Attacks and Defenses

Barton Miller
Jonathon Giffin, Somesh Jha
University of Wisconsin
{bart,giffin,jha}@cs.wisc.edu

WiSA - Wisconsin Safety Analyzer
http://www.cs.wisc.edu/wisa

Overview

Attacks

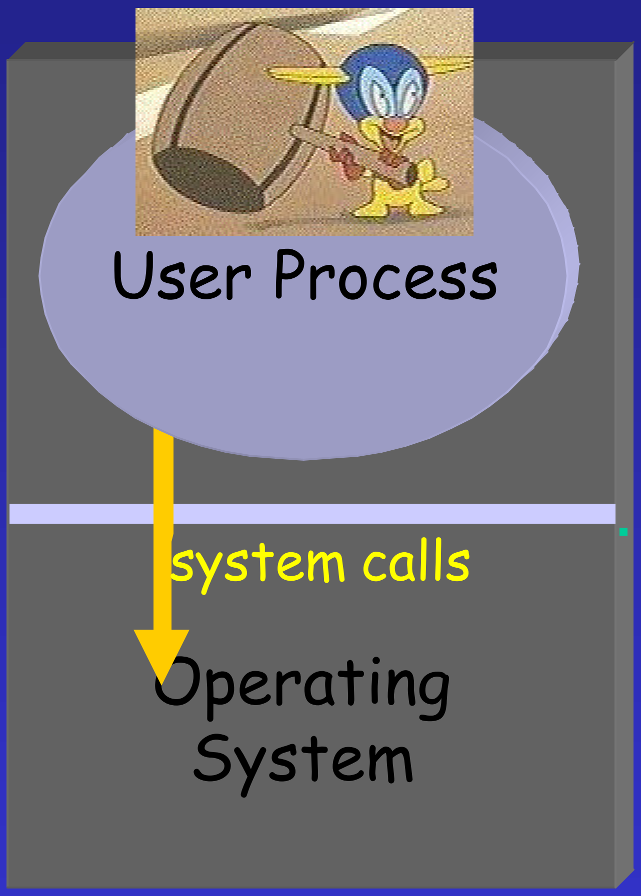
- Server attack (conventional host-based IDS)
- Remote execution attack (remote IDS)

Model-based intrusion detection

- Constructing program models using static binary analysis
- Accuracy/performance tradeoff in prior models
- Our new Dyck model solves tradeoff
- Data-flow analysis to recover arguments

