

Detecting Manipulated Remote Call Streams

Jonathon Giffin, Somesh Jha, Barton Miller

Computer Sciences Department

University of Wisconsin

`giffin@cs.wisc.edu`

Intrusion Detection and Specification-Based Monitoring

- **The Condor attack**

How to easily do dangerous and malicious things to a running job

- **Binary analysis**

How to detect attempted intrusions with pre-execution static analysis and runtime monitoring

- **Program instrumentation**

How to improve model precision & performance

Intrusion Detection

Misuse Detection

- Specify patterns of attack or misuse
- Ensure misuse patterns do not arise at runtime
- Snort
- **Rigid: cannot adapt to novel attacks**

Specification-Based Monitoring

- Specify constraints upon program behavior
- Ensure execution does not violate specification
- Our work; Ko, et al.
- **Specifications can be cumbersome to create**

Anomaly Detection

- Learn typical behavior of application
- Variations indicate potential intrusions
- IDES
- **High false alarm rate**

Specification

Analyst or
Administrator

Training
Sets

Static
Source Code
Analysis

Static
Binary Code
Analysis

Execution
Obeys Static
Rule Set

Execution
Matches
Model of
Application

Enforcement

Our Approach: Specification

Static analysis of **binary** code

- Specifications are automatically generated
- Not reliant upon analysts to produce accurate specifications
- Analyzes all execution paths
- Source code may be unavailable

Our Approach: Enforcement

Operate an automaton modeling correct system call sequences

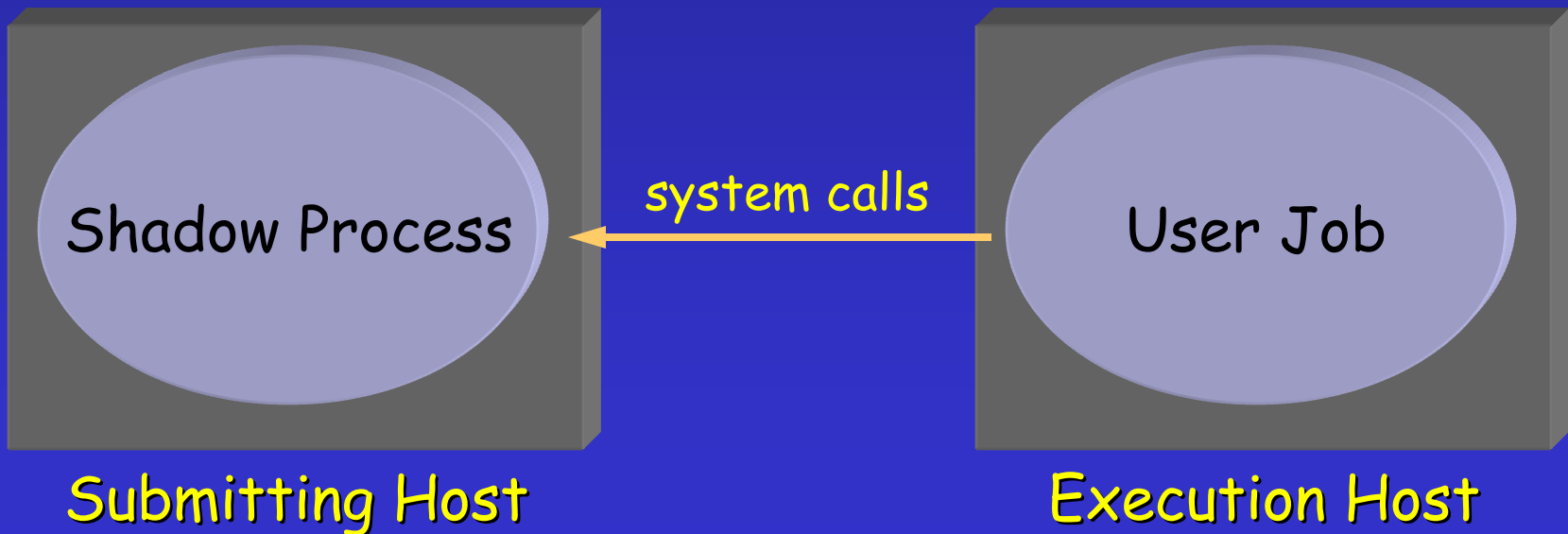
- Dynamic ruleset

Technical Contributions

- Binary analysis
- Model comparisons
- Techniques to improve precision
 - Null call insertion
 - Call site renaming
- Techniques to improve performance
 - Stack abstractions
 - Null call insertion: Practical results using push-down automaton (PDA) models

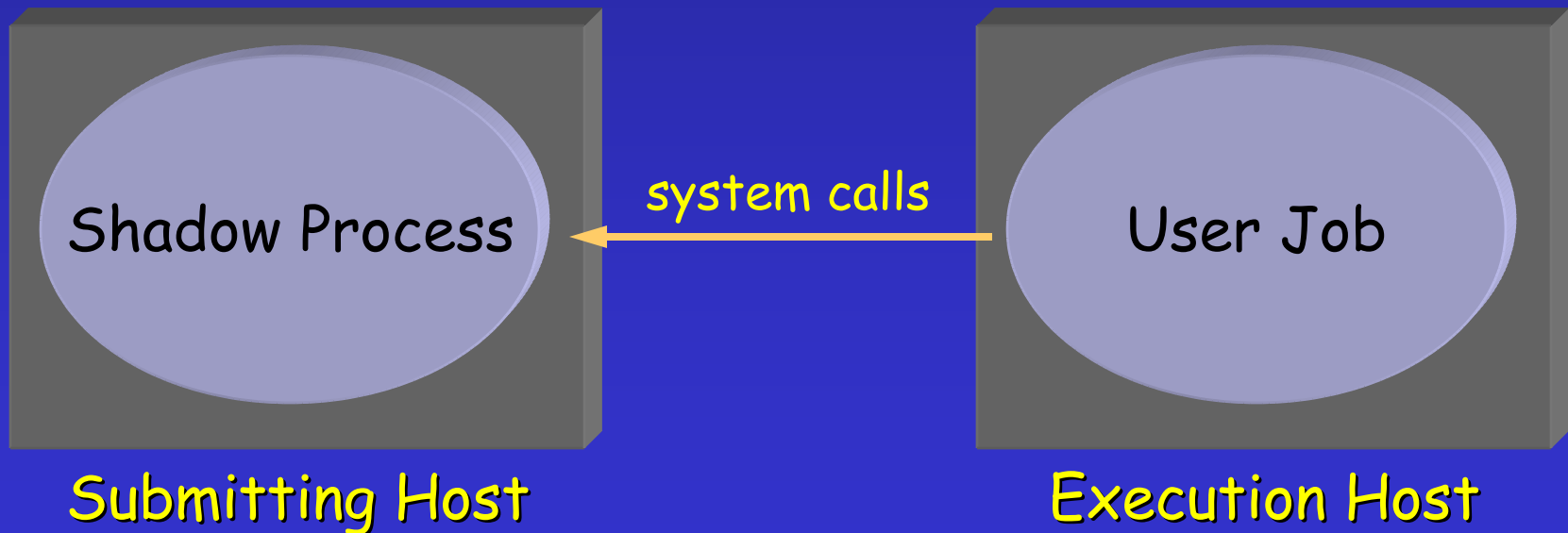
Example: The Condor Attack

- Users dispatch programs for remote execution
- Remote jobs send critical system calls back to local machine for execution



Example: The Condor Attack

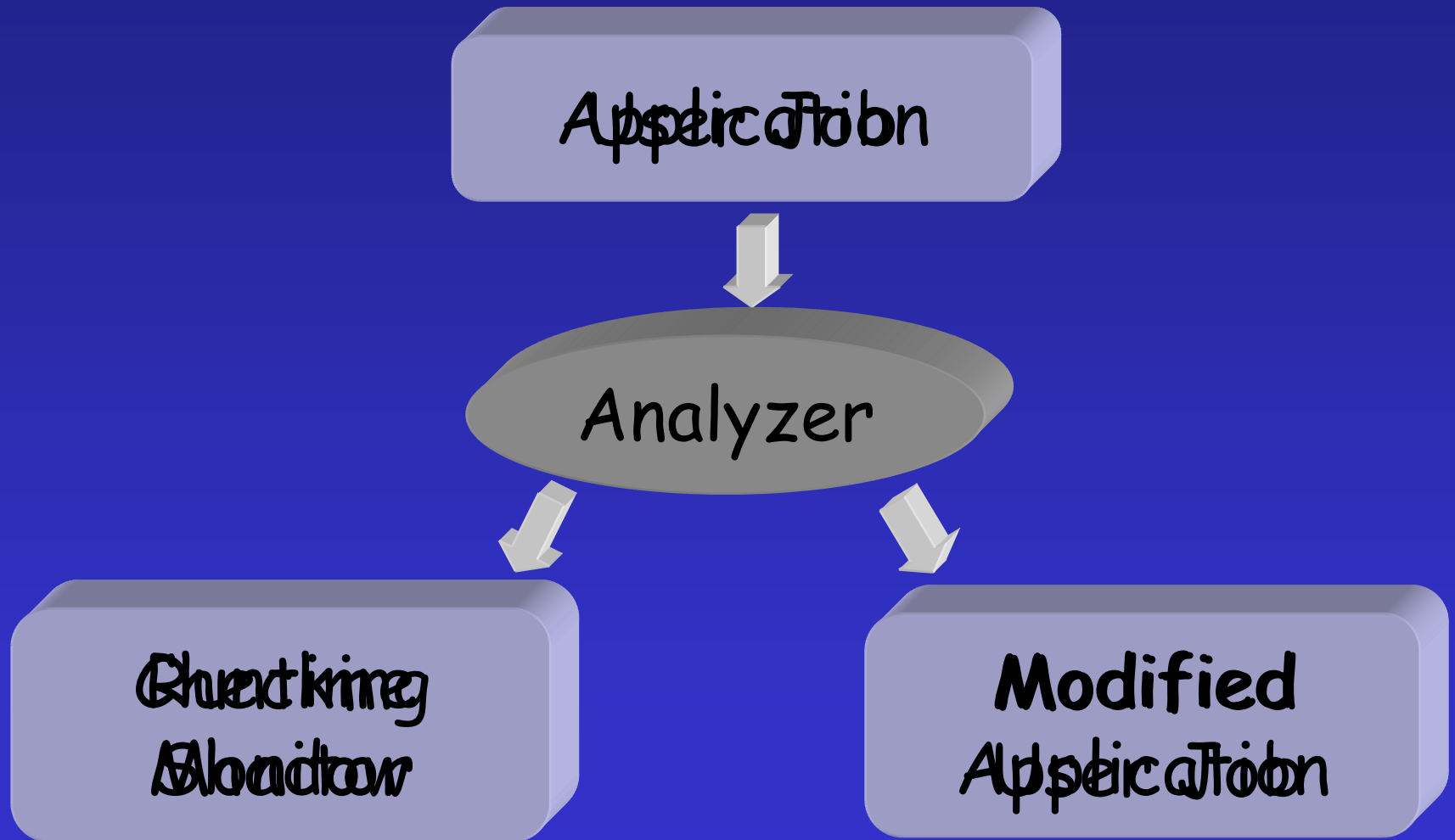
- Attackers can manipulate remotely executing program to gain access to user's machine



