# Enforcing Authorization Policies using Transactional Memory Introspection

Arnar Birgisson
Úlfar Erlingsson

Reykjavik University

Mohan Dhawan
Vinod Ganapathy
Liviu Iftode

Rutgers University

# Take home slide

We can utilize the mechanisms of
Software Transactional Memory
to greatly improve
security policy enforcement

# Security policy enforcement

Common difficulties
with policy enforcement:

Difficulty 1

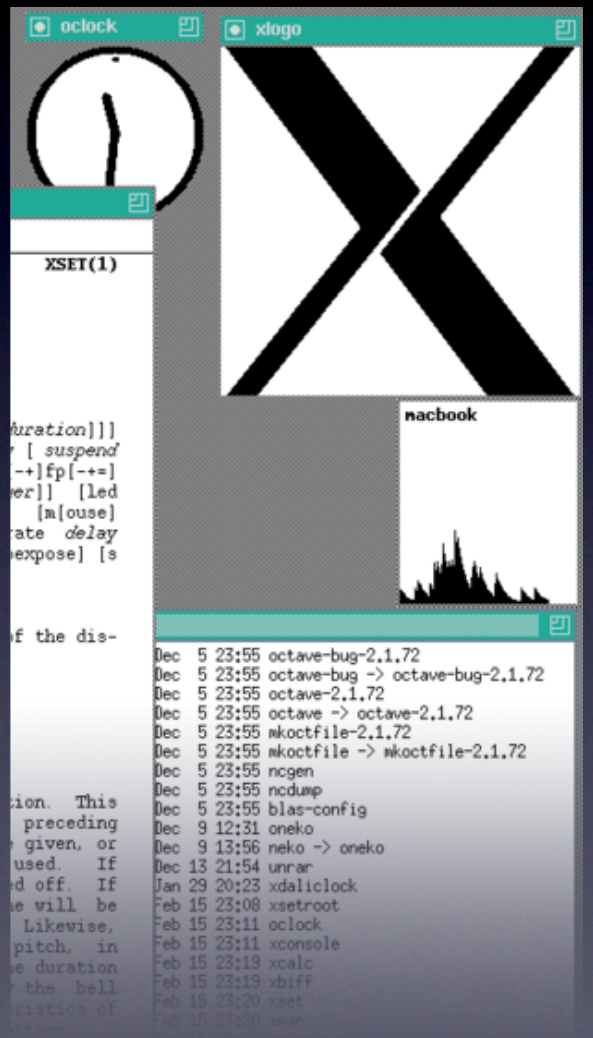
Time of check vs. time of use

Difficulty 2

Correctly handling failures

Difficulty 3

Ensuring complete mediation



# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```

# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```

# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {

    perform operation on resource
}
```

# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {
```

**Other thread may run here!**

```
    perform operation on resource
}
```

# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {
```

**Other thread may run here!**

```
    perform operation on resource
}
```

Interleaving code may invalidate the check

Need a synchronization mechanism.

# Policy enforcement difficulty 1
# Time of check vs. time of use

```
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```

# Solution for difficulty 1
# Software Transactional Memory (STM)

```
if (allowed(principal, resource, operation)) {
    perform operation on resource
}
```

# Solution for difficulty 1
# Software Transactional Memory (STM)

```
atomically {
  if (allowed(principal, resource, operation)) {
      perform operation on resource
  }
}
```

# Solution for difficulty 1
# Software Transactional Memory (STM)

```
atomically {
  if (allowed(principal, resource, operation)) {
      perform operation on resource
  }
}
```

Uses parallel, speculative execution

# Solution for difficulty 1
# Software Transactional Memory (STM)

```
atomically {
  if (allowed(principal, resource, operation)) {
      perform operation on resource
  }
}
```

Uses parallel, speculative execution

Monitors all access to memory

# Solution for difficulty 1
# Software Transactional Memory (STM)

```
atomically {
  if (allowed(principal, resource, operation)) {
     perform operation on resource
  }
}
```

Uses parallel, speculative execution

Monitors all access to memory

Can roll back and retry on conflict

# Solution for difficulty 1
# Software Transactional Memory (STM)

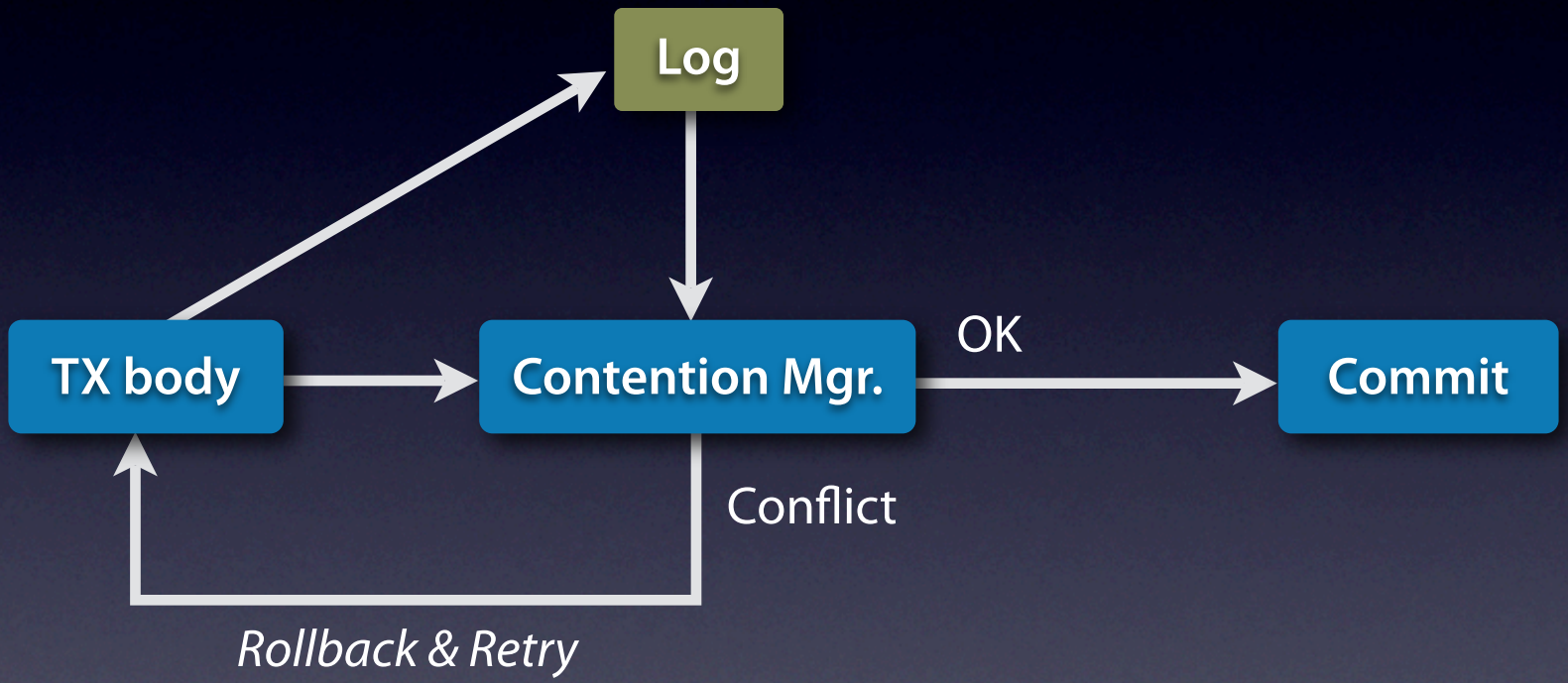
```
atomically {
  if (allowed(principal, resource, operation)) {
      perform operation on resource
  }
}
```

STM guarantees atomicity and isolation of atomic blocks.