Milestones

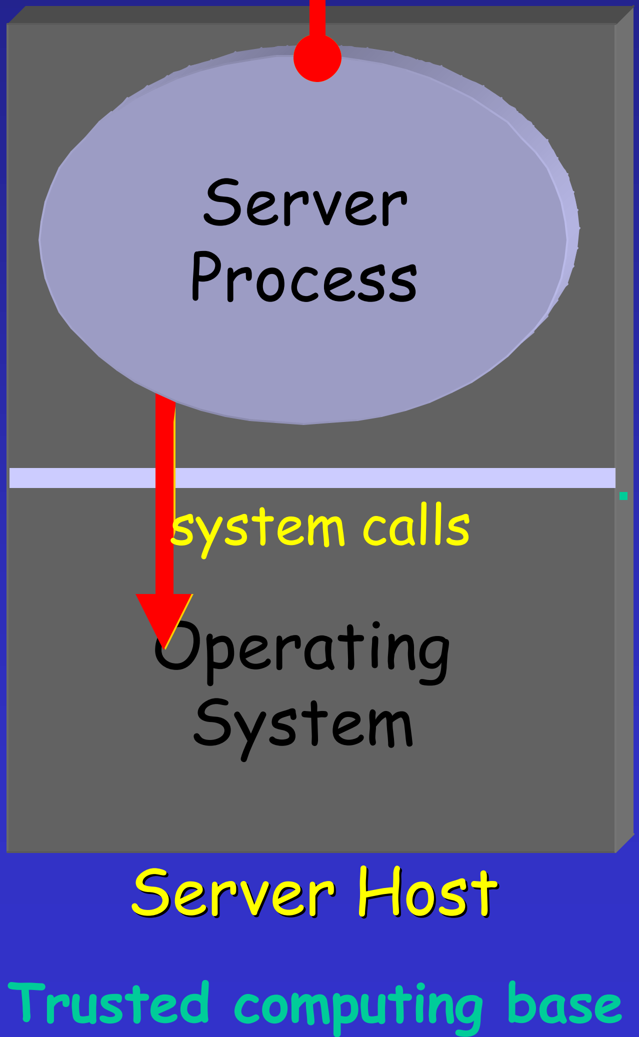
Worldview



Trusted computing base

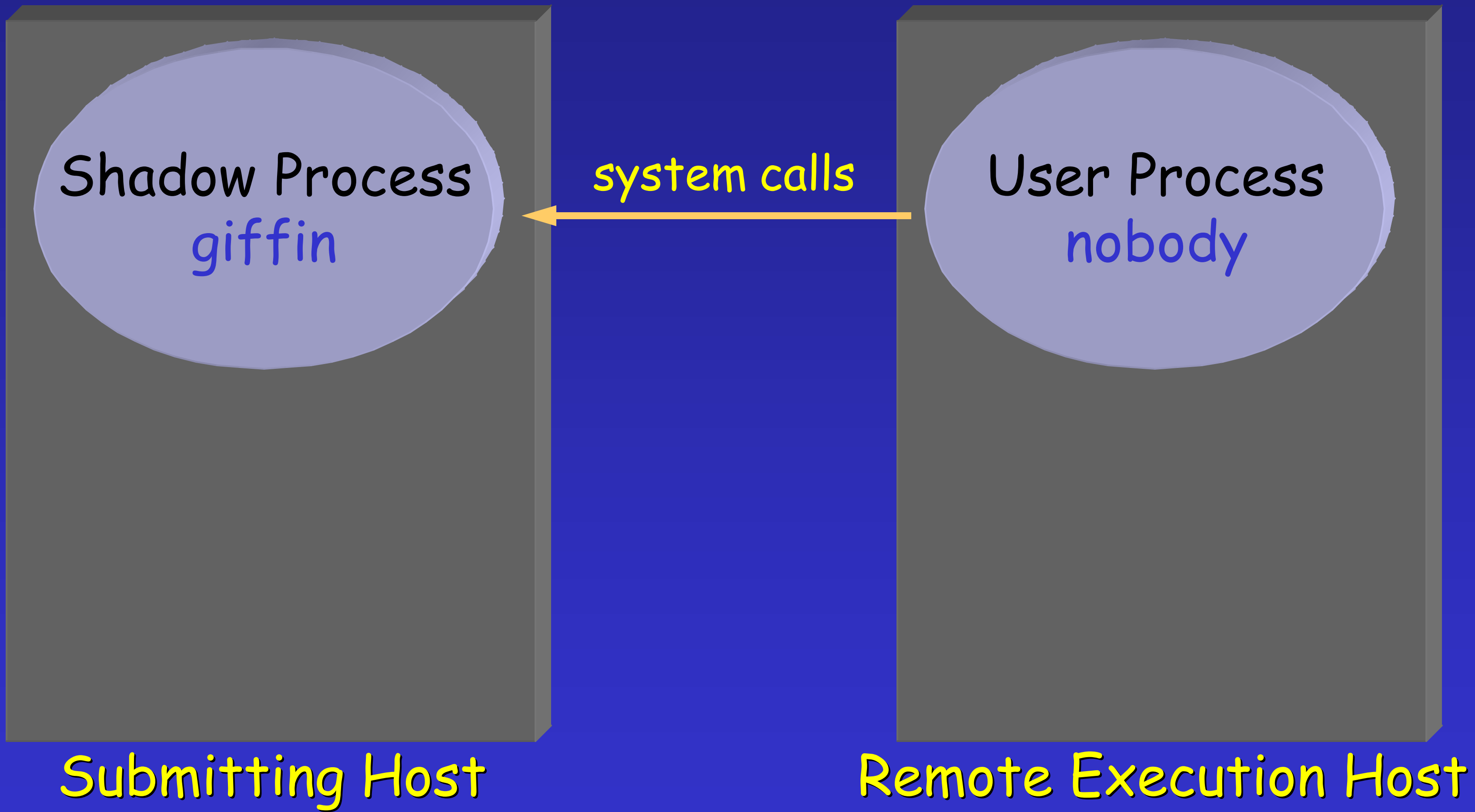
- Running processes make operating system requests