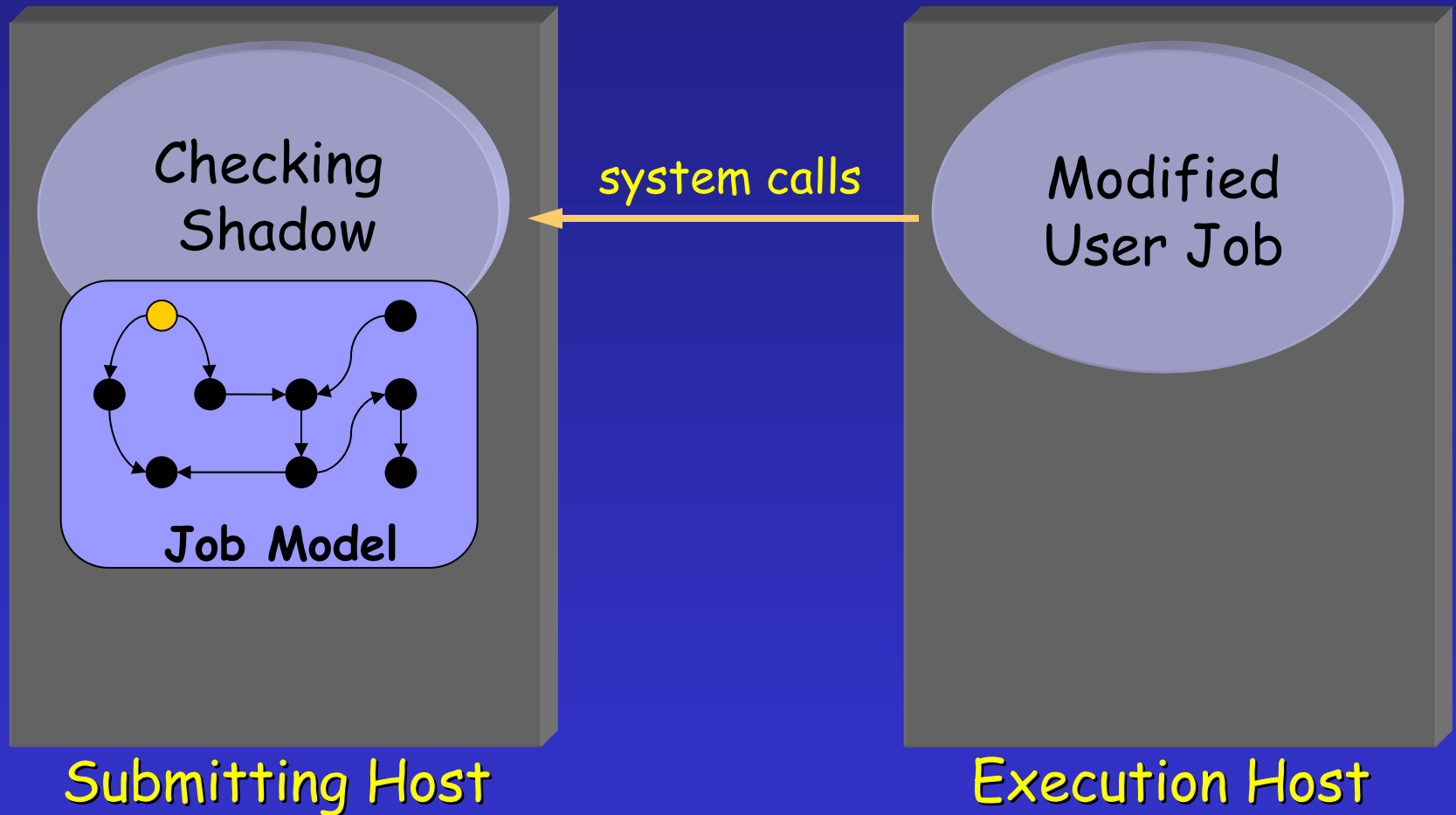
Countering Remote Attacks

- **Goal:** Even if an intruder can see, examine, and fully control the remote job, no harm can come to the local machine.
- **Method:** Model all possible sequences of remote system calls. At runtime, update the model with each received call.
- **Key technology:** Static analysis of binary code.

