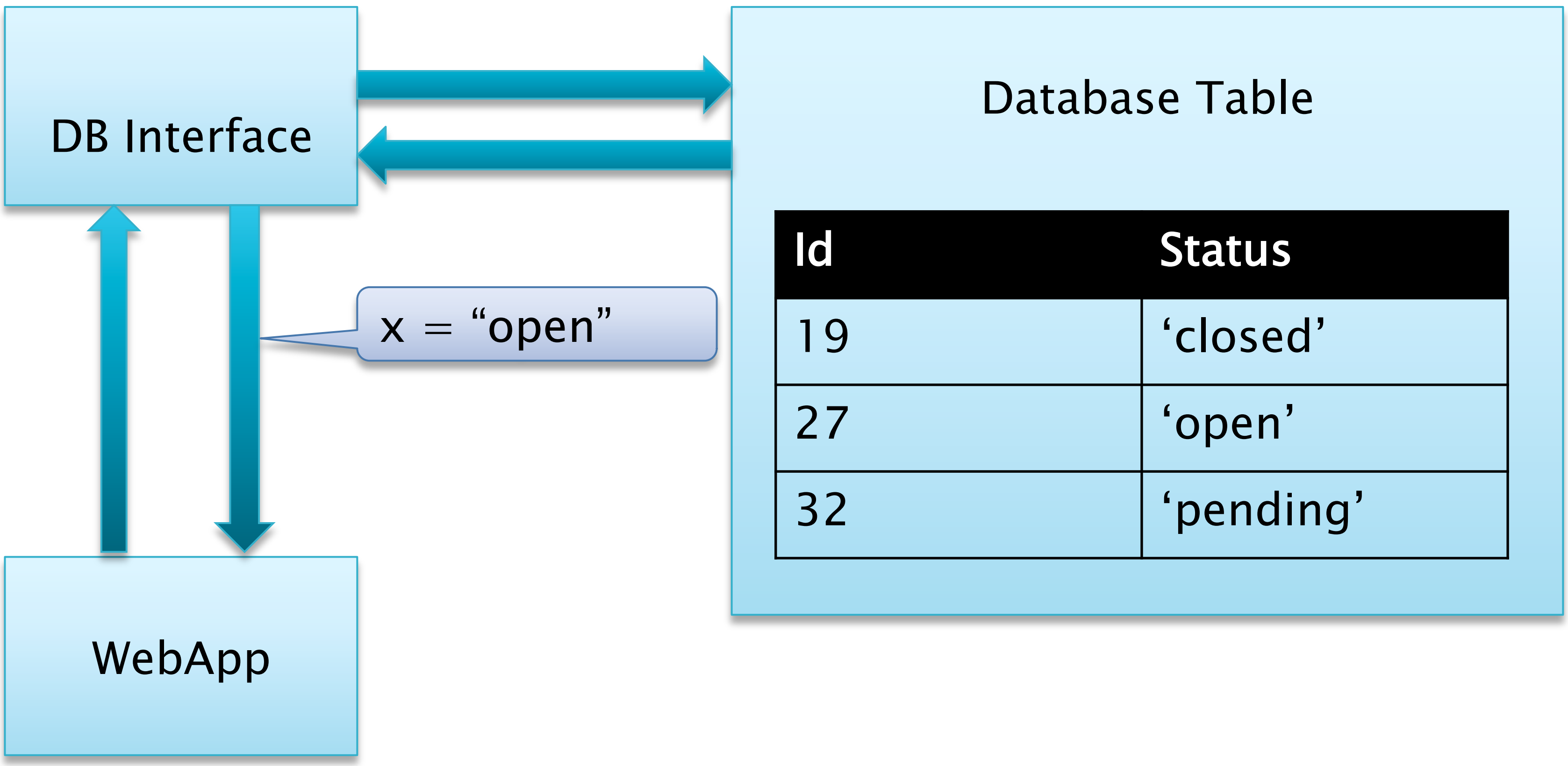


Id	Status
19	‘closed’
27	‘open’
32	‘pending’

Web Service Information Flow



Web Service Information Flow



Id	Status
19	‘closed’
27	‘open’
32	‘pending’

