MODIST: Transparent Model Checking of Unmodified Distributed Systems

Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu,
Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long,
Lintao Zhang, Lidong Zhou

Columbia University, Microsoft Research Asia,
Mirosoft Research Silicon Valley, Tsinghua University

Distributed system: hard to get right

- Complicated protocol + code
  - Node has no centralized view of entire system
  - Must correctly handle a large number of failures
    - Link failure, message delay, machine crash
  - Getting worse: larger scale, failures more likely
- Randomized testing
  - Low coverage
  - Non-deterministic

MODIST summary

- MOdel checker for DISTributed systems
  - Comprehensive: check many corner cases
  - "In-situ:" check unmodified, real implementations
  - Deterministic: detected errors can be replayed

- Results
  - Checked Berkeley DB replication, Paxos-MPS (managing Microsoft production data centers) [D3S, NSDI08], and PacificA [MSR-TR]
  - 35 bugs, 31 confirmed
  - 10 protocol bugs, found in every system checked

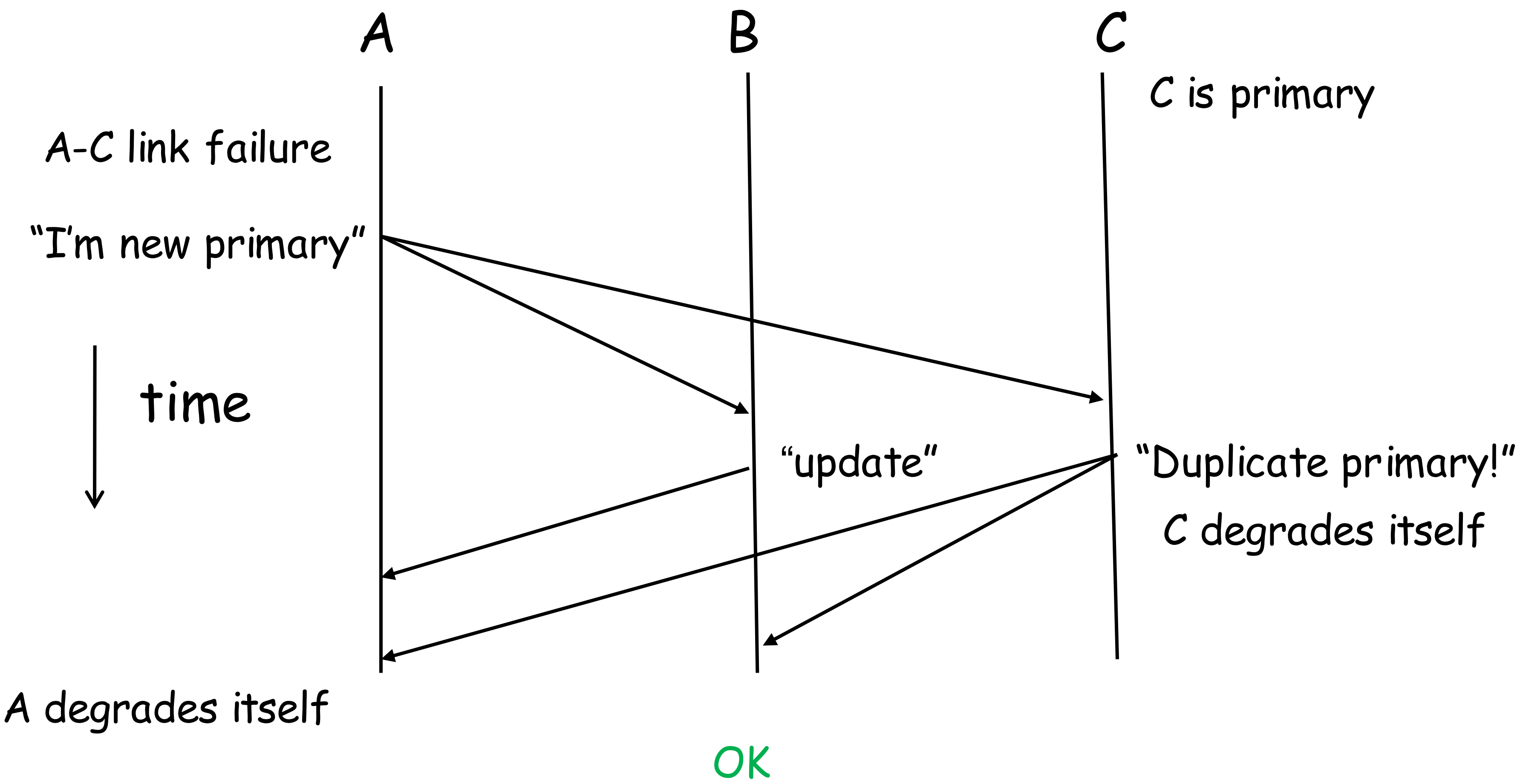
Outline

- Overview
  - Real Berkeley DB bug
  - How MODIST finds the bug
- Implementation challenges
- Errors