- Changes to trusted computing base done via these requests
- Attacker subverts process to generate malicious requests

Example: Server Attack



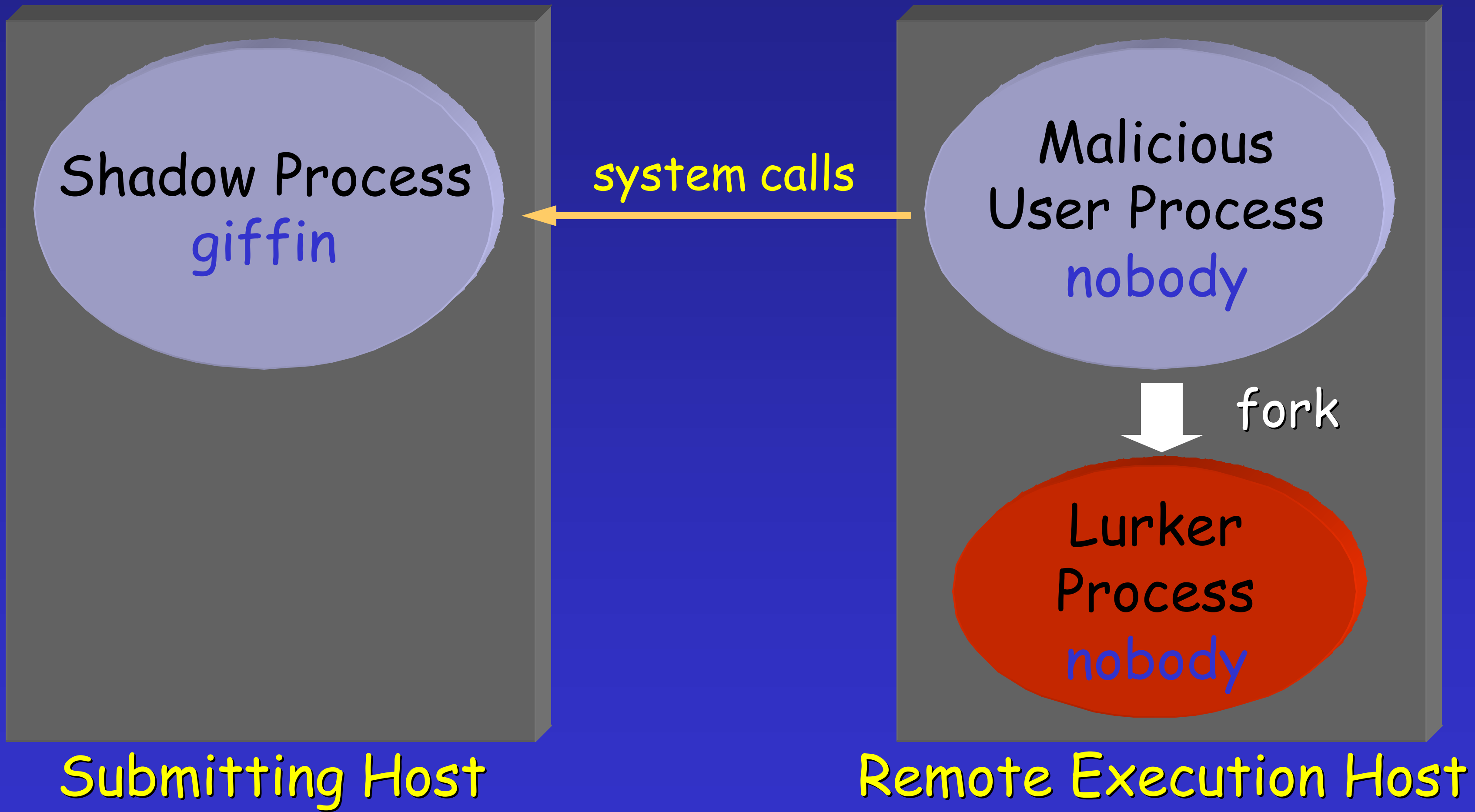
- Goal: Execute malicious code in the server