DBTaint Information Flow

DBTaint

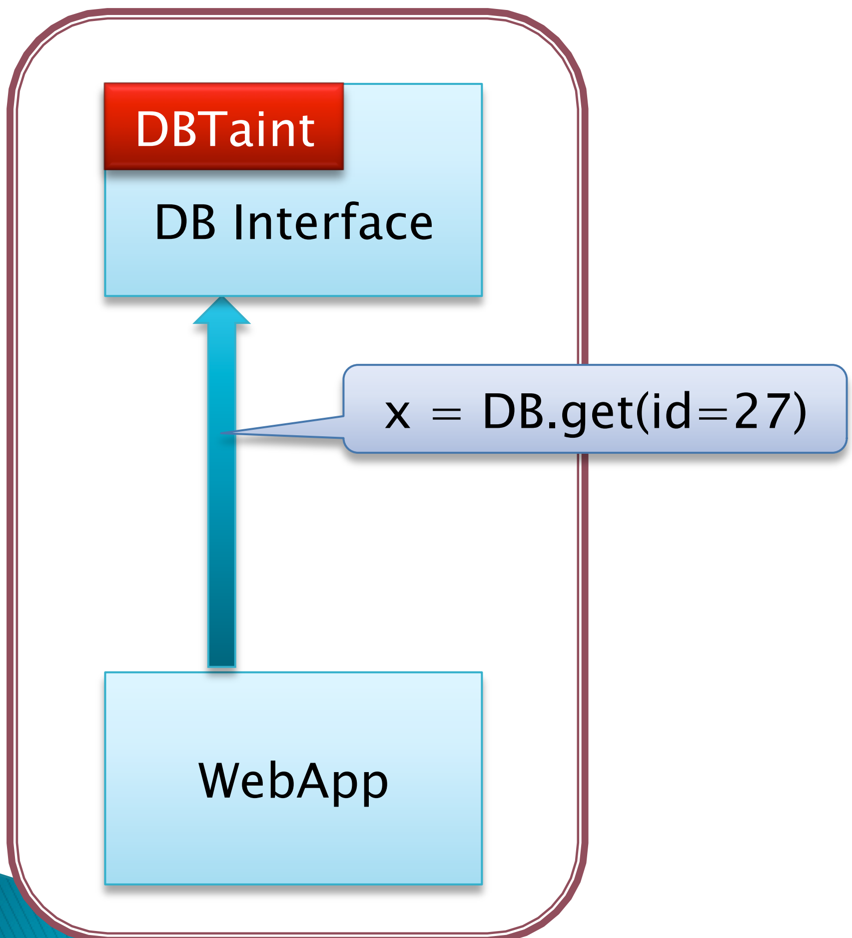
DB Interface

WebApp

Database Table

Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