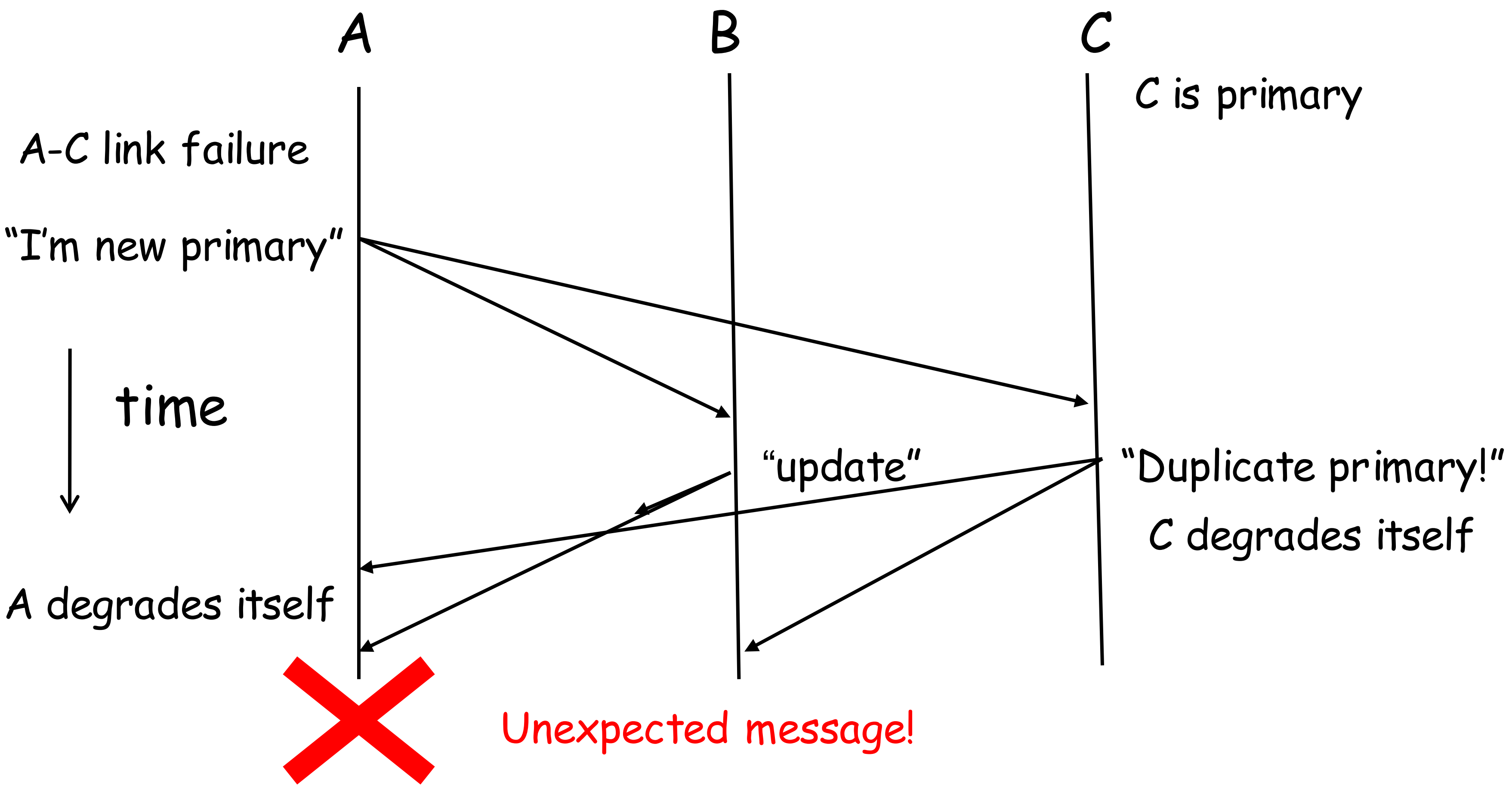
Berkeley DB replication

- Based on Paxos
  - single primary, multiple secondaries
  - Primary can read and write
  - Secondary can only read
- When primary fails, secondaries can elect new primary
- When duplicate primary detected, degrade both and re-elect
- Bug is in leader election protocol