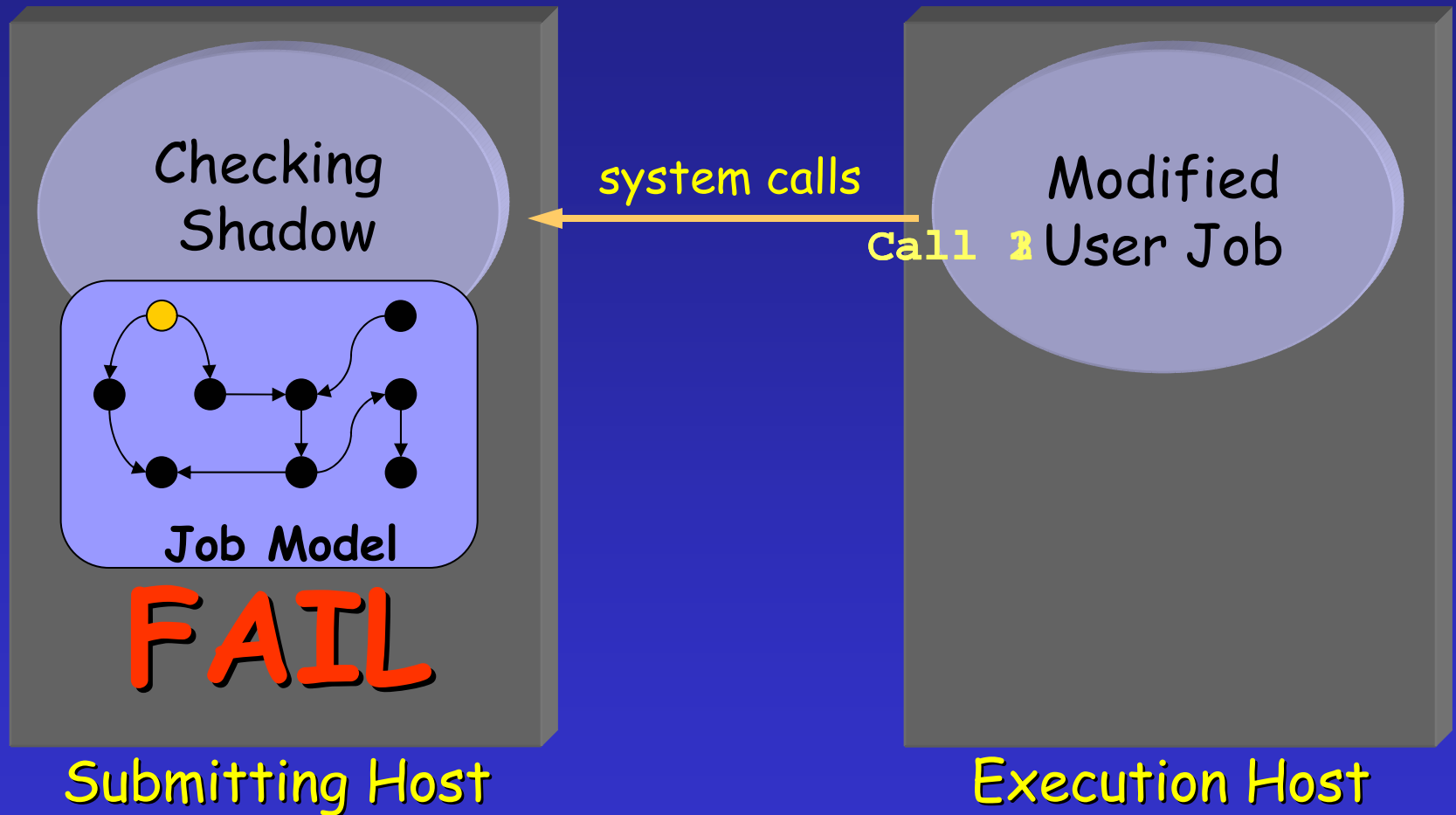
Execution Monitoring



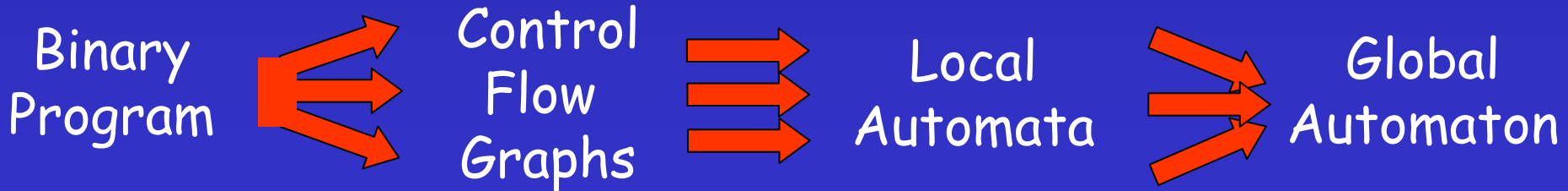
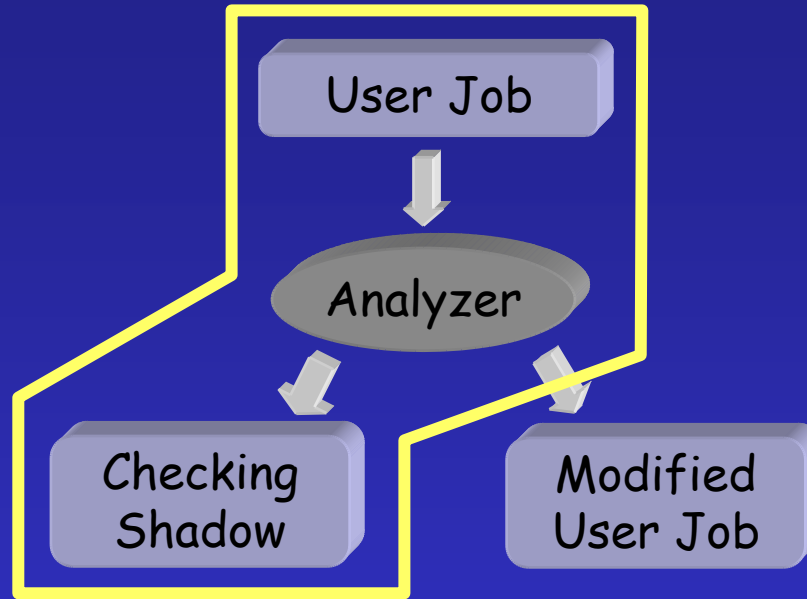
Execution Monitoring



Execution Monitoring



Model Construction

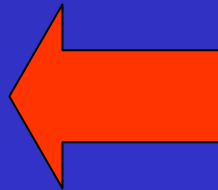
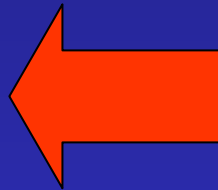


The Binary View (SPARC)

function:

```
    save %sp, 0x96, %sp
    cmp %i0, 0
    bge L1
    mov 15, %o1
    call read
    mov 0, %o0
    call line
    nop
    b L2
    nop
L1:
    call read
    mov %i0, %o0
    call close
    mov %i0, %o0
L2:
    ret
    restore