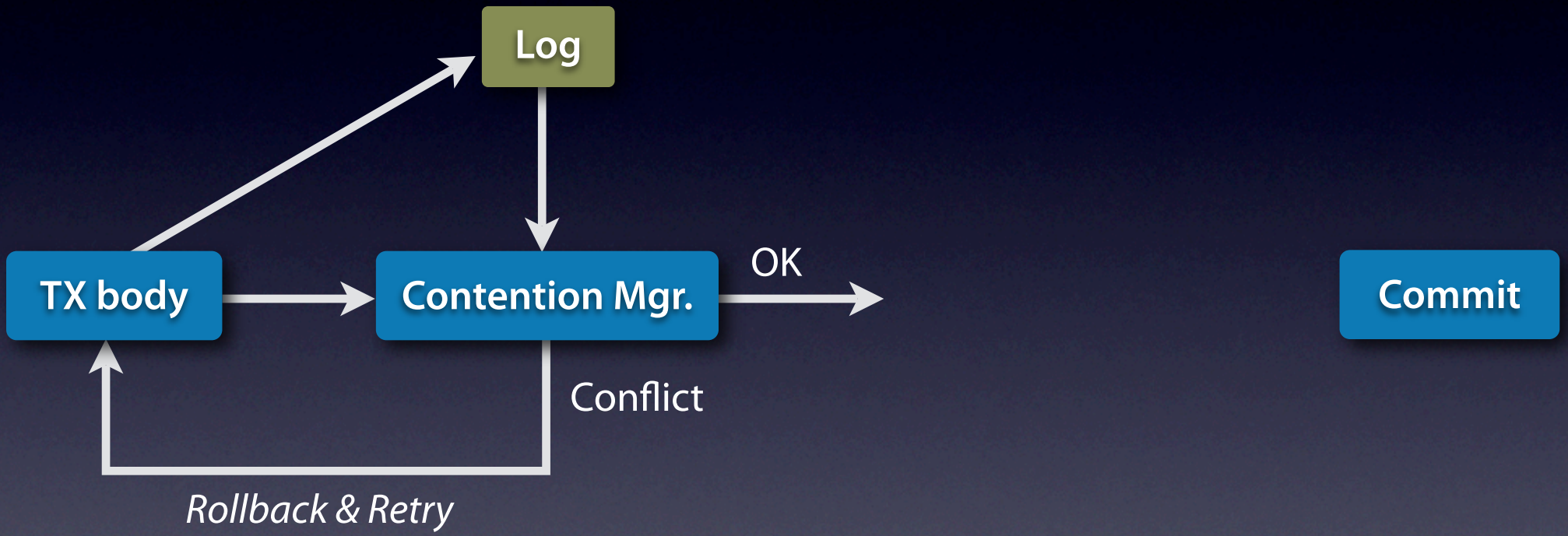
# Transactional Memory Introspection (TMI)

Utilizes the mechanisms of
Software Transactional Memory
— such as its bookkeeping —
to greatly improve
security policy enforcement