A real Berkeley DB bug



A real Berkeley DB bug



MODIST: simple to use

$ cat bdb.conf

```
# command                                      # working dir    # inject failure?
  ex_rep_mgr.exe –n 3 –m localhost:8000 …        ./node1              1
  ex_rep_mgr.exe –n 3 –m localhost:8001 …        ./node2              1
  ex_rep_mgr.exe –n 3 –m localhost:8002 …        ./node3              1
```

$ modist.exe bdb.conf

```
spawning process 1: ex_rep_mgr.exe …
…
fail link from process 1 to process 3
…
process 3 send to process 1
…
restarting
spawning process 1: ex_rep_mgr.exe
…
```

$ modist.exe bdb.conf –r traces/0/trace

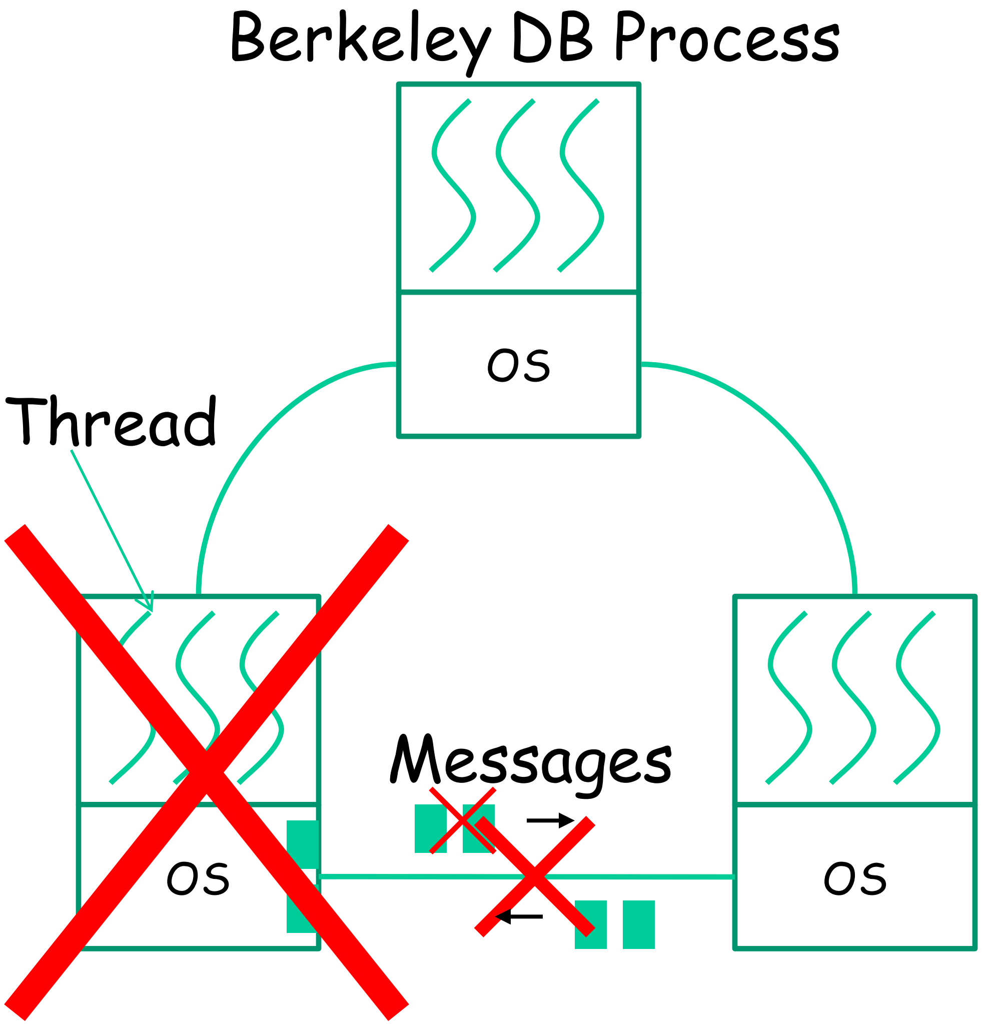
Outline

- Overview
  - Real Berkeley DB bug
  - How MODIST finds the bug
- Implementation challenges
- Errors