```

```
function (int a) {
    if (a < 0) {
        read(0, 15);
        line();
    } else {
        read(a, 15);
        close(a);
    }
}
```



Control Flow Graph Generation

function:

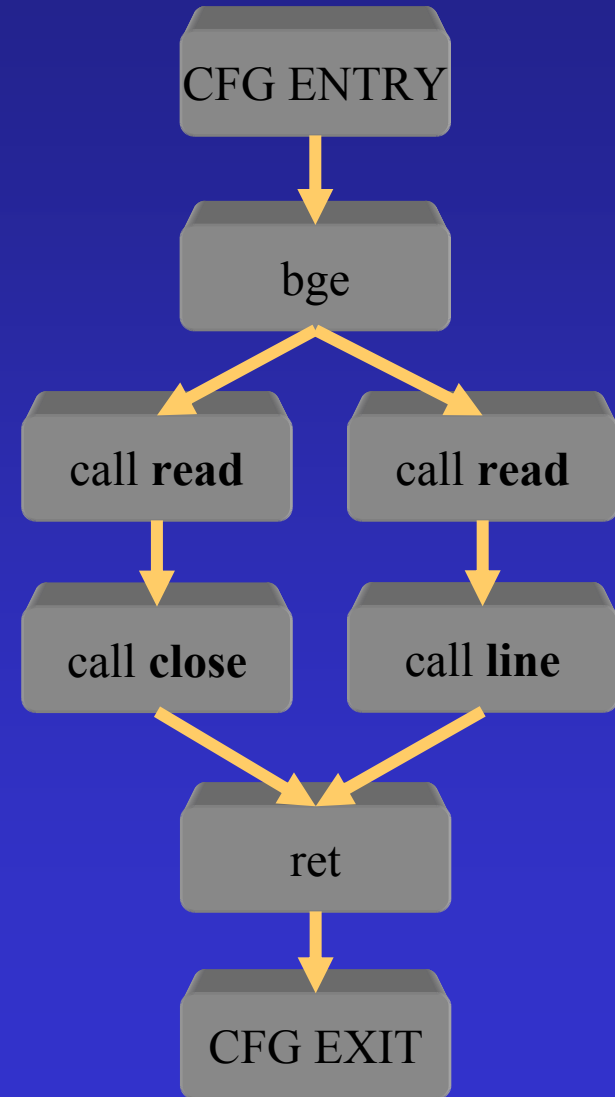
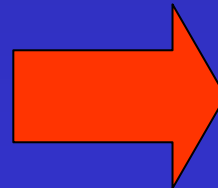
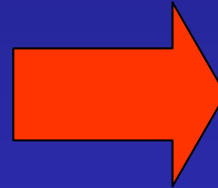
```
save %sp, 0x96, %sp
cmp %i0, 0
bge L1
mov 15, %o1
call read
mov 0, %o0
call line
nop
b L2
nop
```

L1:

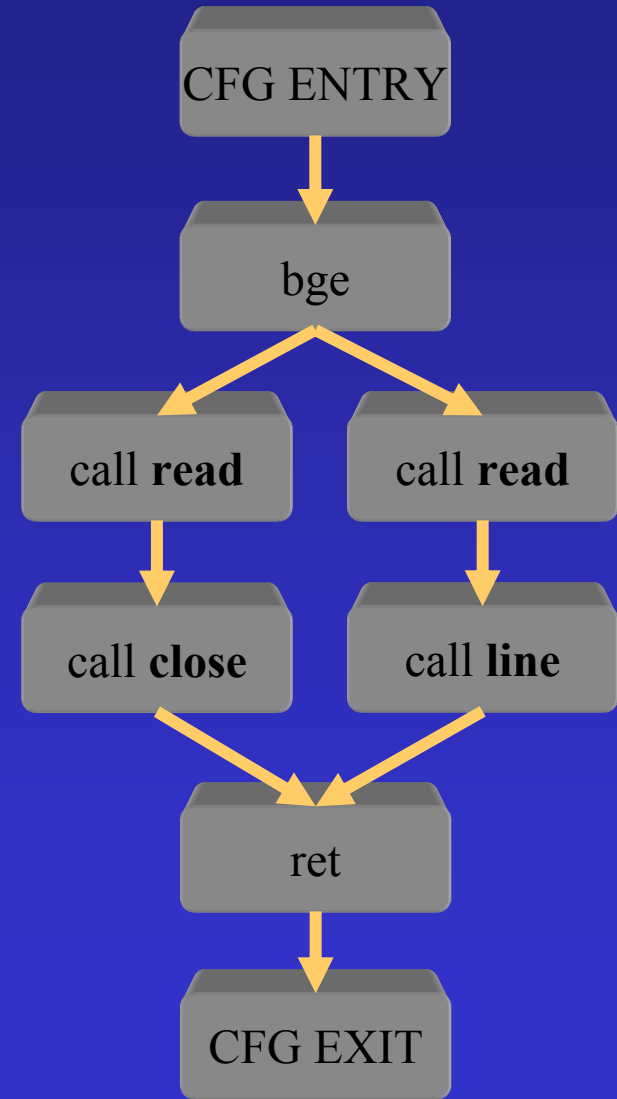
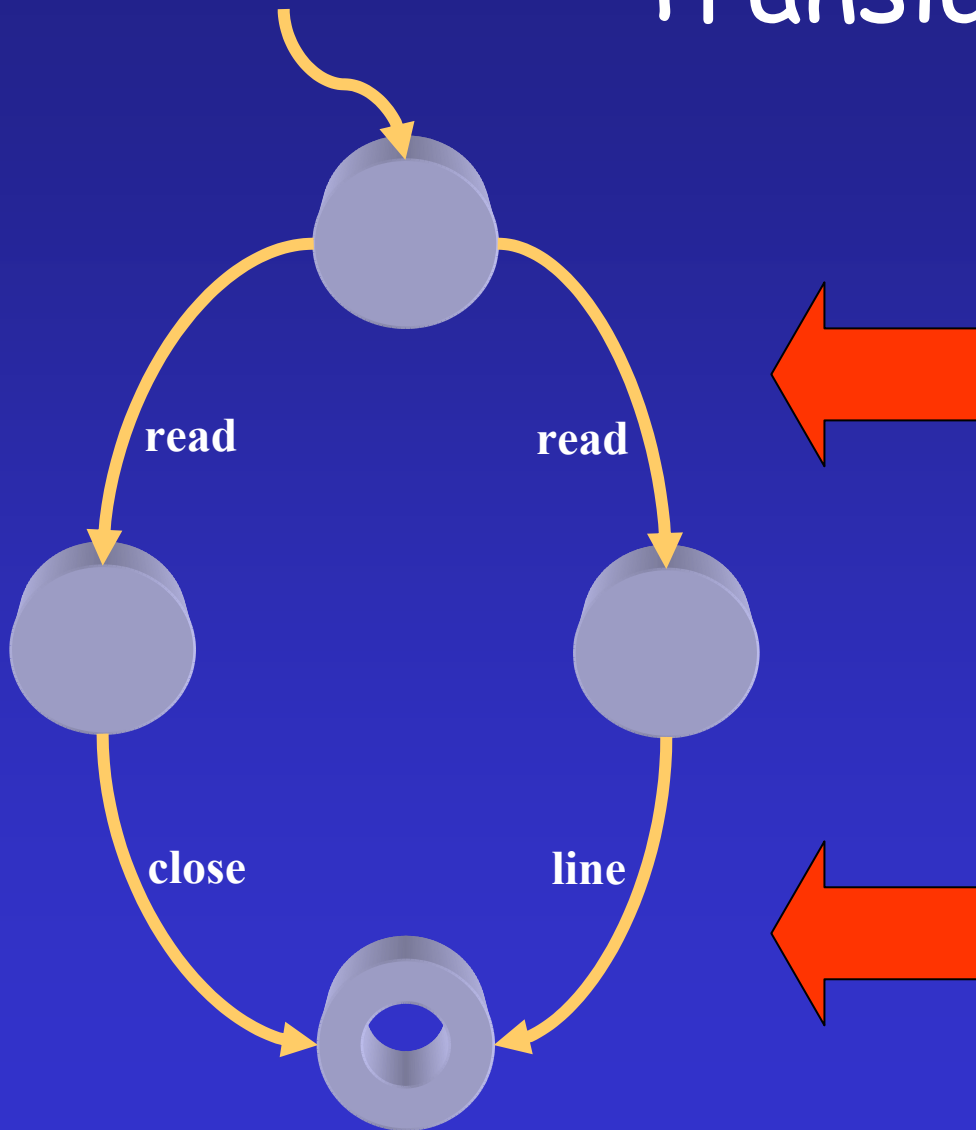
```
call read
mov %i0, %o0
call close
mov %i0, %o0
```