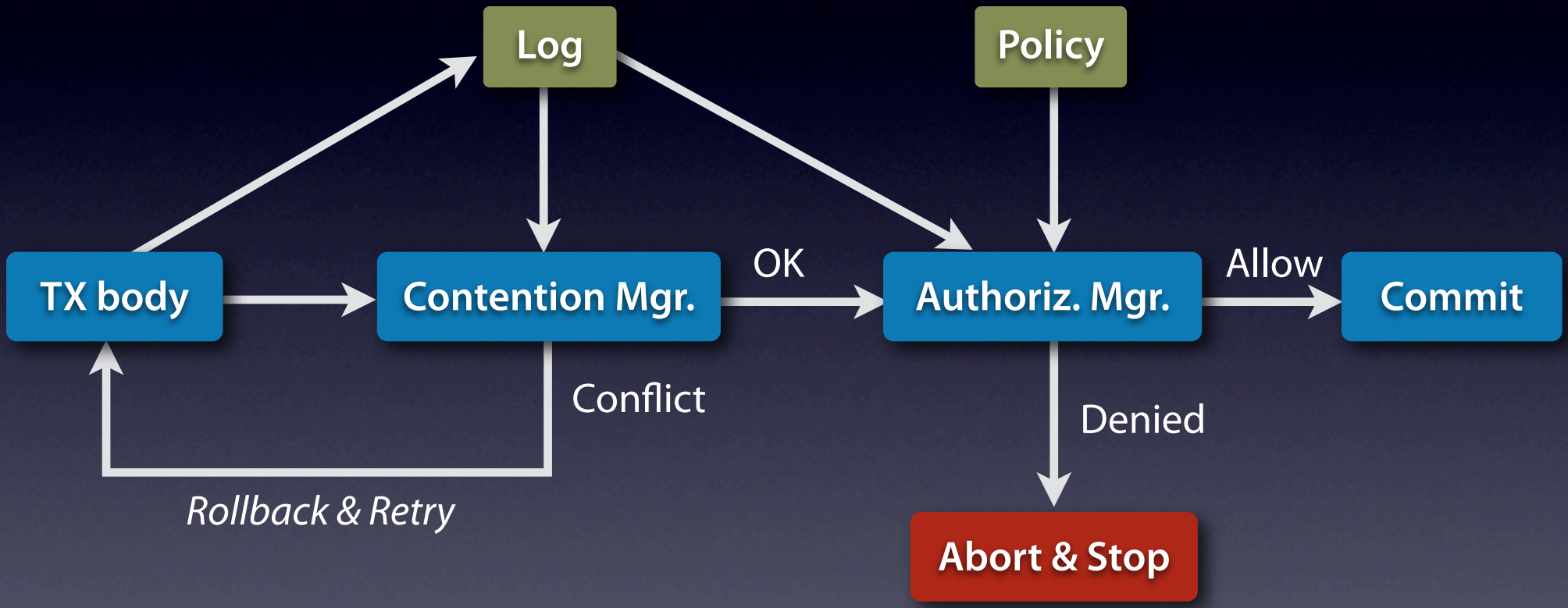
# Where TMI fits in with STM



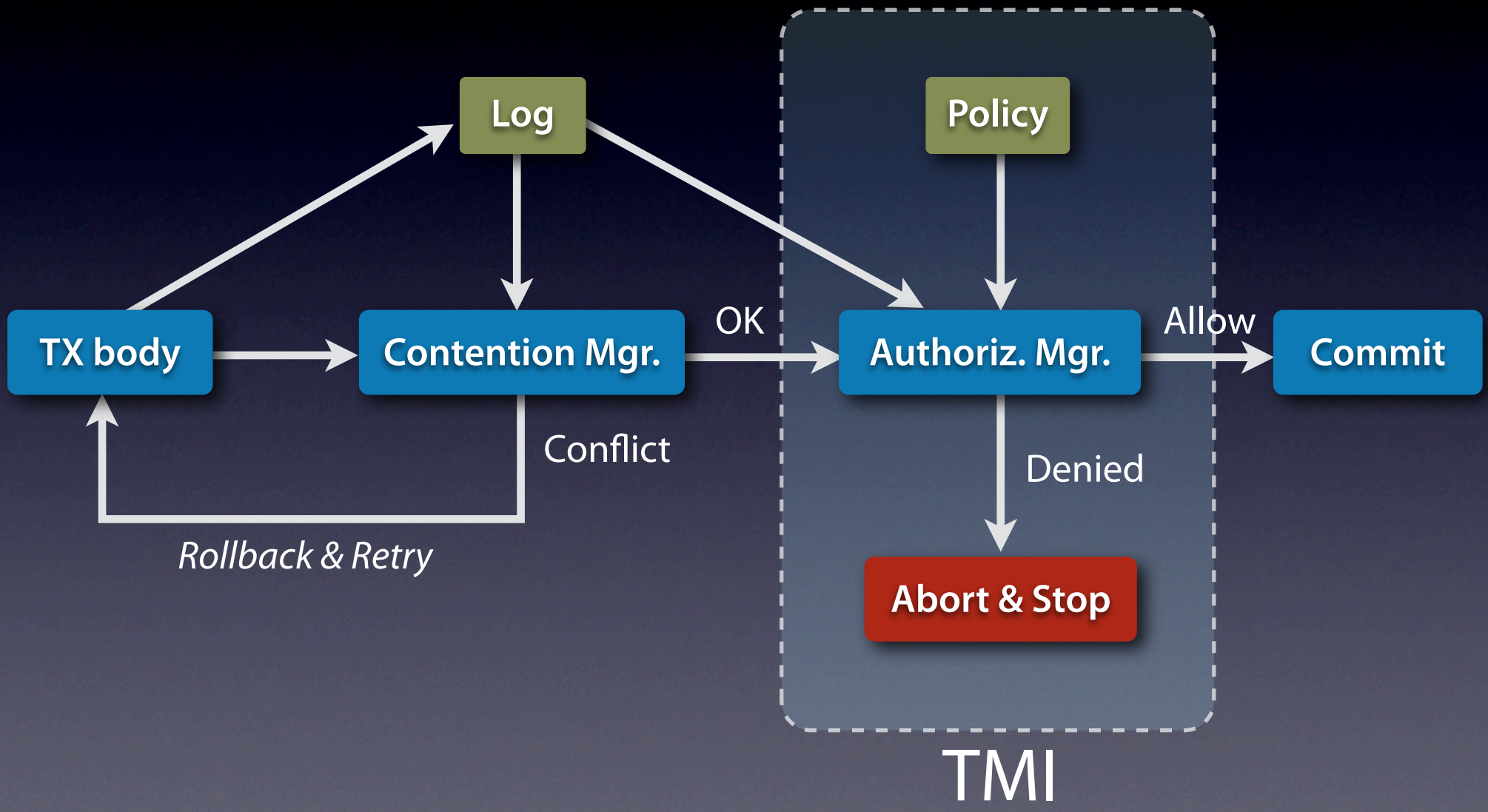
# Where TMI fits in with STM



# Where TMI fits in with STM



# Where TMI fits in with STM



# Policy enforcement difficulty 2
# Error handling

```
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}
```

# Policy enforcement difficulty 2
# Error handling

```
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```

# Policy enforcement difficulty 2
# Error handling

```
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```

This quickly becomes hard to manage

# Policy enforcement difficulty 2
# Error handling *(cont'd)*

- Error handling accounts for a large fraction of server software, over two-thirds [IBM'87]
- Exception handling code itself is prone to errors [Fetzer and Felber '04]
- SecurityException is the one most often handled incorrectly [Weimer & Necula OOPSLA'04]

# Policy enforcement difficulty 3
# Complete mediation

```
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```

# Policy enforcement difficulty 3
# Complete mediation

```
if (allowed(principal, resource1, op1)) {
    perform op1 on resource1
} else {
    clean up and report error
}

if (allowed(principal, resource2, op2)) {
    perform op2 on resource2
} else {
    clean up after op1;
    clean up after op2 and report error
}
```

Easy to forget or miss checks in complex code

# Policy enforcement difficulty 3
# Complete mediation *(cont'd)*

- A real problem in current practice
- Bugs of this kind found in the Linux kernel, `page_cache_read` did not check for file permissions [Zhang *et al*. USENIX Security '02]
- Decentralized, ad-hoc hard-coded access checks, leads to errors when code changes.
- Also a problem in Linux [Jaeger *et al*. '04]

TMI solves these difficulties
and
simplifies application code

# Handling errors with STM abort

```
atomically {
    if (allowed(principal, resource1, op1)) {
       perform op1 on resource1;
    } else {
        clean up and report error
    }

    if (allowed(principal, resource2, op2)) {
       perform op2 on resource2;
    } else {
        clean up after op1;
        clean up after op2 and report error
    }
}
```

# Handling errors with STM abort

```
atomically {
    if (allowed(principal, resource1, op1)) {
       perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
       perform op2 on resource2;
    }
}
```

# Handling errors with STM abort

```
atomically {
    if (allowed(principal, resource1, op1)) {
       perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
       perform op2 on resource2;
    }
} on abort {
    report error;  // no cleanup necessary
}
```

# Complete mediation and decoupling of enforcement code

```
atomically {
    if (allowed(principal, resource1, op1)) {
       perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
       perform op2 on resource2;
    }
} on abort {
    report error;  // no cleanup necessary
}
```

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
    if (allowed(principal, resource1, op1)) {
        perform op1 on resource1;
    }
    if (allowed(principal, resource2, op2)) {
        perform op2 on resource2;
    }
} on abort {
    report error;  // no cleanup necessary
}
```

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
        perform op1 on resource1;
        perform op2 on resource2;
} on abort {
    report error;  // no cleanup necessary
}
```

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
        perform op1 on resource1;
        perform op2 on resource2;
} on abort {
    report error;  // no cleanup necessary
}
```

The TMI reference monitor is invoked

- on every security-relevant memory access
- before every transaction commit

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
        perform op1 on resource1;
        perform op2 on resource2;
} on abort {
    report error;  // no cleanup necessary
}
```

The TMI reference monitor is invoked

- on every security-relevant memory access
- before every transaction commit

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
        perform op1 on resource1;
        perform op2 on resource2;
} on abort {
    report error;   // no cleanup necessary
}
```

The TMI reference monitor is invoked

- on every security-relevant memory access
- before every transaction commit

# Complete mediation and decoupling of enforcement code

```
atomically [principal] {
        perform op1 on resource1;
        perform op2 on resource2;
} on abort {
    report error;   // no cleanup necessary
}
```