Core model checking idea

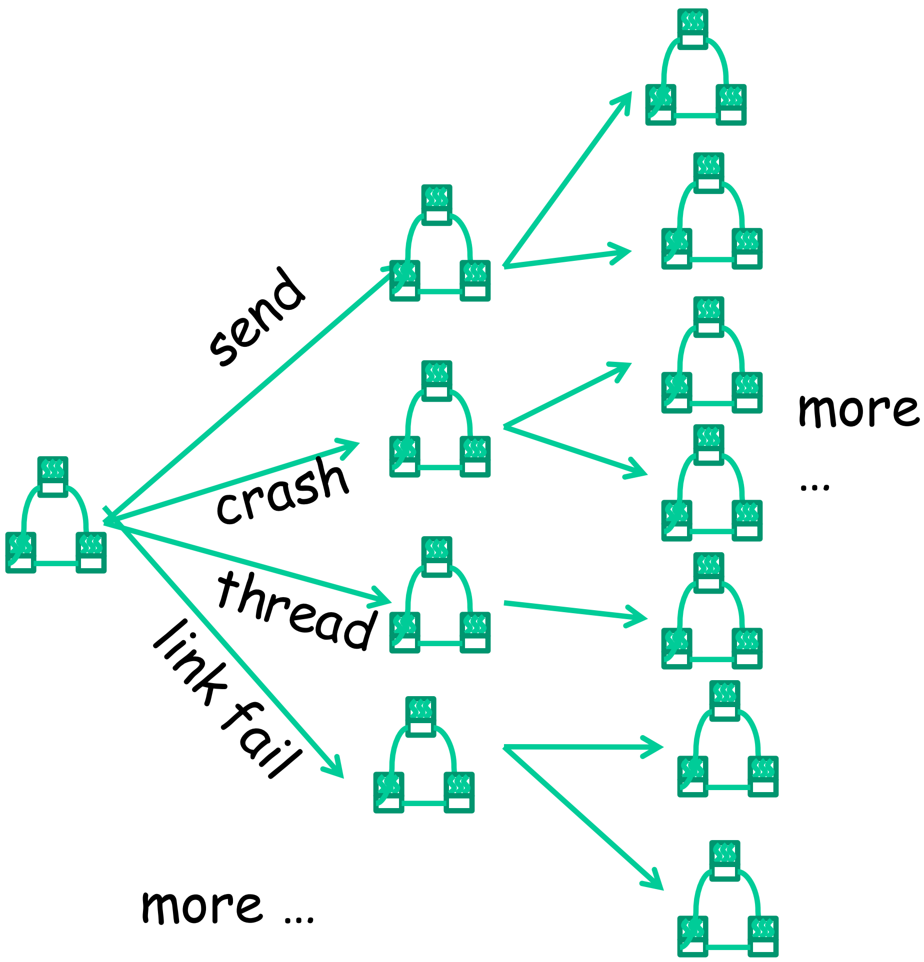
- Goal: explore all states and actions
- Advantage: rare actions appear as often as common ones, thereby quickly driving system into corner case for errors

Actions in Berkeley DB replication



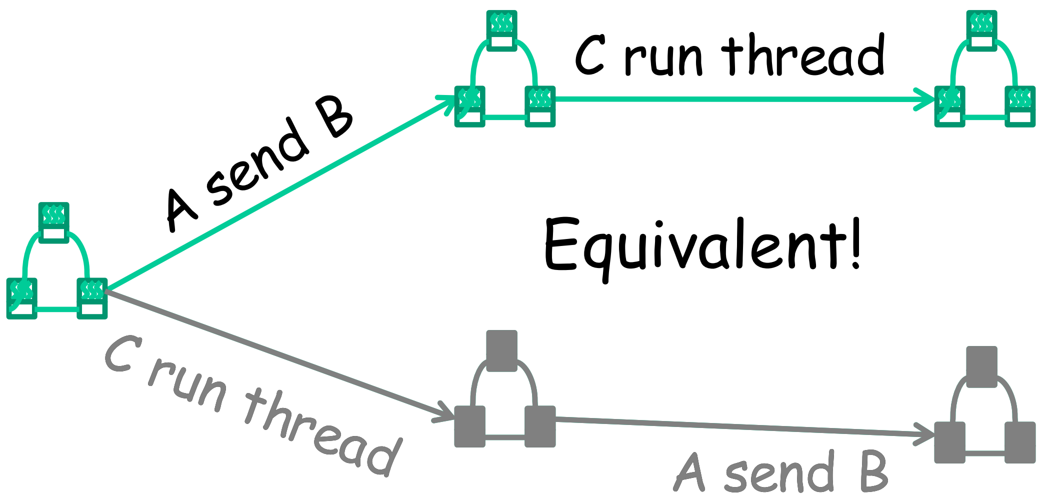
- Normal actions
  - Send message
  - Recv message
  - Run thread
  - …
- Rare actions
  - Delay message
  - Fail link
  - Crash machine
  - …