L2:

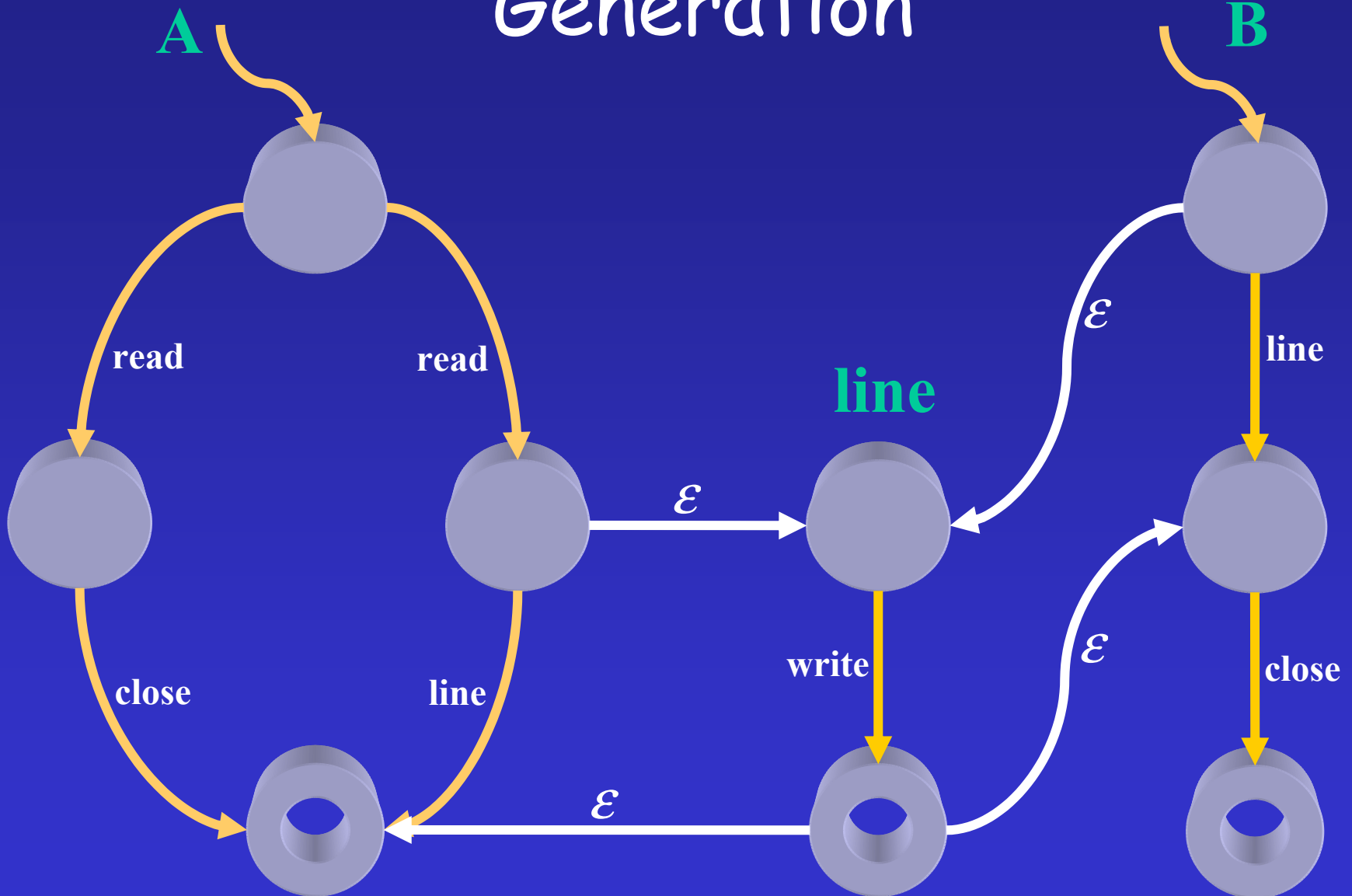
```
ret
restore
```



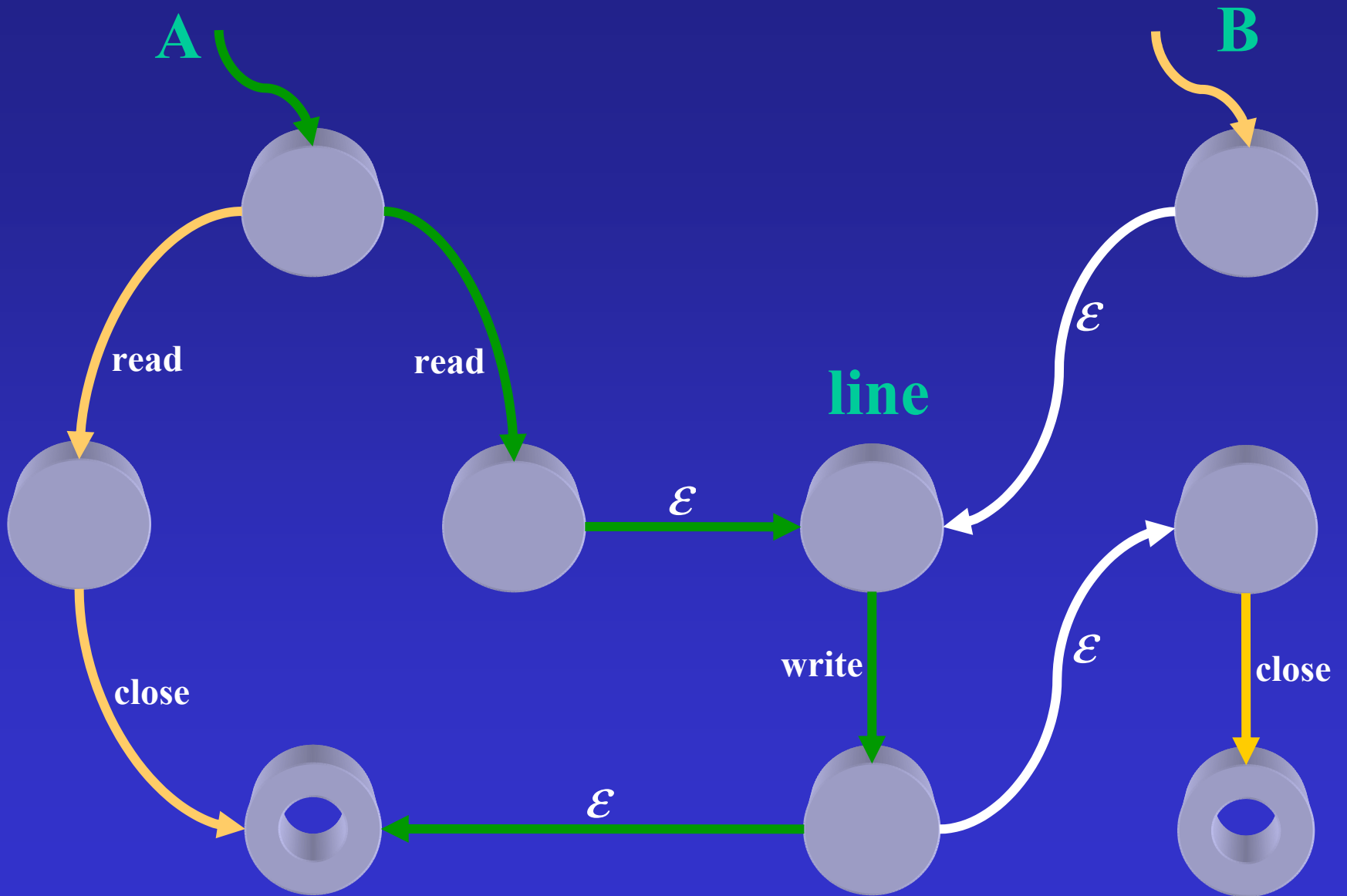
Control Flow Graph Translation



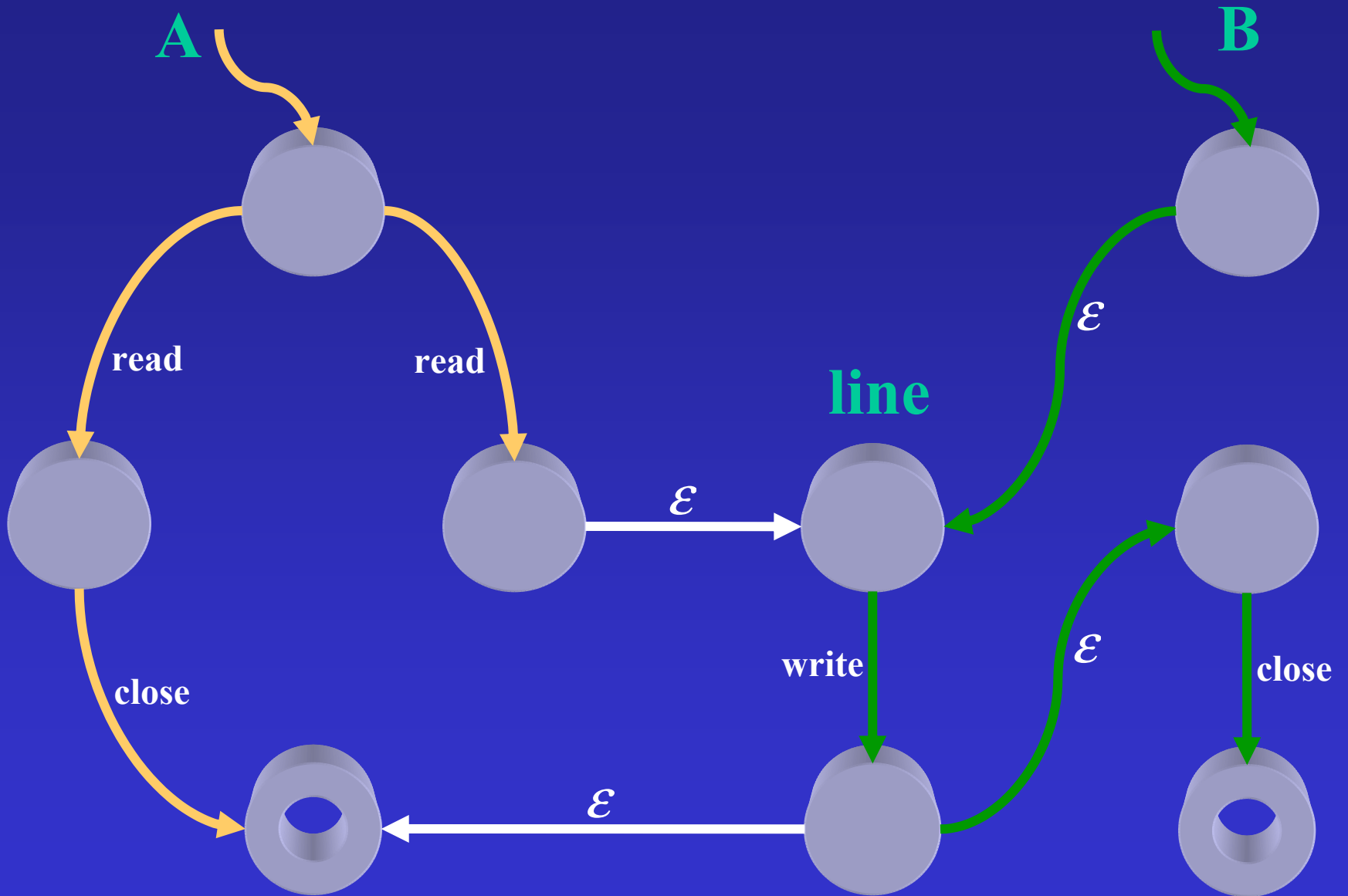
Interprocedural Model Generation



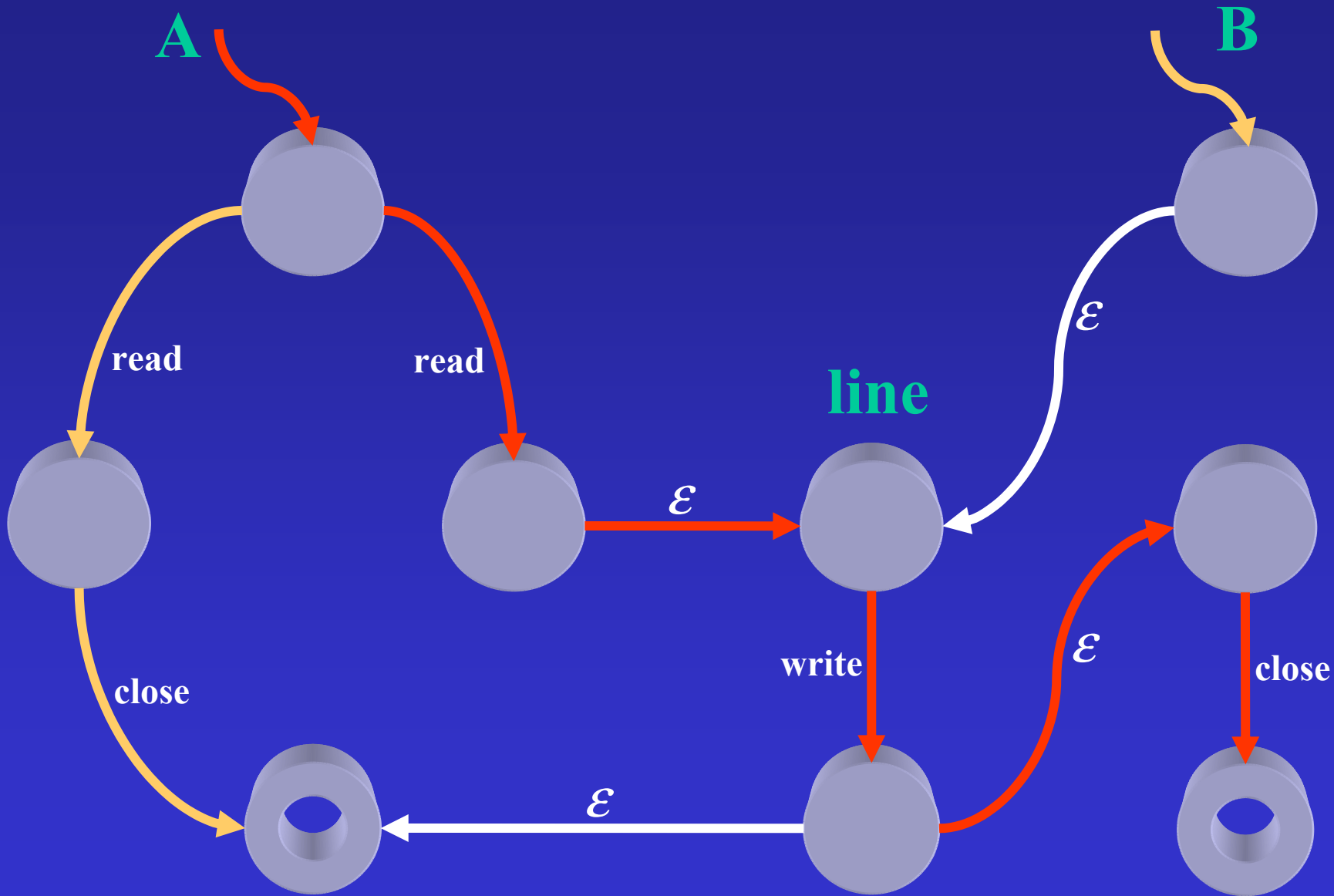
Possible Paths



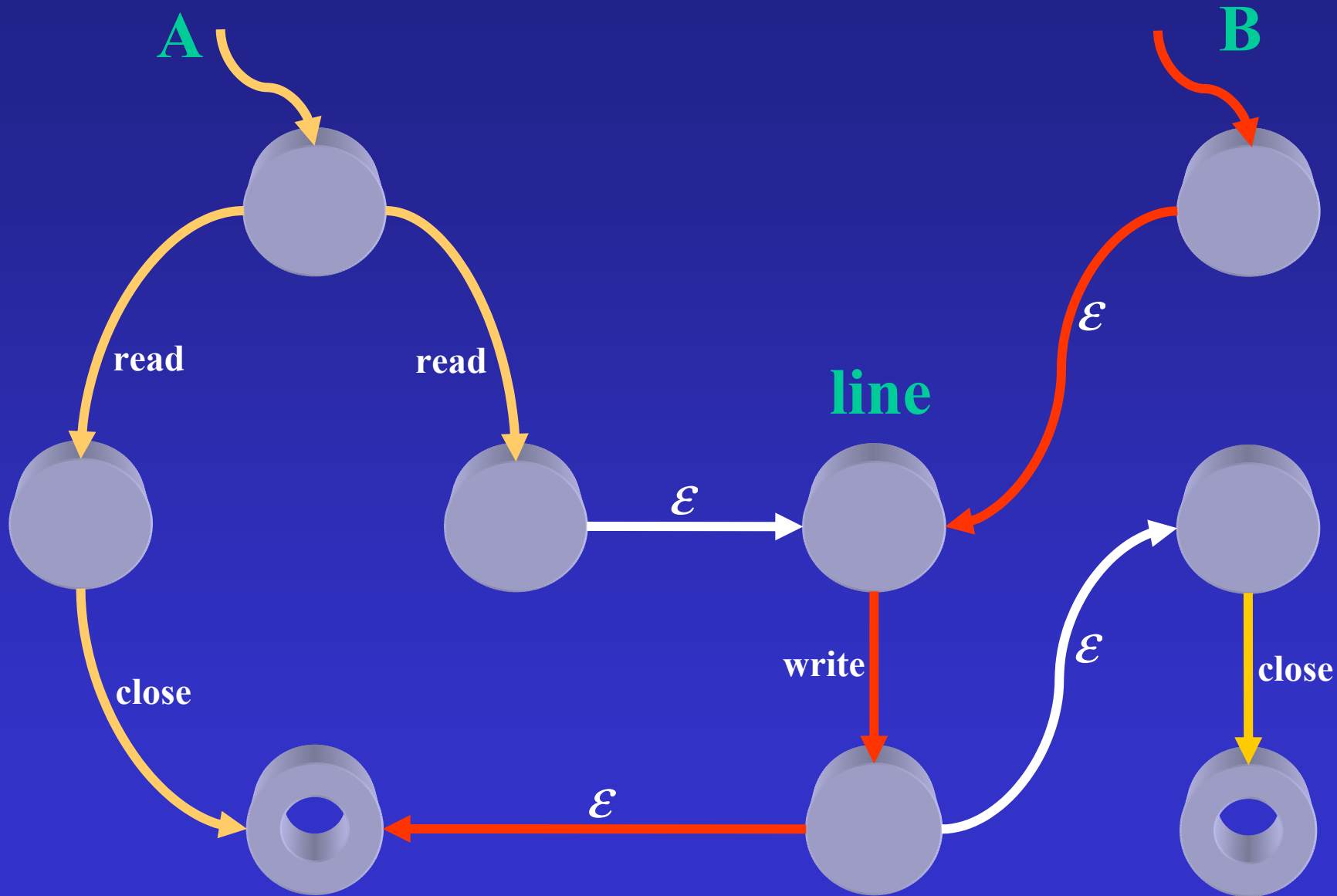
Possible Paths



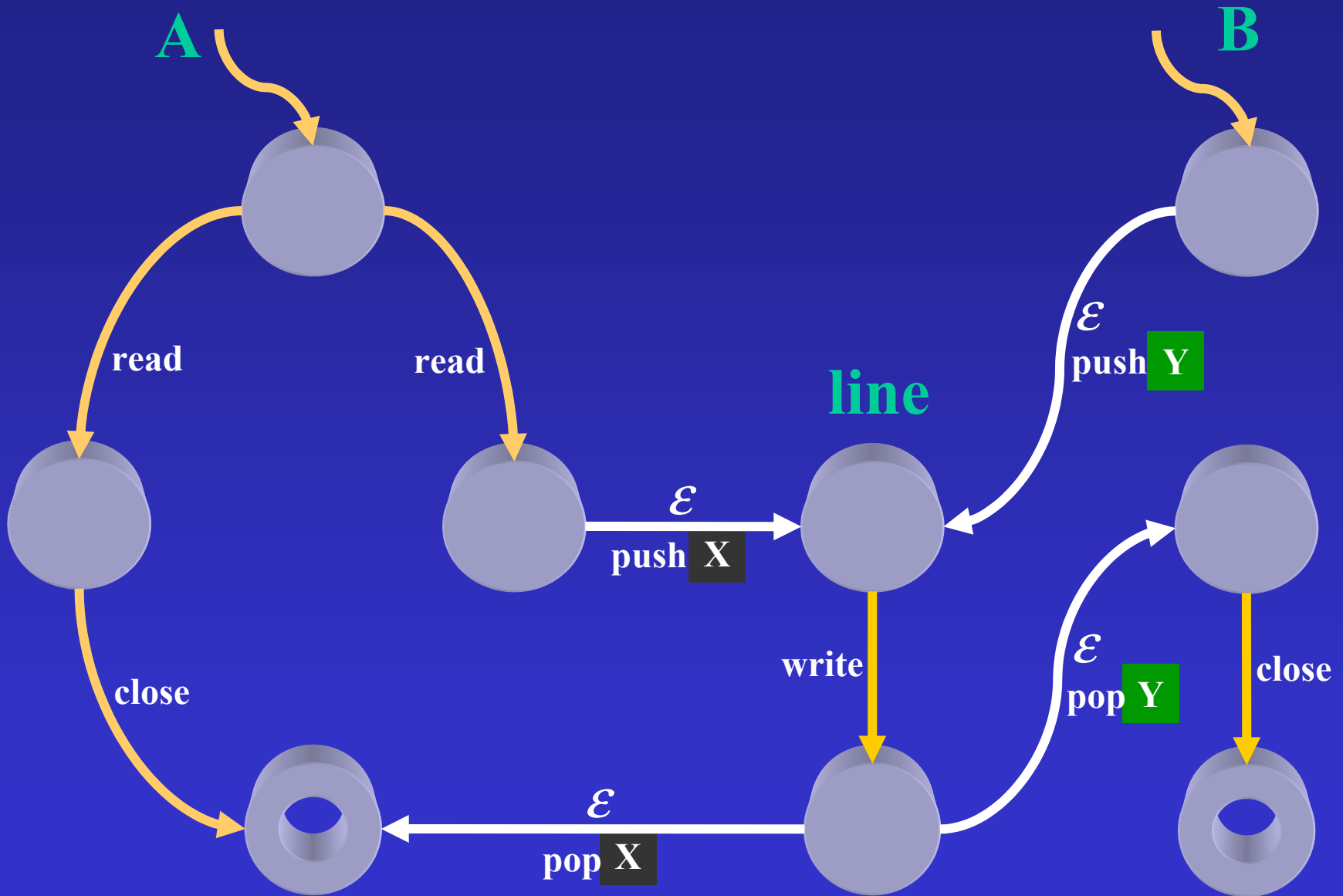
Impossible Paths



Impossible Paths

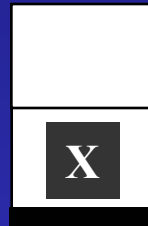


Adding Context Sensitivity



PDA State Explosion

- ϵ -edge identifiers maintained on a stack
 - Stack may grow to be unbounded