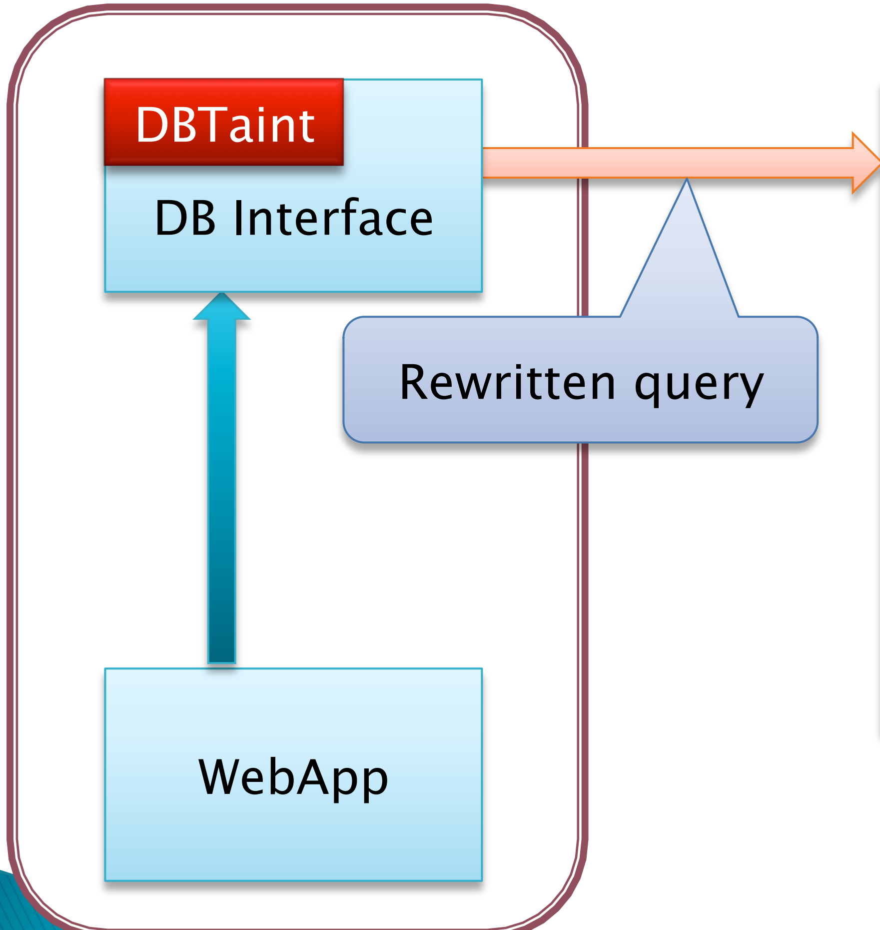
DBTaint Information Flow



Database Table

Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