Ideal: exploring all actions



- Built-in checks
  - Crash
  - Deadlocks
  - Infinite loops
- User-written checks
  - Local assertions
  - Global assertions
    - [D3S, NSDI 08]
- MODIST amplifies

Avoiding redundancy



- Explore only one interleaving of independent actions
  - Partial order reduction [Verisoft, POPL97] [DPOR, PLDI05]
  - Our implementation handles both message passing and thread synchronizations

Outline

- Overview
  - Berkeley DB bug example
  - How MODIST finds the bug
- Implementation challenges
- Errors

Challenges

- How to expose actions?
- How to check often-untested timeout code?
- How to simulate failures?
  - Must be realistic to avoid false positives
- How to schedule actions?
  - Must be deterministic for error replay
    - E.g., asynchronous IO
  - Must avoid deadlocks
  - Must be extensible

Challenges

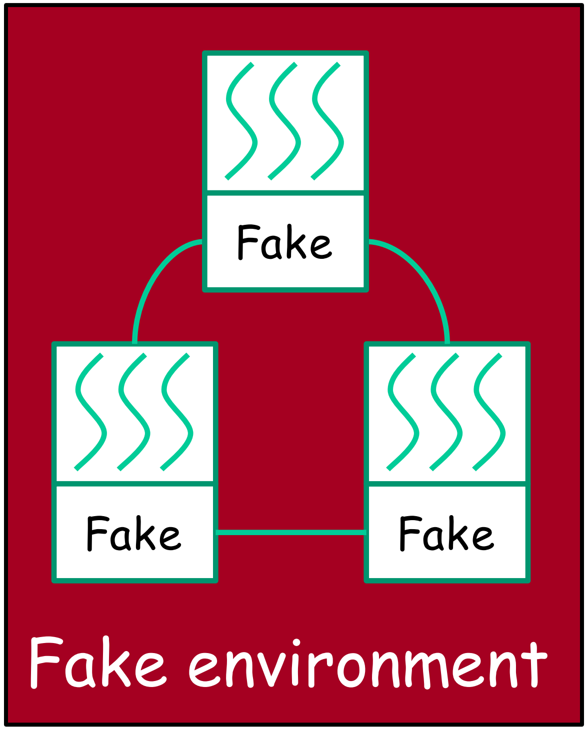
- How to expose actions?
- How to check often-untested timeout code?
- How to simulate failures?
  - Must be realistic to avoid false positives
- How to schedule actions?
  - Must be deterministic for error replay
    - E.g., asynchronous IO
  - Must avoid deadlocks
  - Must be extensible