Example: Remote Execution Attack



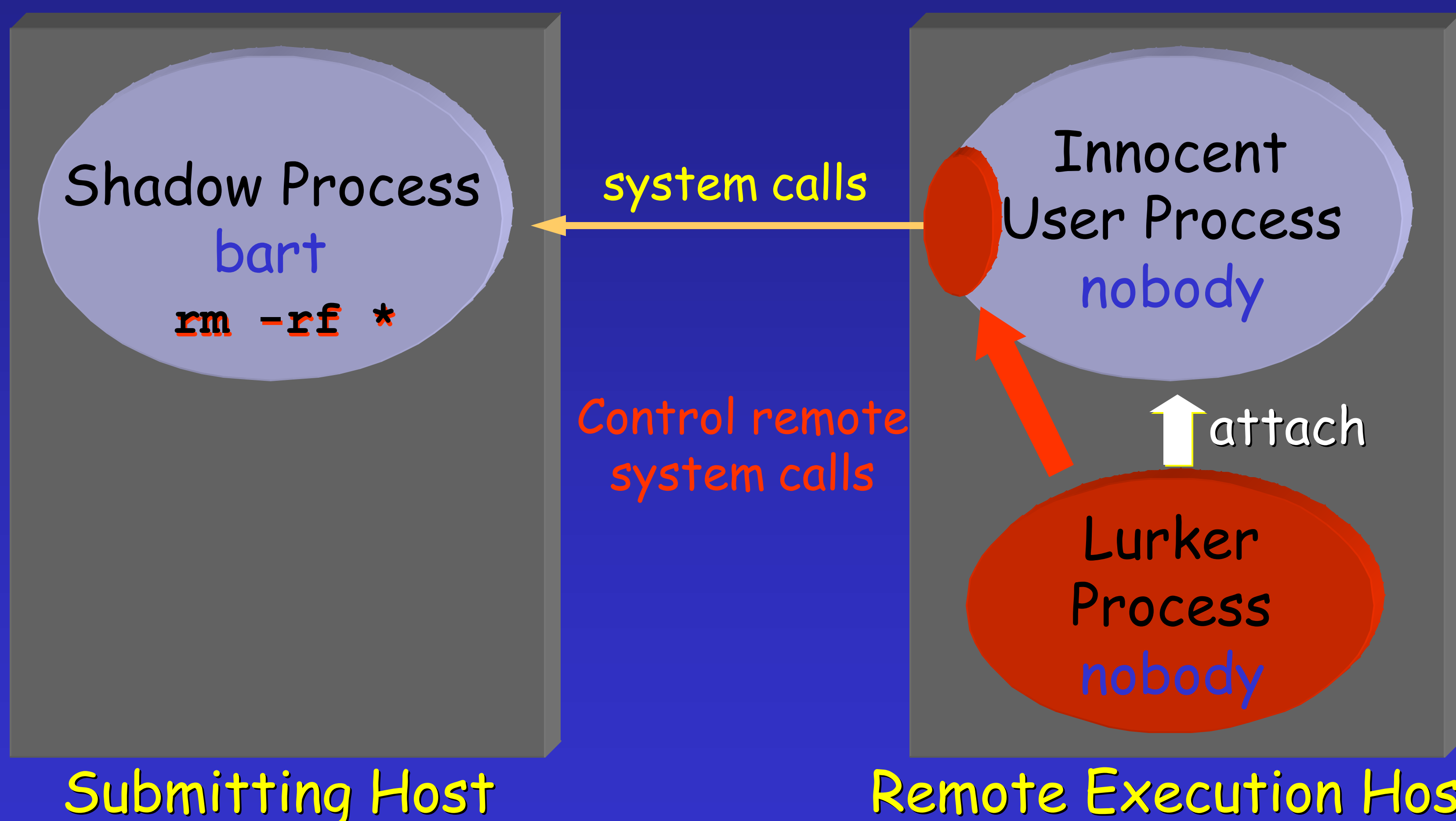
Trusted computing base

Example: Remote Execution Attack



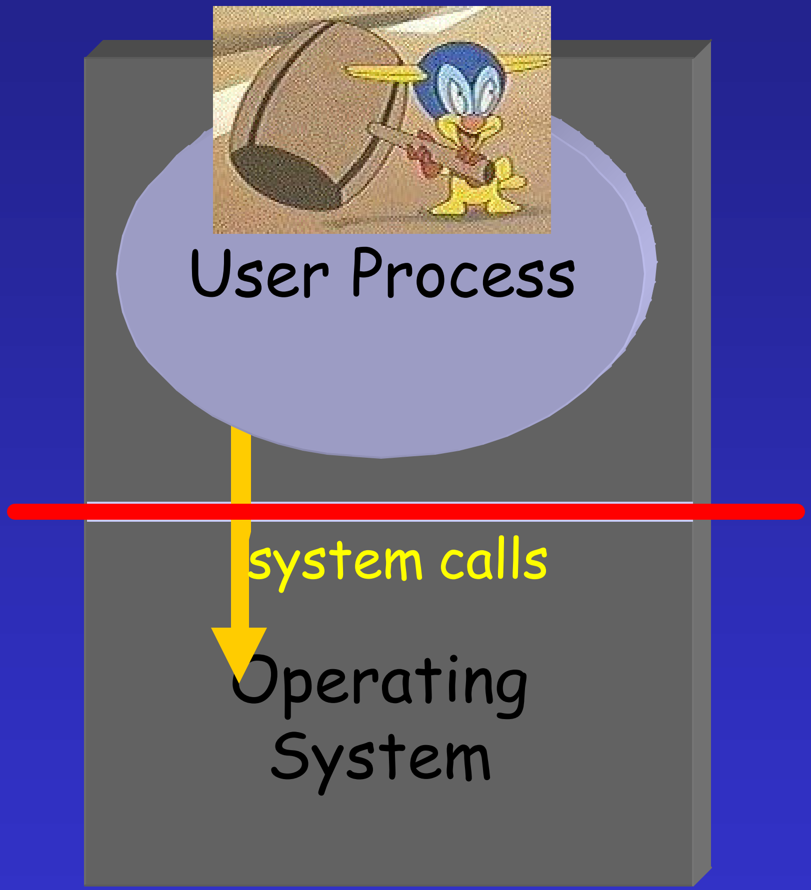
Trusted computing base

Example: Remote Execution Attack



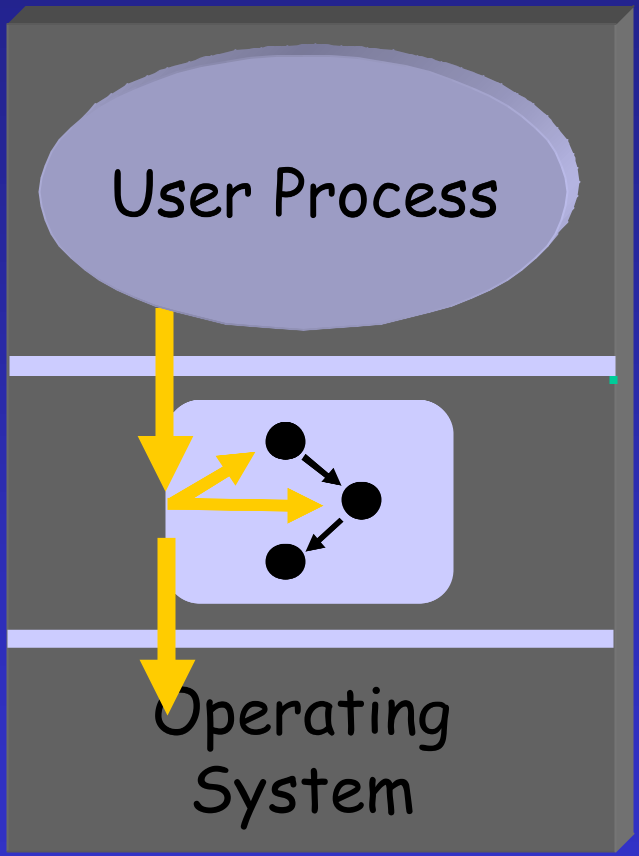
Trusted computing base

Our Objective



- Detect malicious activity before harm caused to local machine
- … before operating system executes malicious system call