- Solution:
 - Bound the maximum size of the runtime stack
 - A regular language overapproximation of the context-free language of the PDA



Prototype Implementation

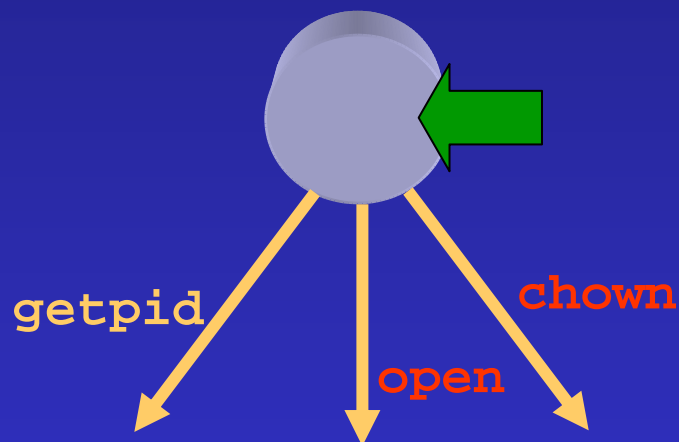
- Simulates remote execution environment
- Measure model precision
- Measure runtime overheads
- Measure the effect of changing maximum stack depth on bounded PDA model

Test Programs

	Program Size in Instructions	Workload
gzip	56,686	Compress a 13 MB file
GNU finger	95,534	Finger 3 non-local users
procmail	107,167	Process 1 incoming email message