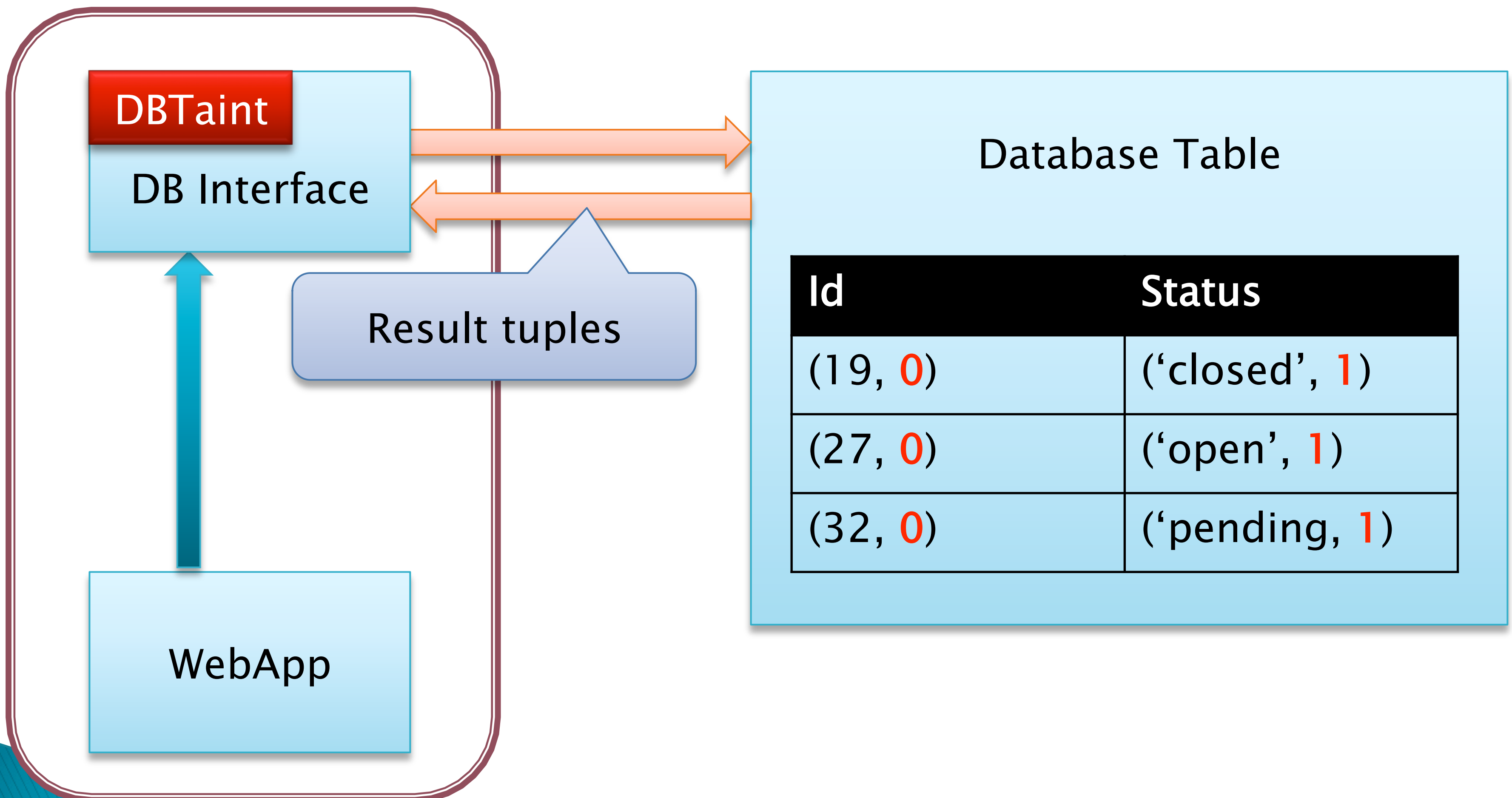
DBTaint Information Flow



Database Table

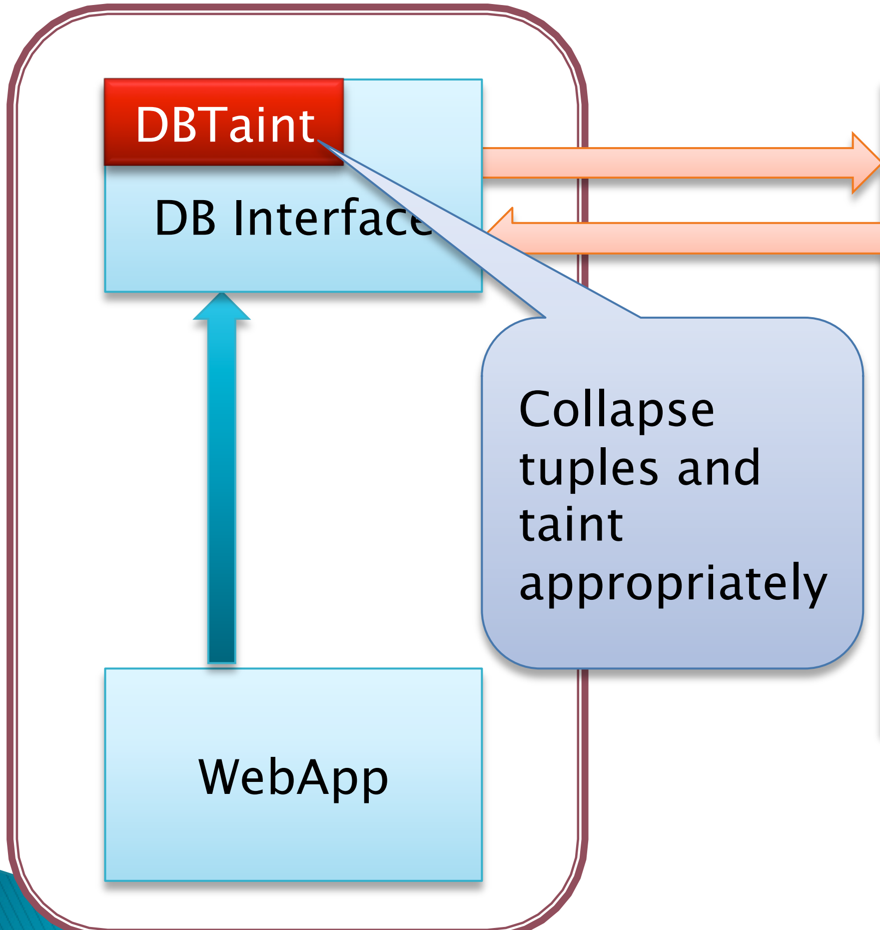
Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