The TMI reference monitor is invoked

- on every security-relevant memory access
- before every transaction commit

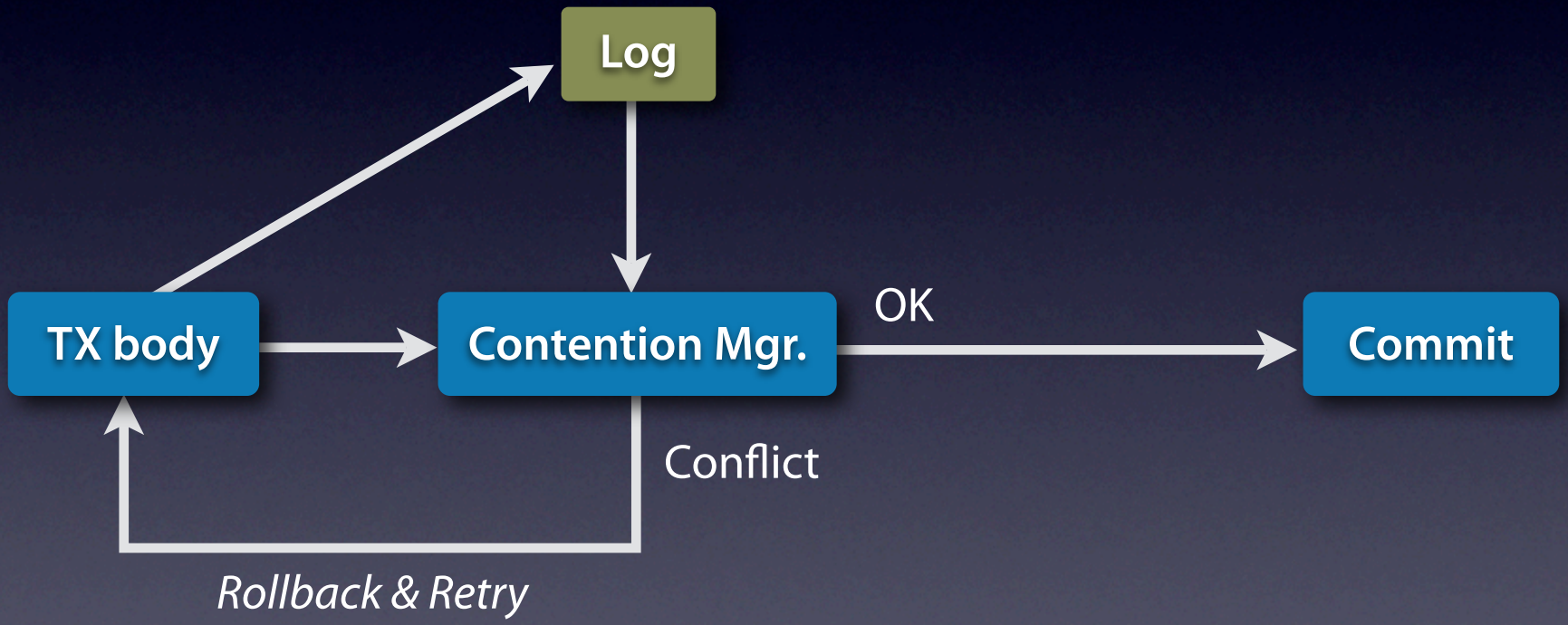
A TMI *authorization manager* must make a policy decision — but can do so at any time before commit.

# Policy evaluation in TMI

- *TMI authorization manager* evaluates the policy
- Supplied by the programmer, decoupled from application logic
- Invoked on *all* accesses to sensitive resources. *Complete mediation for free.*

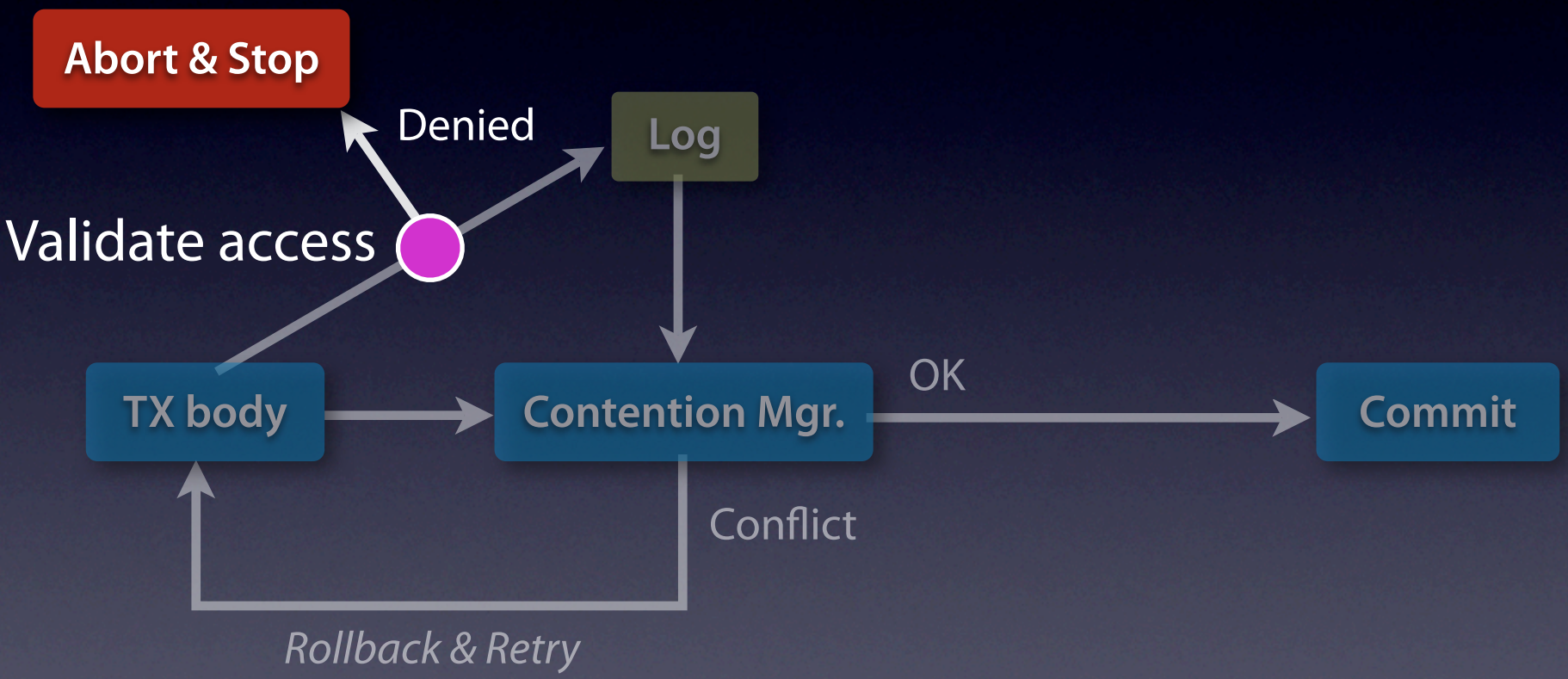
```
before commit of each transaction T {
    for (resource, op) in T.log {
        if (not allowed(T.principal, resource, op)
            abort T;
    }
}
```