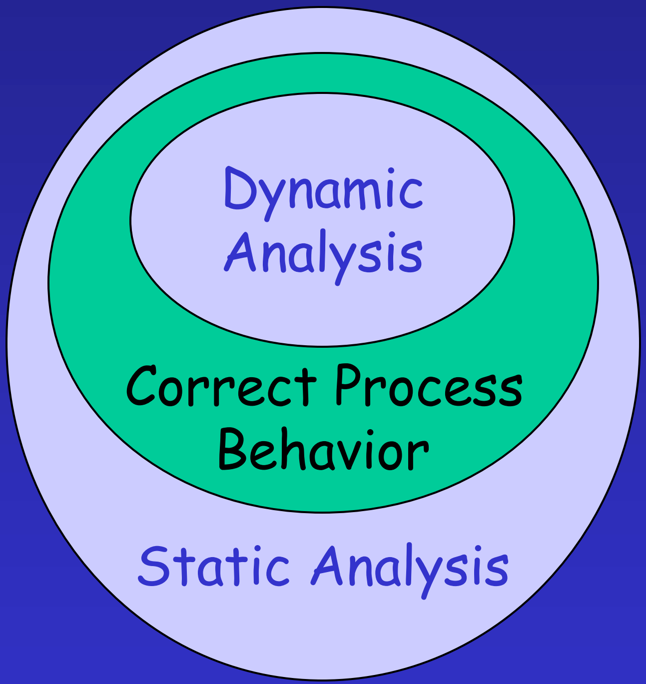
Model-Based Intrusion Detection



Trusted computing base

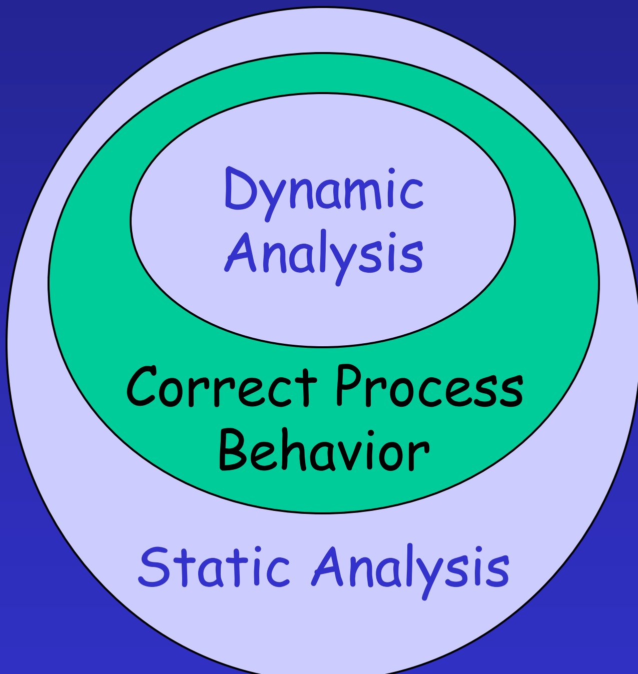
- Build model of correct program behavior
- Runtime monitor ensures execution does not violate model
- Runtime monitor must be part of trusted computing base

Automated Model Construction



- Dynamic analysis
  - Under-approximates correct behavior
  - False alarms
  - Forrest, Sekar, Lee
- Static analysis
  - Over-approximates correct behavior
  - False negatives
  - Wagner&Dean, our work
  - Previous attempts at precise models problematic

Automated Model Construction

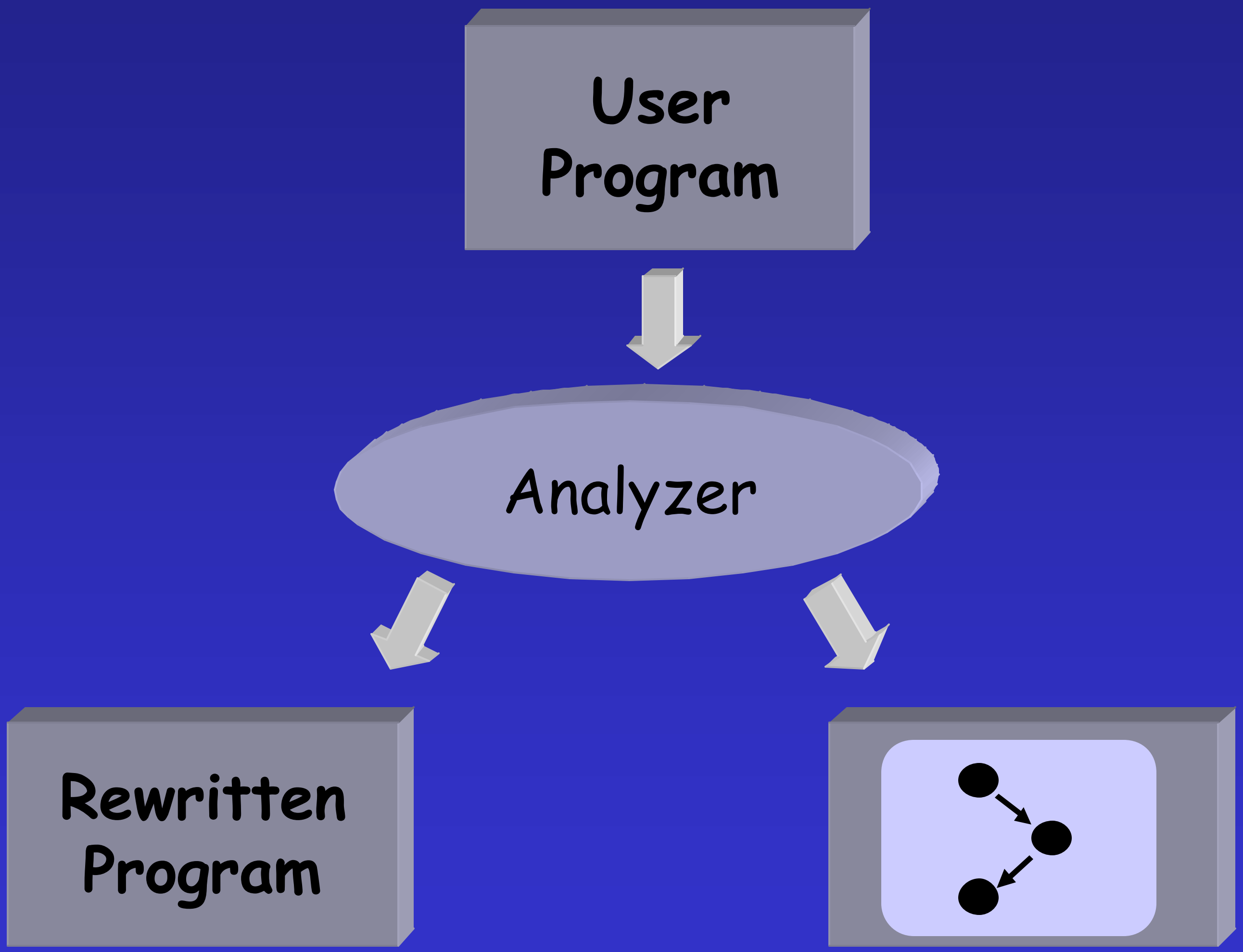


- Static analysis challenge
  - Design an efficient, context-sensitive model
- Answers
  - Dyck model
  - Argument dependency recovery