DBTaint Information Flow



Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

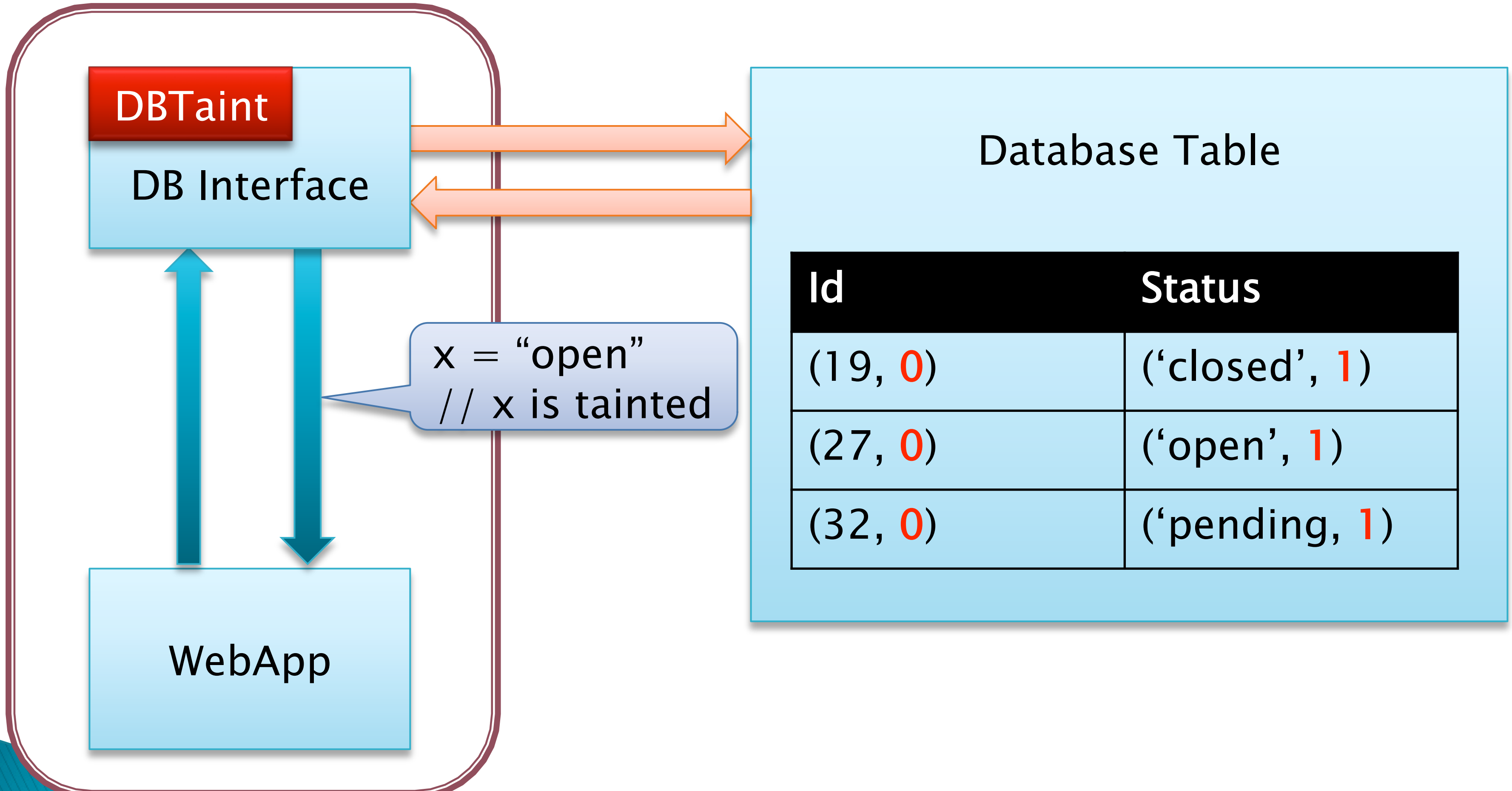
DBTaint Information Flow



Database Table

Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

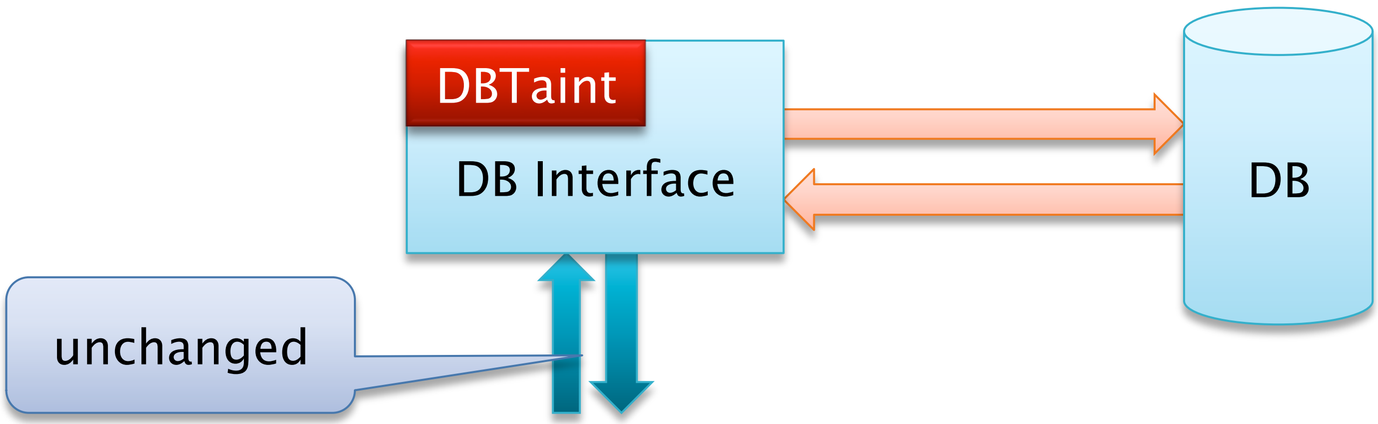
DBTaint Information Flow



Id	Status
(19, 0)	(‘closed’, 1)
(27, 0)	(‘open’, 1)
(32, 0)	(‘pending, 1)

Database Client-Server Integration

- Account for composite types in SQL queries
- Collapse and taint result tuples as needed
- These changes are:
  - Transparent to web application
  - High-level, portable



Example: Parameterized INSERT

- Parameterized queries
- Prepare:
  - INSERT … (id, status) VALUES (?, ?)

- Execute
  - (27, ‘open’)

Example: Parameterized INSERT

- Parameterized queries
- Prepare:
  - INSERT … (id, status) VALUES (?, ?)
  - // with DBTaint:
  - INSERT … (id, status) VALUES (ROW(?, ?), ROW(?, ?))

Example: Parameterized INSERT

- Parameterized queries
- Prepare:
  - INSERT … (id, status) VALUES (?, ?)
  - // with DBTaint:
  - INSERT … (id, status) VALUES (ROW(?, ?), ROW(?, ?))

- Execute
  - (27, ‘open’) // 27 is untainted, ‘open’ is tainted
  - // with DBTaint:
  - (27, 0, ‘open’, 1)

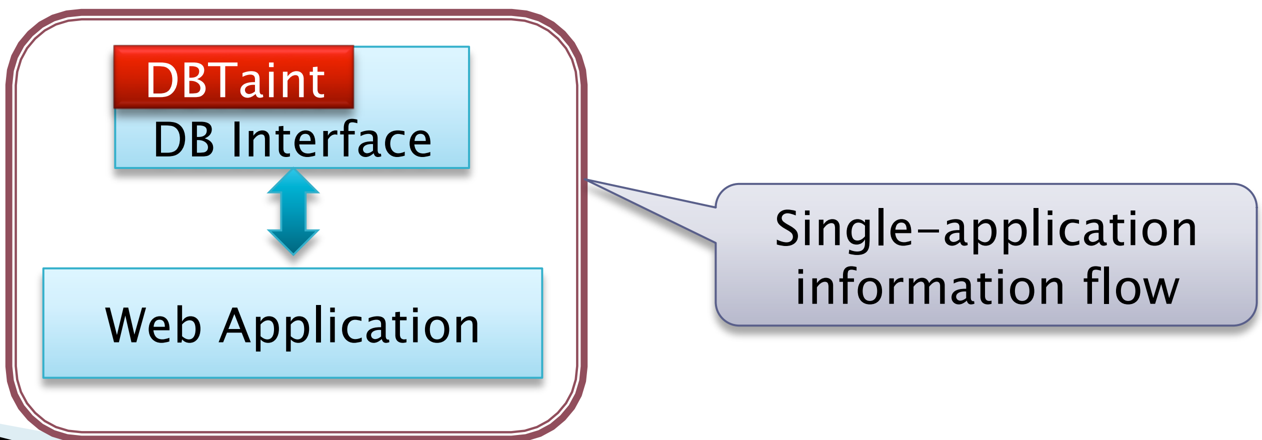
Rewriting Parameterized Queries

- Prepare phase:
  - Queries are passed with placeholders for data
- Execute phase:
  - Data values are passed separately, independently
- Taint tracking engine requirement:
  - Only need to track taint values per variable

- We handle non-parameterized queries too
  - See paper for details

Information Flow in the Web App

- Leverage existing single-application information flow tracking systems
- No changes to Web application