# Variants of authorization managers



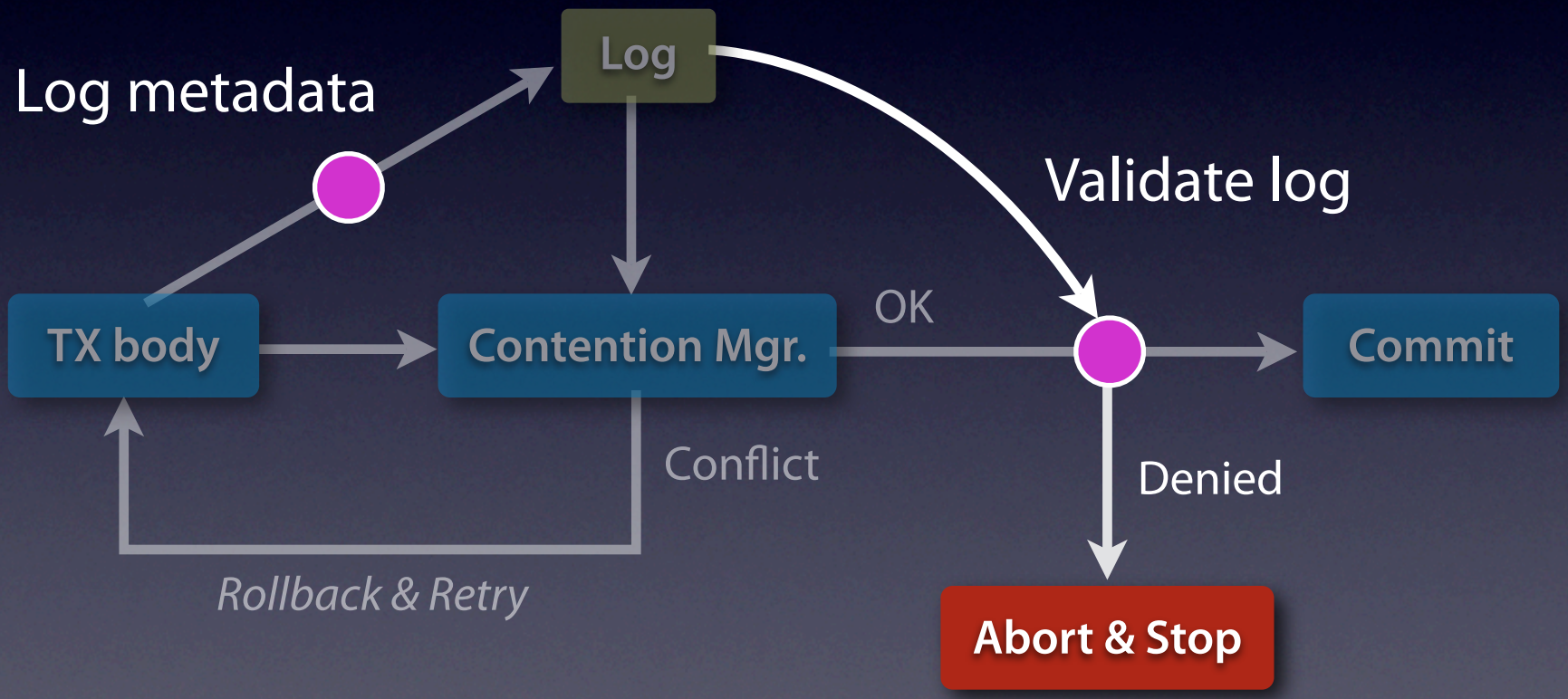
# Variants of authorization managers

***Eager***



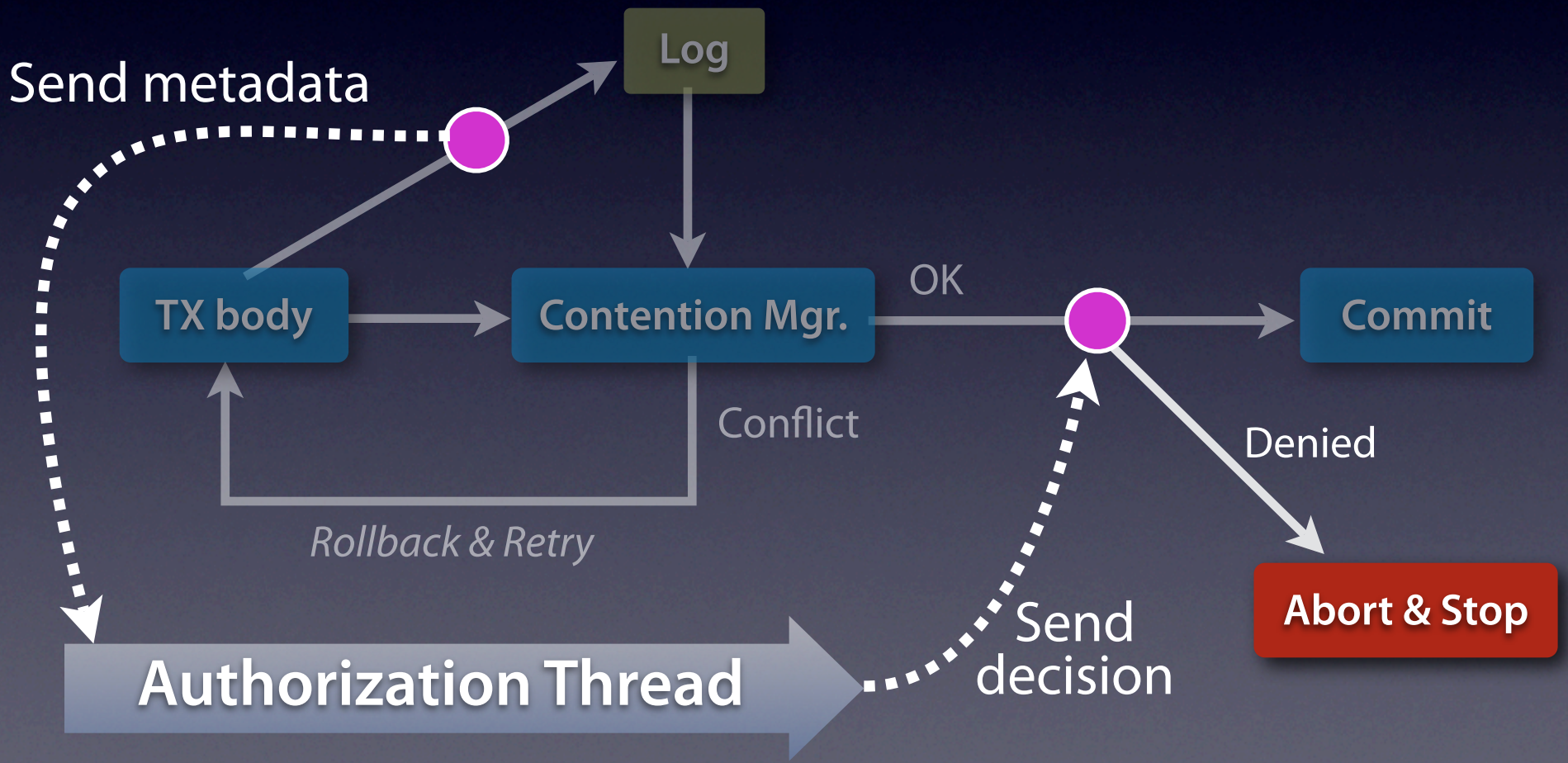
# Variants of authorization managers

***Lazy***



# Variants of authorization managers

***Overlapped***



# Implementation

- Extends Sun's DSTM2 library for Java
  [Herlihy, Luchango and Moir OOPSLA'06]
- Programmer specifies *security metadata*
- Pluggable authorization mgr. receives metadata
- Eager, lazy, overlapped or custom authoriz. mgr.
- Adds less than 500 LOC to DSTM2