Exposing actions

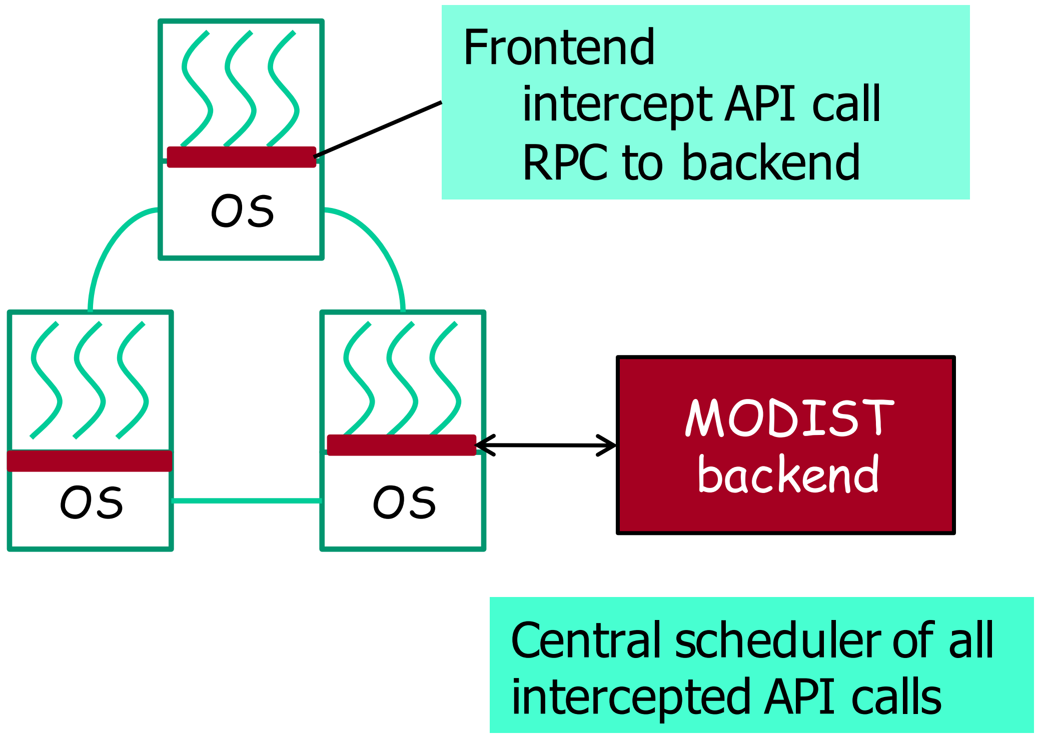
- To check, must know and control actions
- Previous work on distribute system model checking: users must expose actions
  - MaceMC: write app in special language
  - CMC: port app into fake environment
    - We used it to check FS [FiSC, OSDI06]
    - Difficult to check new app, OS
- MODIST uses in-situ checking architecture [EXPLODE, OSDI06]: interlace control needed into checked system

Architecture comparison

Traditional approach



MODIST



- Transparent
- Easy to port to new OS

Frontend: simple

- Intercepted 82 API functions
  - E.g., networking, thread synchronization
- Most wrappers are simple: return failure or call real API function
  - No need to re-implement API functions
  - Average 67 lines per wrapper

Challenges

- How to expose actions?
- How to check often-untested timeout code?
- How to simulate failures?
  - Must be realistic to avoid false positives
- How to schedule actions?
  - Must be deterministic for error replay
    - E.g., asynchronous IO
  - Must avoid deadlocks
  - Must be extensible

Checking timeouts

- System code heavily uses implicit timers

```
db_timespec now;
now = gettime();  // return current time
if (now >= t + 10 ) // timeout check
        ... // timeout handling code
else
        ... // no timeout
```

- Challenge: can intercept gettime(), but what to return?
  - Want to explore both branches
  - Must know t + 10, but no API call
  - Previous work: manual

Static symbolic analysis

- Key observations
  - Time values are used in simple ways
    - Berkeley DB: db_timespec, mostly +,-, sometimes *,/
    - ➔ Static analysis can pick up time values easily
  - Programmers check timeout soon after current time
    - Intuition: want current time to be "fresh"
    - Berkeley DB: 12 out of 13 are within a few lines
    - ➔ Track only short flows of time values
- Our solution: static intra-procedural symbolic analysis to discover implicit timers
  - Much simpler than state of art symbolic analysis [KLEE, OSDI08]

Outline

- Overview
  - Berkeley DB bug example
  - How MODIST finds the bug
- Implementation challenges
- Errors

Errors

System	KLOC	Protocol bugs	Impl. bugs	Total
Berkeley DB	172.1	2	5	7
Paxos-MPS	53.5	2	11	13
PacificA	12	6	9	15
Total	237.6	10	25	35

- Large, complex systems
- Total 35 bugs, all previously unknown, 31 confirmed
- Protocol bugs in every system, total 10

Conclusion

- MODIST: in-situ model checker for distributed systems
  - Comprehensive, transparent, deterministic
  - Effective
    - Checked Berkeledy DB, Paxos-MPS, PacificA
    - 35 bugs, 10 protocol bugs
- Real distributed protocols are buggy
  - Interestingly, based on proven-correct protocols
  - Bugs stem from concretitzation or customizations