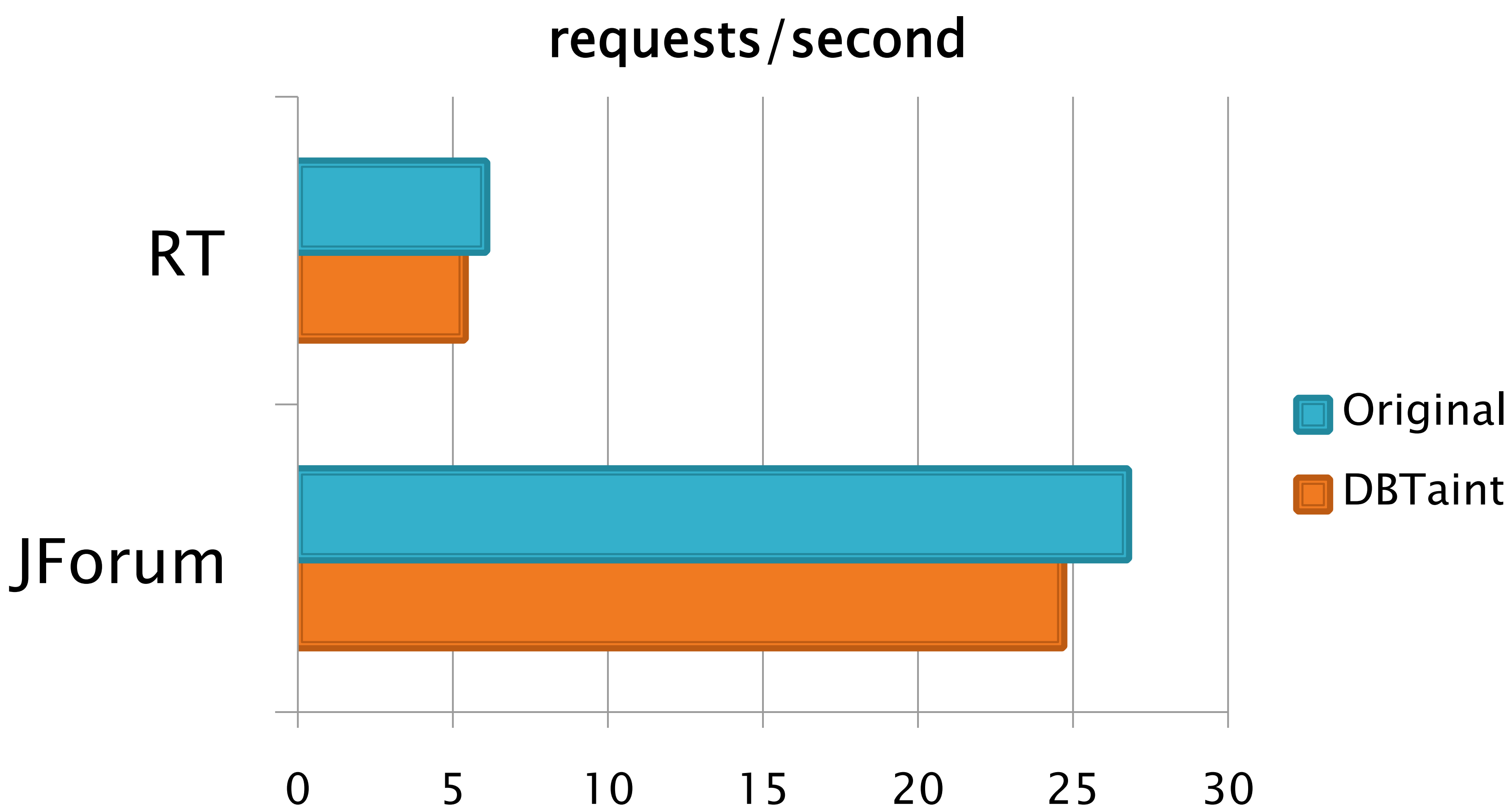
Implementation

- Languages
  - Perl
  - Java
- Database Interfaces
  - Perl DataBase Interface (DBI)
  - Java Database Connectivity (JDBC)
- Database
  - PostgreSQL

Evaluation

- RT: Request Tracker (ticket tracking system)
  - 60,000+ lines of Perl
  - Perl DBI (DataBase Interface) API
  - Perl taint mode
- JForum (discussion board system)
  - 30,000+ lines of Java
  - Java Database Connectivity (JDBC) API
  - Character-level taint engine [Chin, Wagner '09]

Performance Evaluation



Enhanced Taint Tracking

- Cross-application information flow tracking
- Persistent taint tracking
- Multiple Web applications, multiple Databases

Conclusion

- End-to-end information flow through Web services
- Compatible with existing Web services
  - Requires no changes to Web applications
- Taint propagation through database functions