# Evaluation

GradeSheet: 900 LOC, simple enforcement code,
1 atomic block

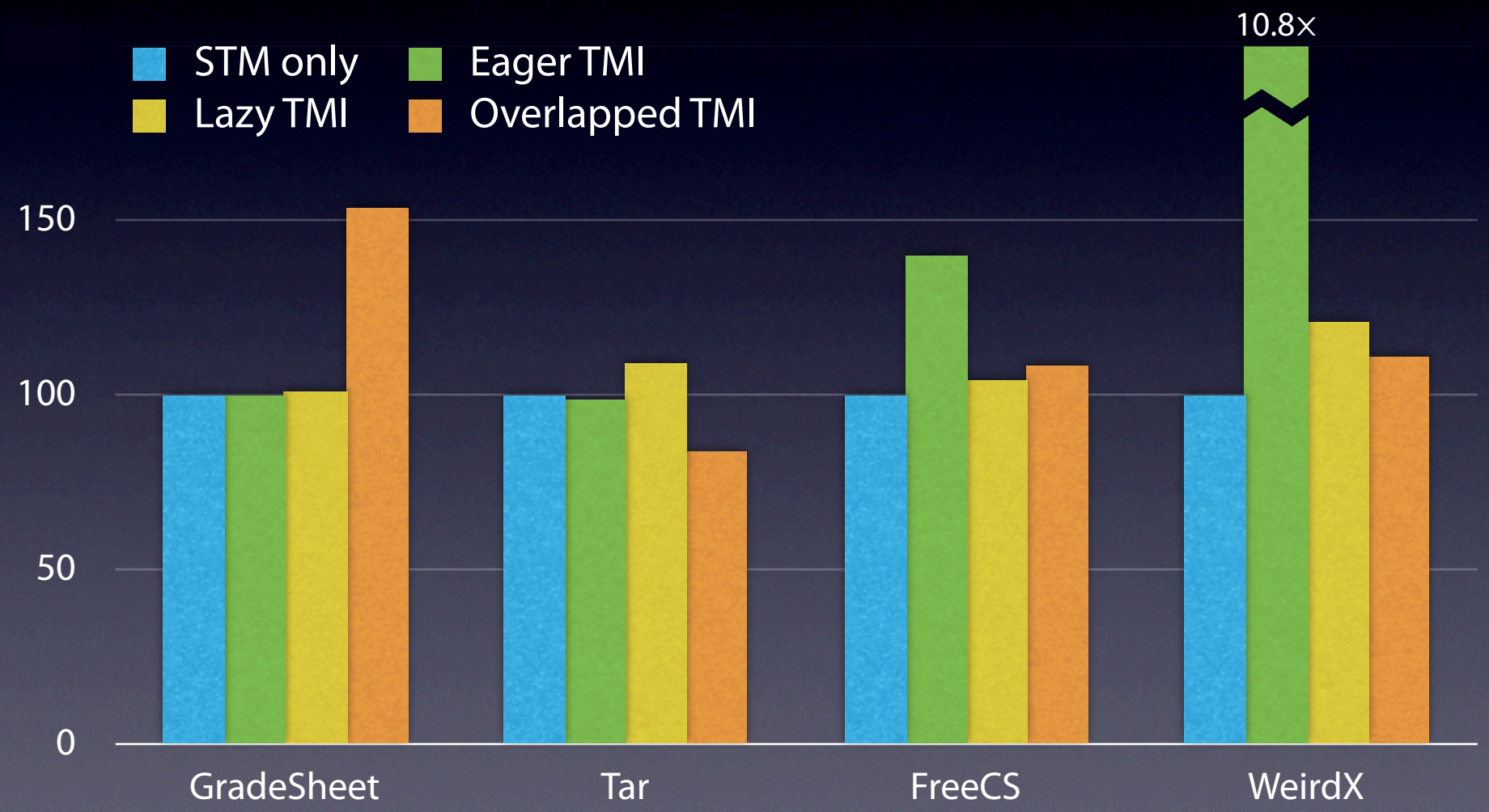
Tar: 5,000 LOC, Java Stack Inspection,
1 atomic block