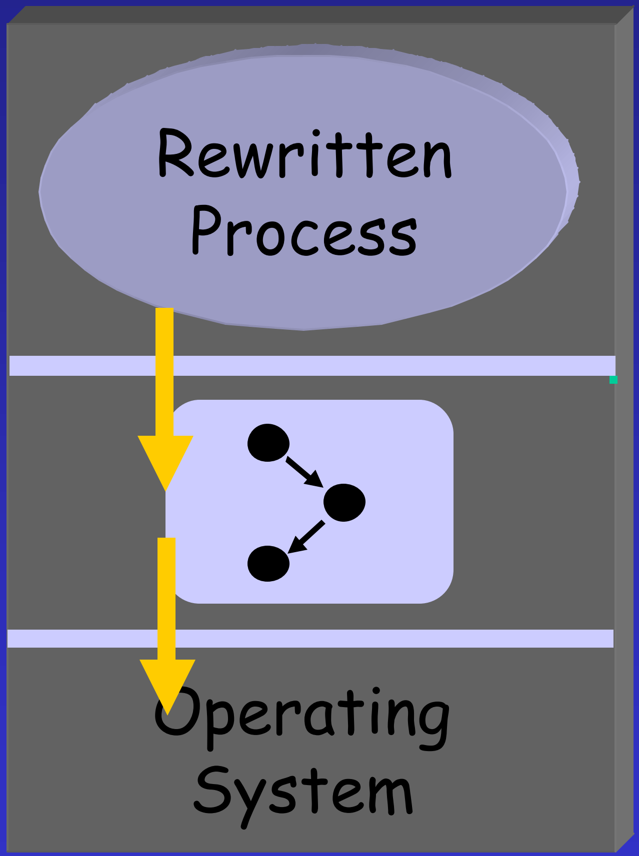
Our Approach

- Build model of correct program behavior
  - Static analysis of binary code
  - Construct an automaton modeling all system call sequences the program can generate
- Ensure execution does not violate model
  - Use automaton to monitor system calls.
  - If automaton reaches an invalid state, then an intrusion attempt occurred.

Model-Based Intrusion Detection



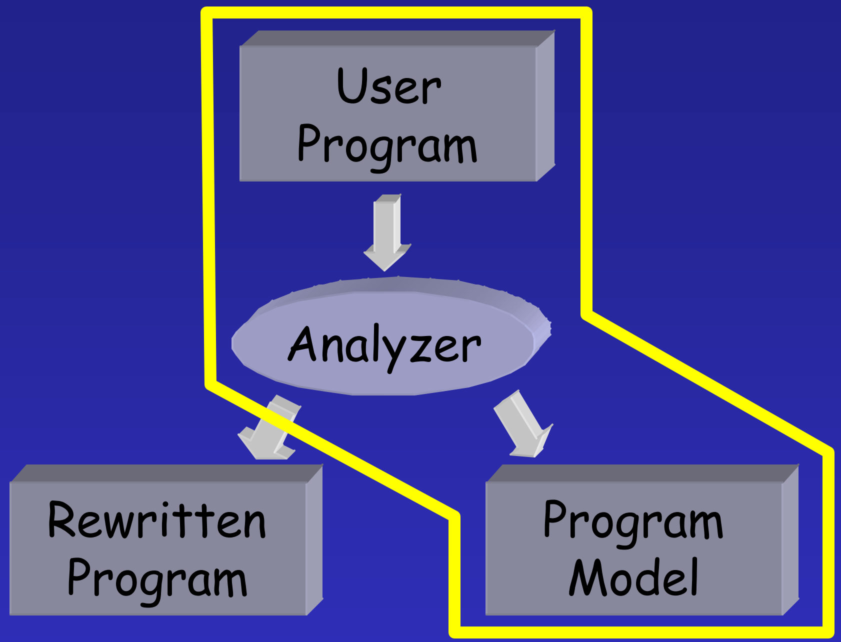
Model-Based Intrusion Detection

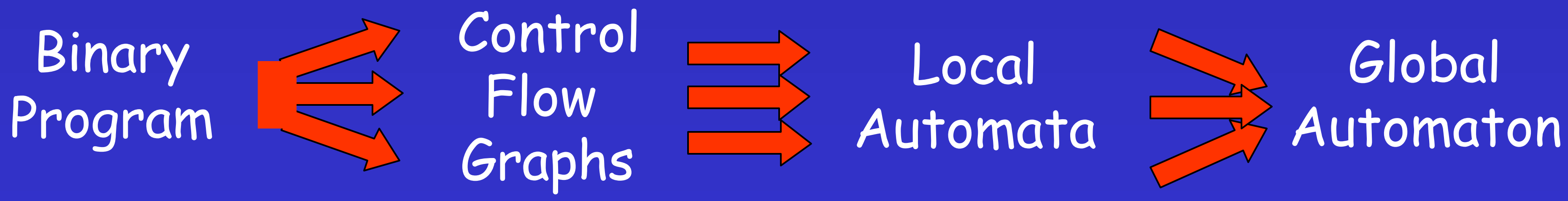


Trusted computing base

- Build model of correct program behavior
- Runtime monitor ensures execution does not violate model
- Runtime monitor must be part of trusted computing base