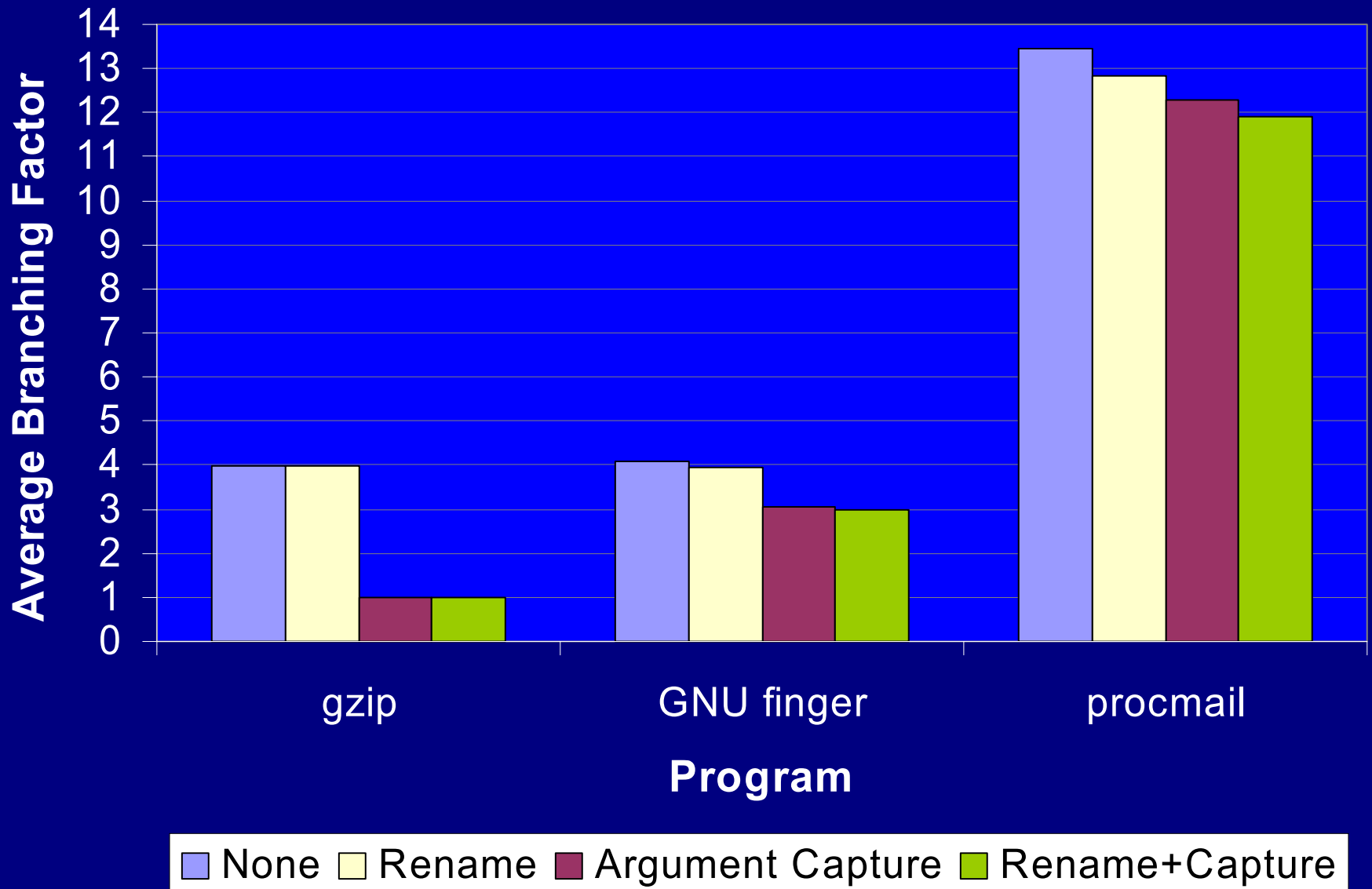
Precision Metric

- Average branching factor

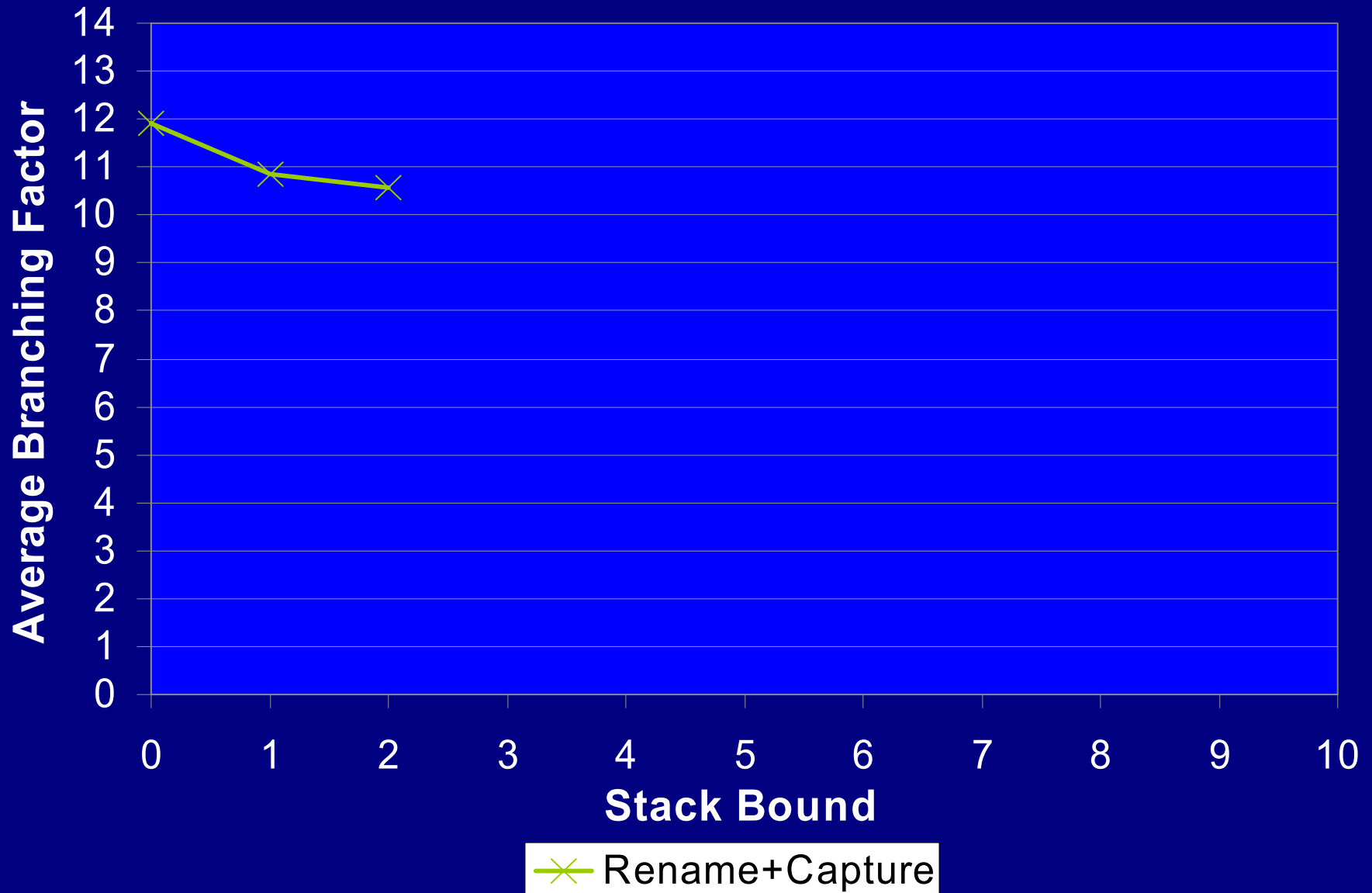


- Lower values indicate greater precision

Precision: NFA Model



Precision: PDA Model, procmail



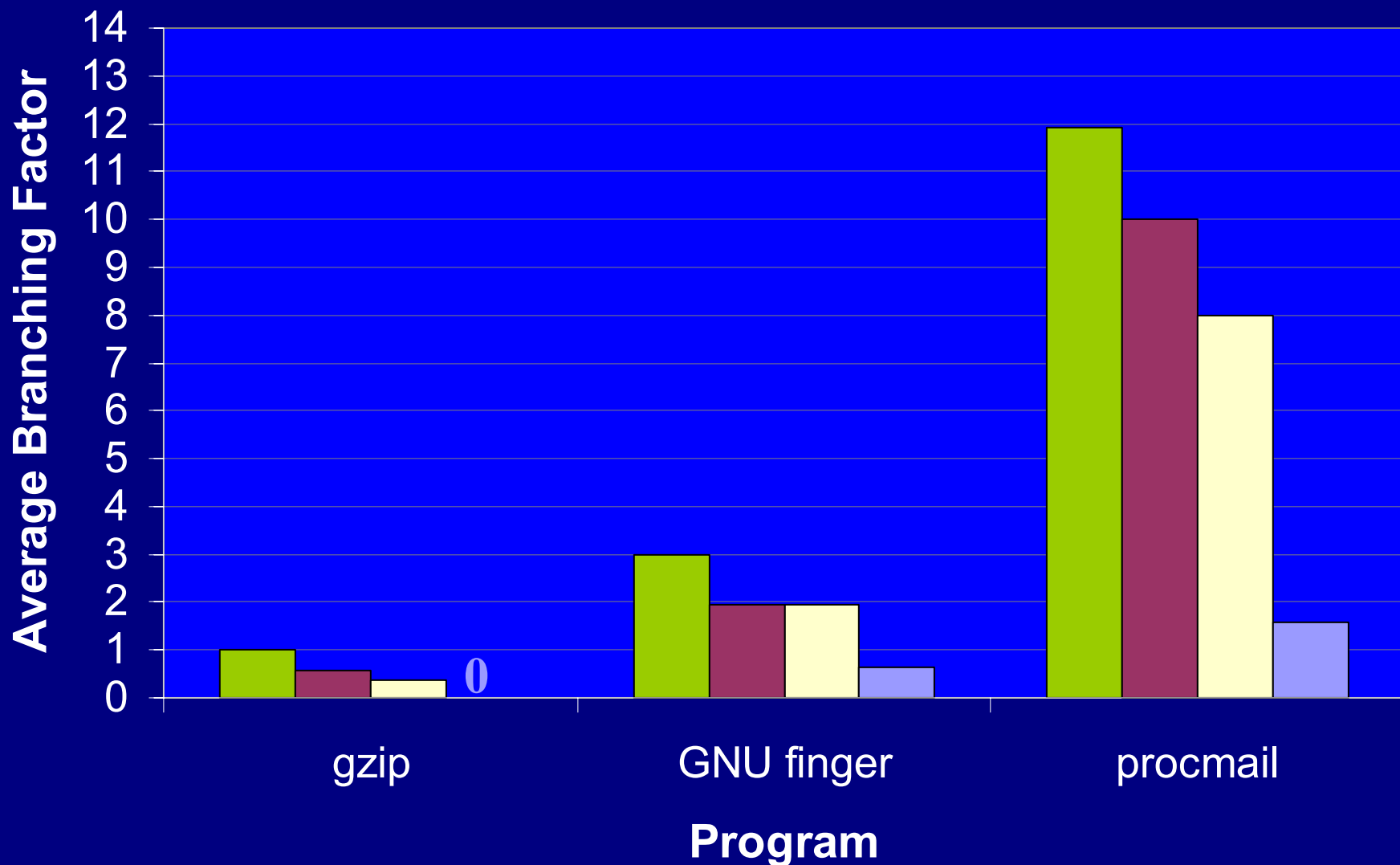
Optimizations to Improve Precision

- Observation: PDA is more precise than NFA because it provides context sensitivity
- Idea: Insert **null calls** into NFA model to add some context sensitivity without suffering runtime cost of PDA