FreeCS: 22,000 LOC, XACML policy enforcement,
47 atomic blocks

WeirdX: 27,000 LOC, XACML policy enforcement,
108 atomic blocks

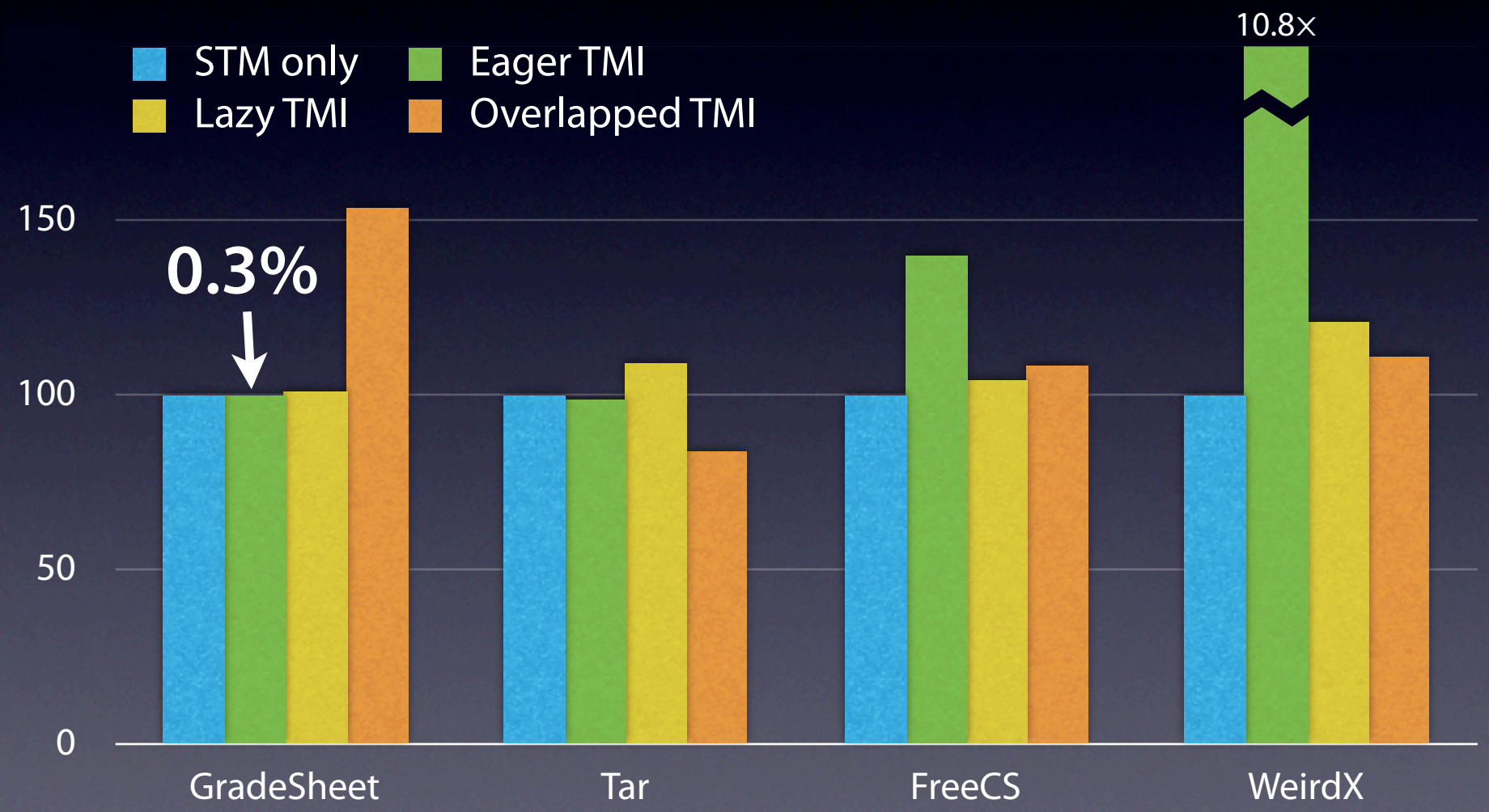
# Evaluation

## Average execution time of a request



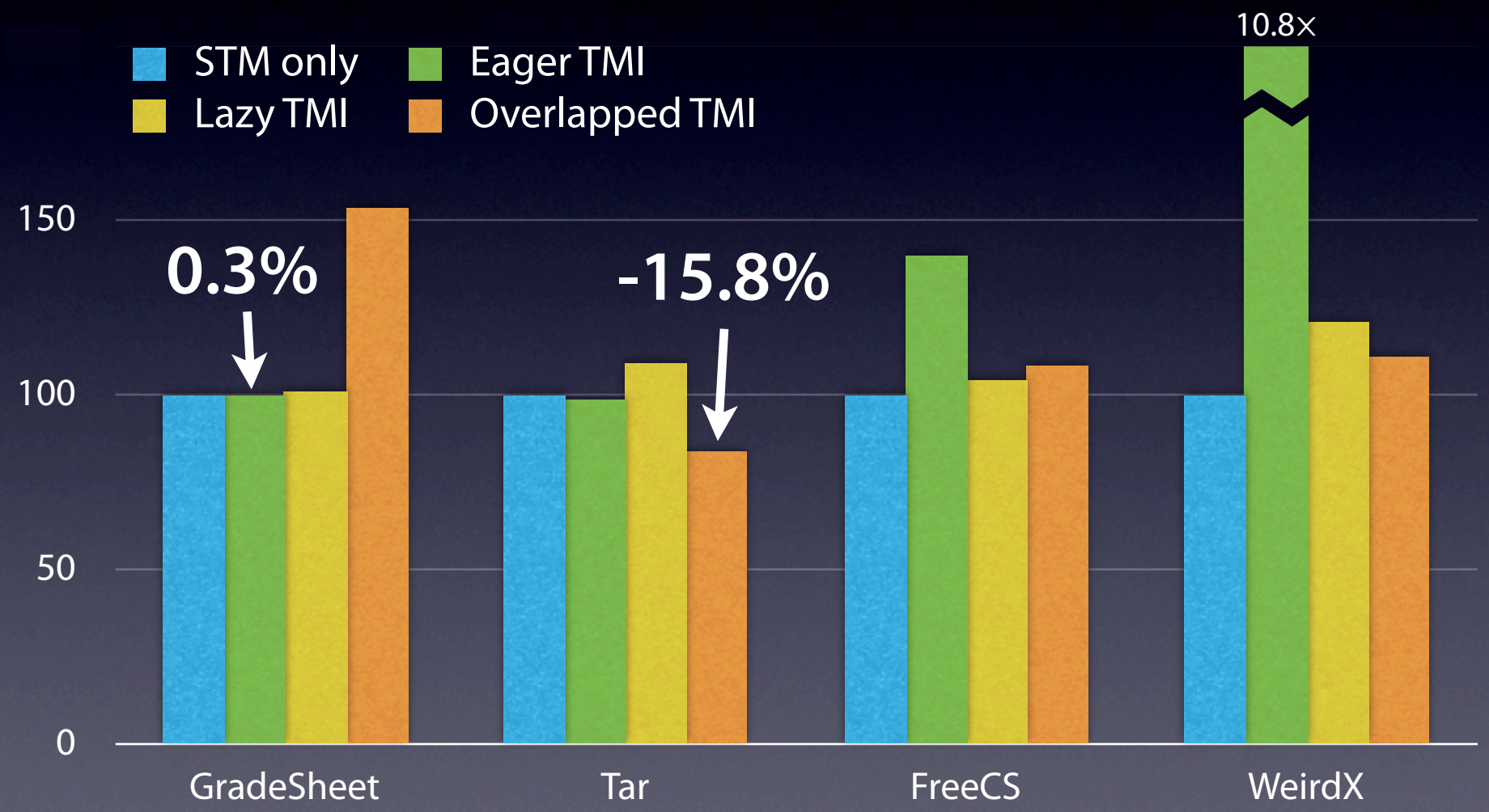
# Evaluation

## Average execution time of a request



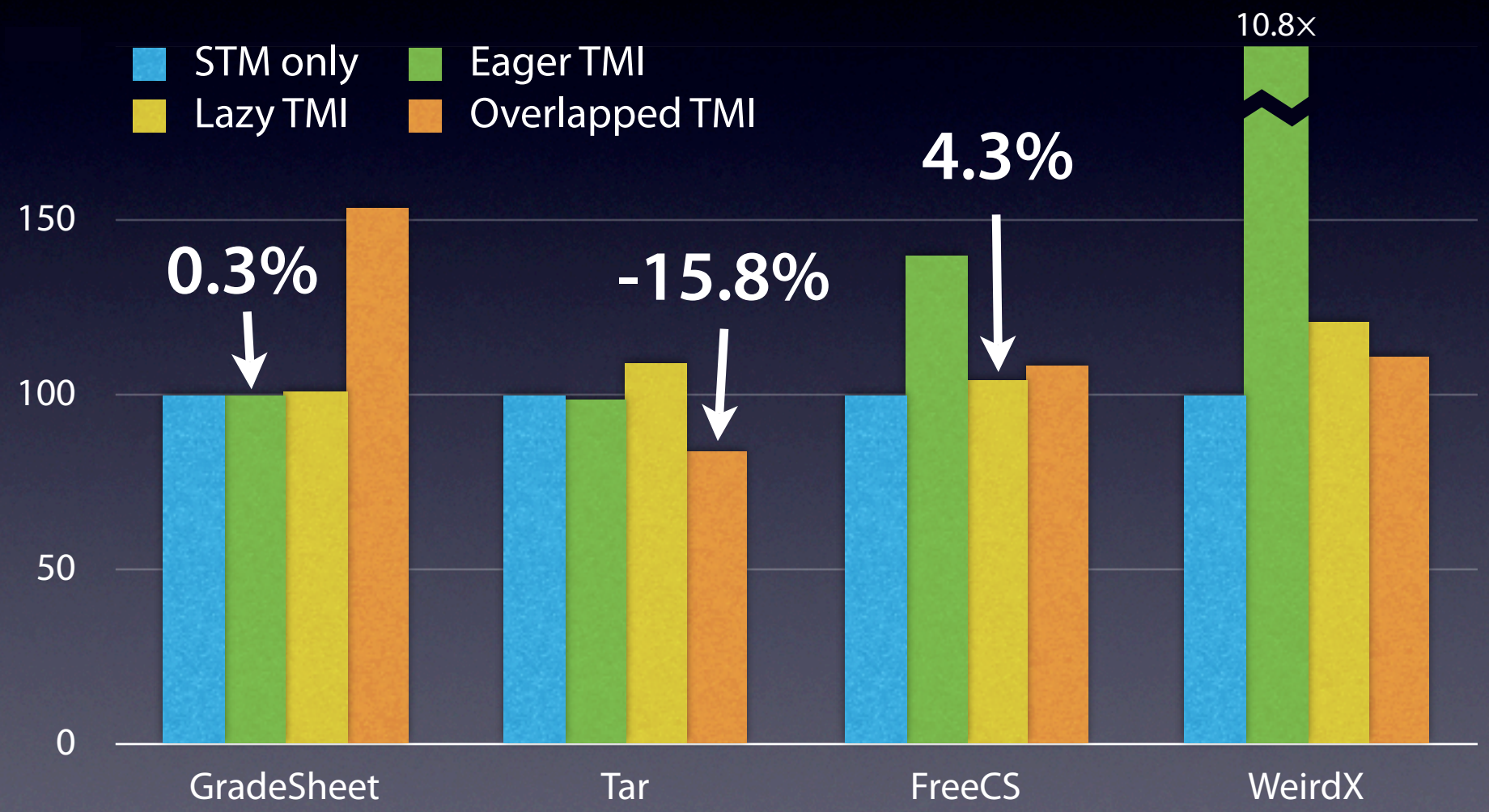
# Evaluation

## Average execution time of a request



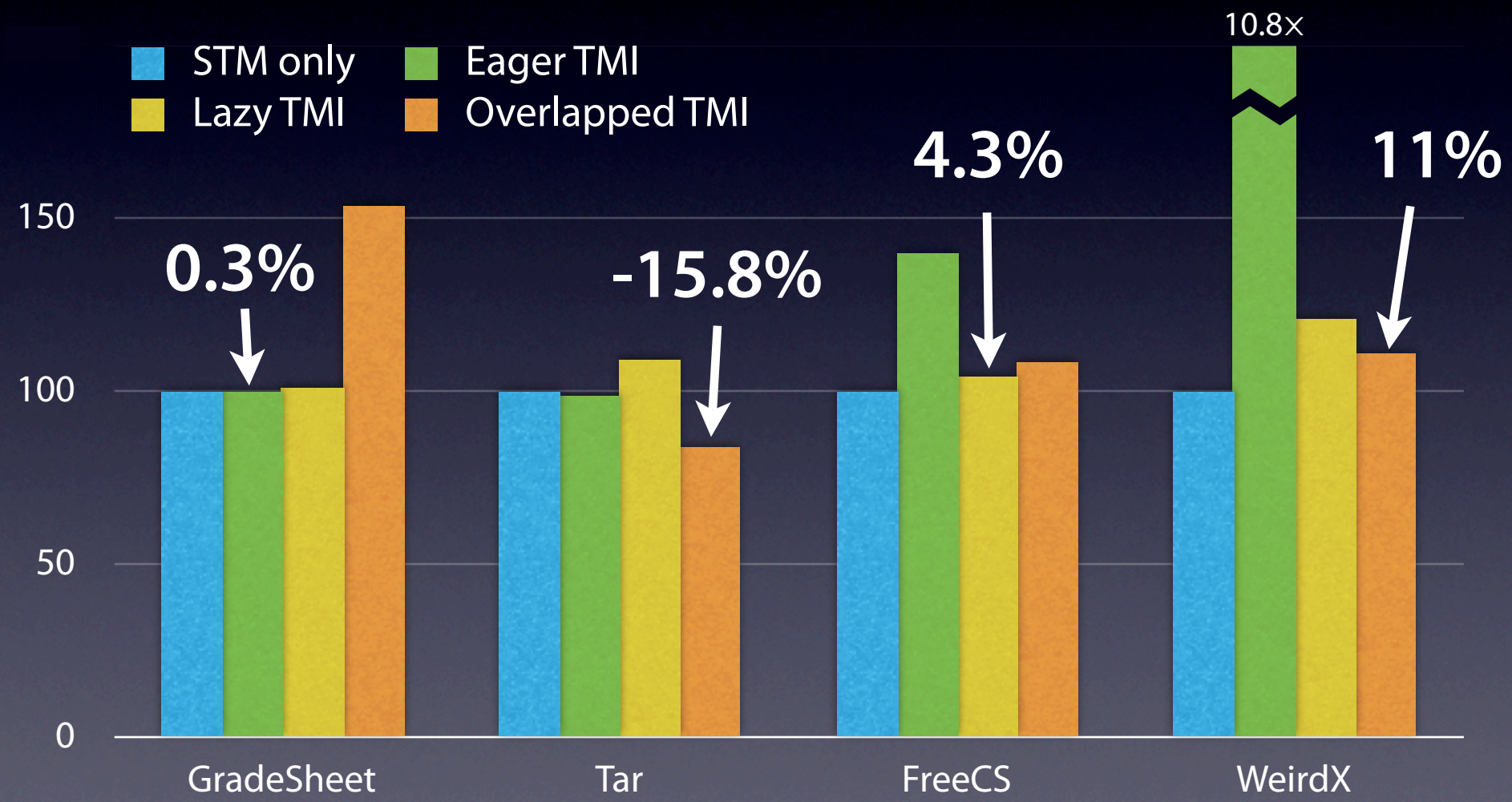
# Evaluation

## Average execution time of a request



# Evaluation

## Average execution time of a request



# Summary

- TMI is a new reference monitor architecture
- Decouples application logic from policy enforcement
- Freedom from TOCTTOU bugs
- Easier handling of authorization failures
- Easier to ensure complete mediation

# Take home slide

TMI utilizes the mechanisms of
Software Transactional Memory
to greatly improve
security policy enforcement

# Take home slide

TMI utilizes the mechanisms of
Software Transactional Memory
to greatly improve
security policy enforcement

***Thank you for your attention!***