Model Construction



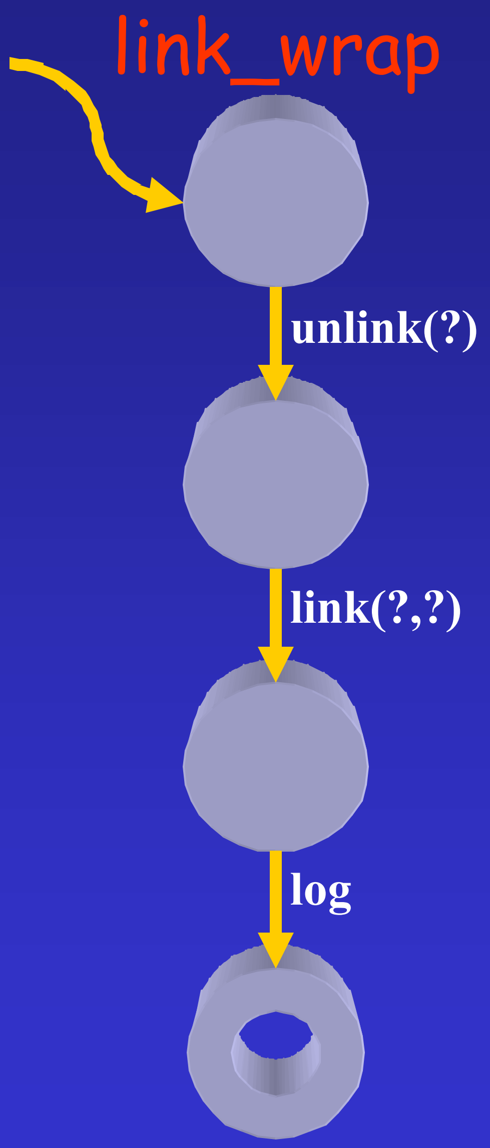


Code Example

```
link_wrap:
  save %sp, -596, %sp
  call unlink
  mov %i1, %o0
  mov %i1, %o1
  call link
  mov %i0, %o0
  add %sp, 56, %o0
  mov 50, %o1
  sethi %hi(str), %o2
  call snprintf
  or %o2, %lo(str), %o2
  call log
  add %sp, 56, %o0
  ret
  restore
```

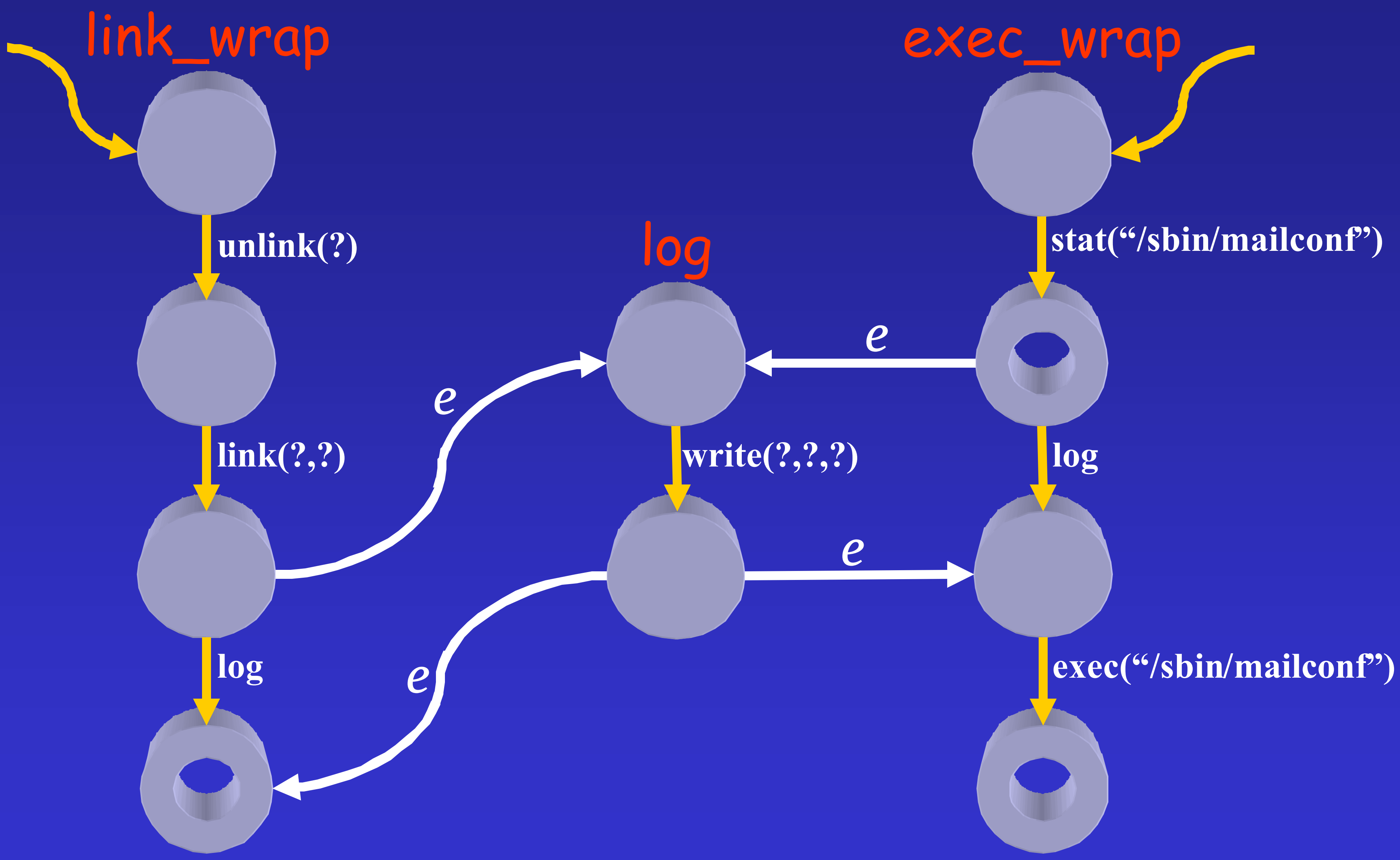
```
void
link_wrap(char *f, char *t)
{
  char msg[BUFFSIZE];

  unlink(t);
  link(f, t);
  snprintf(msg, BUFFSIZE,
    "Linked %s to %s, f, t);
  log(msg);
}
```