Null Call Experiments

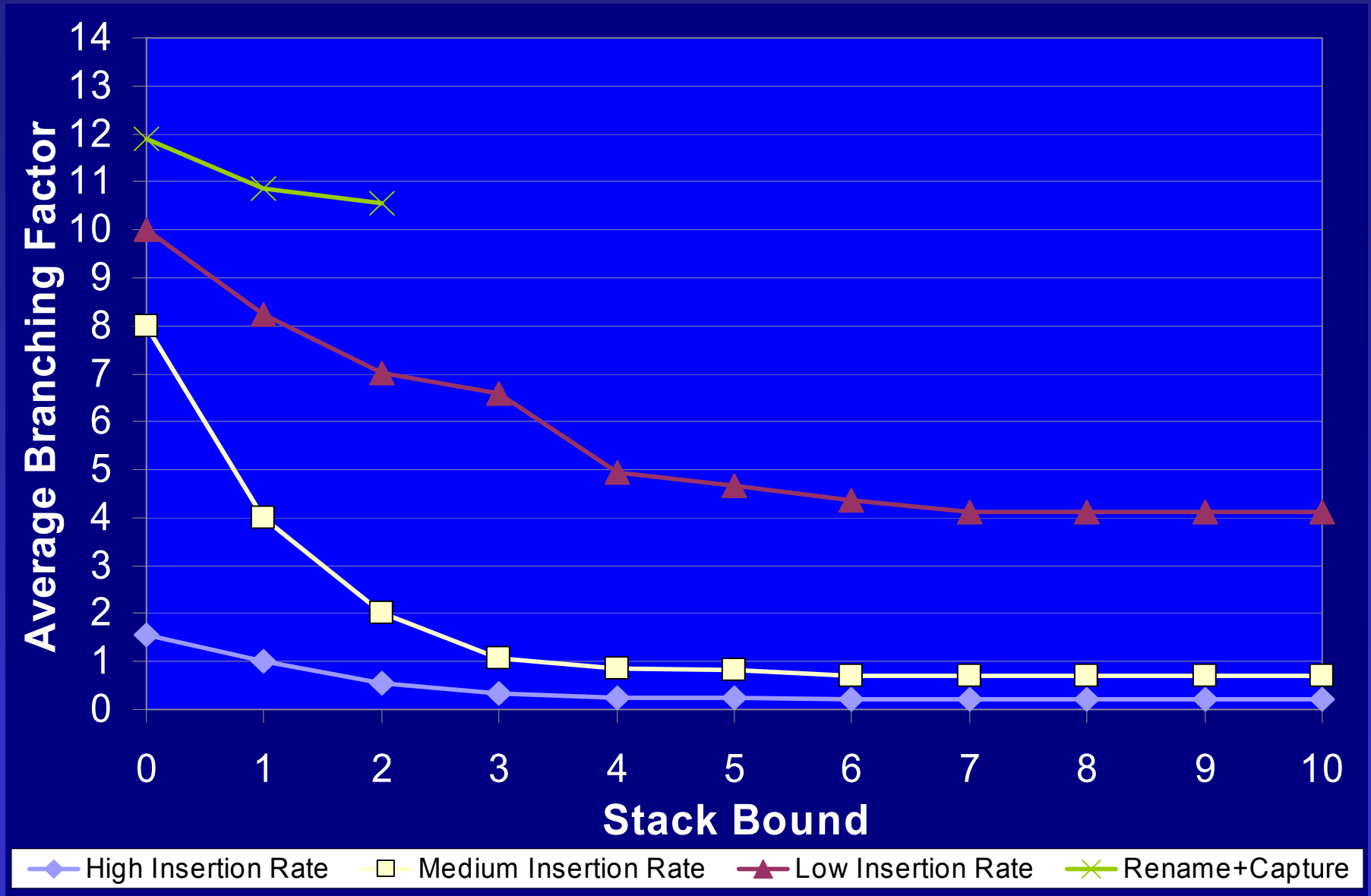
- Inserted null calls at 3 rates
 - **High**: At entries of functions with fan-in of 2 or greater
 - **Medium**: At entries of functions with fan-in of 5 or greater
 - **Low**: At entries of functions with fan-in of 10 or greater

Precision: NFA Model with Null Calls

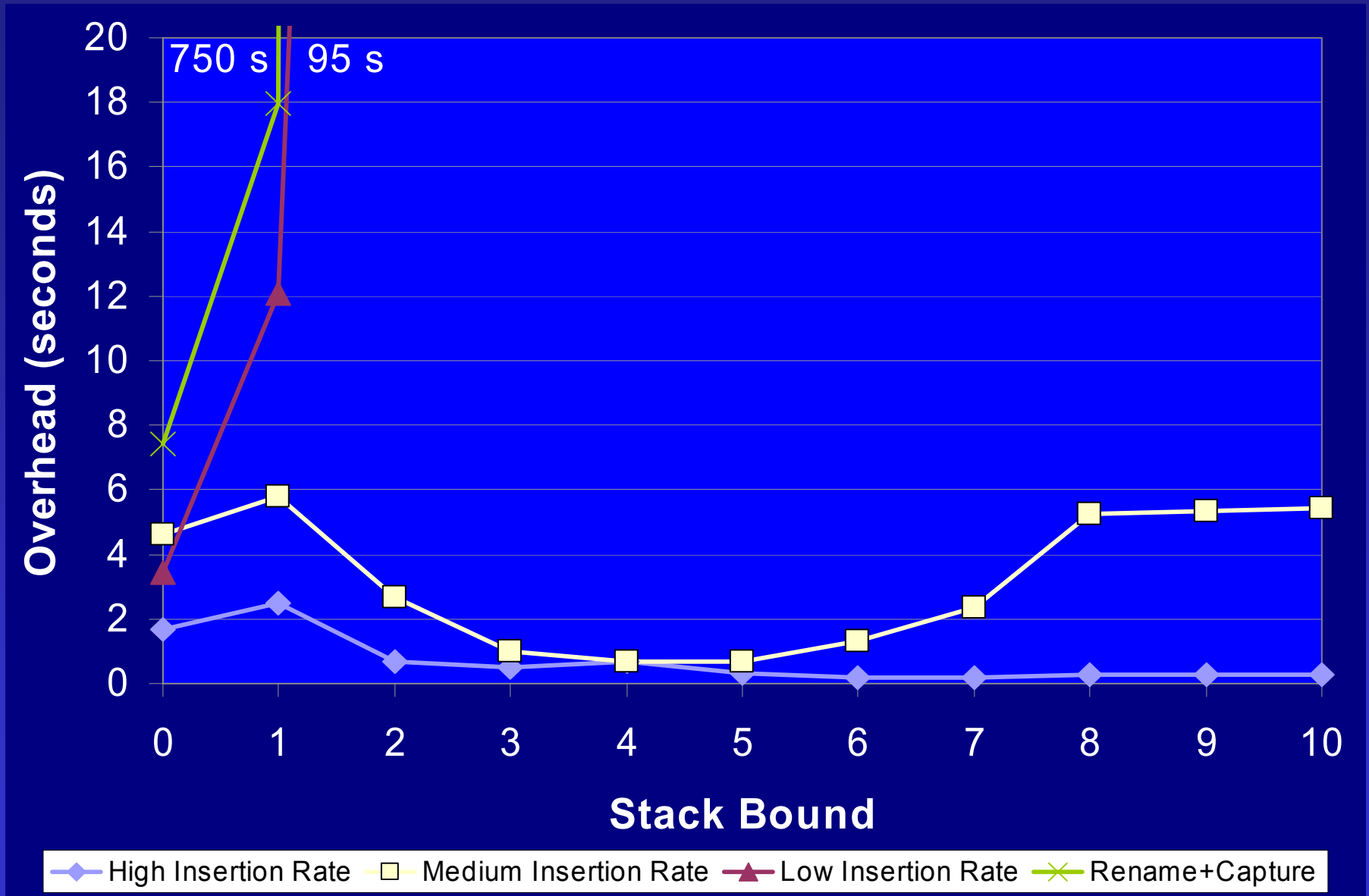


Legend: Rename+Capture (green), Low Insertion Rate (maroon), Medium Insertion Rate (yellow), High Insertion Rate (blue)

Precision: PDA Model with Null Calls, procmail



Overhead: PDA Model with Null Calls, procmail



Important Ideas

- Specifications generated automatically from binary code analysis
- Operate a finite state machine modeling correct execution
- PDA model is precise but suffers high overhead
- Bounded PDA stack & null calls allow use of precise PDA model

Detecting Manipulated Remote Call Streams

Jonathon Giffin, Somesh Jha, Barton Miller

Computer Sciences Department

University of Wisconsin

`giffin@cs.wisc.edu`