Local Automaton




```
void
link_wrap(char *f, char *t)
{
   char msg[BUFFSIZE];

   unlink(t);
   link(f, t);
   snprintf(msg, BUFFSIZE,
     "Linked %s to %s, f, t);
   log(msg);
}
```

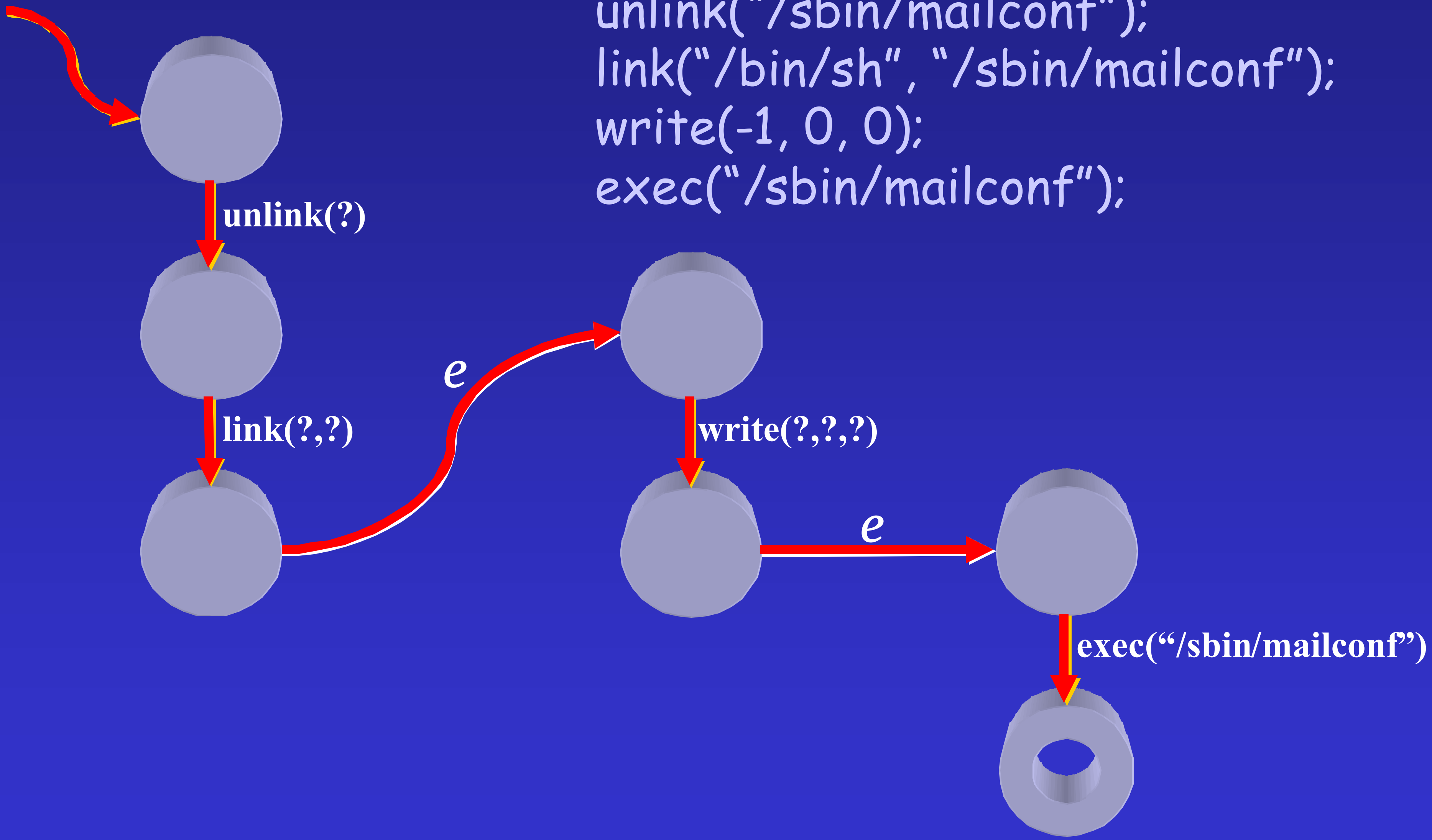
NFA Model



link_wrap
exec_wrap
log
unlink(?)
stat("/sbin/mailconf")
e
e
link(?,?)
write(?,?,?)
log
e
log
e
exec("/sbin/mailconf")

Impossible Paths

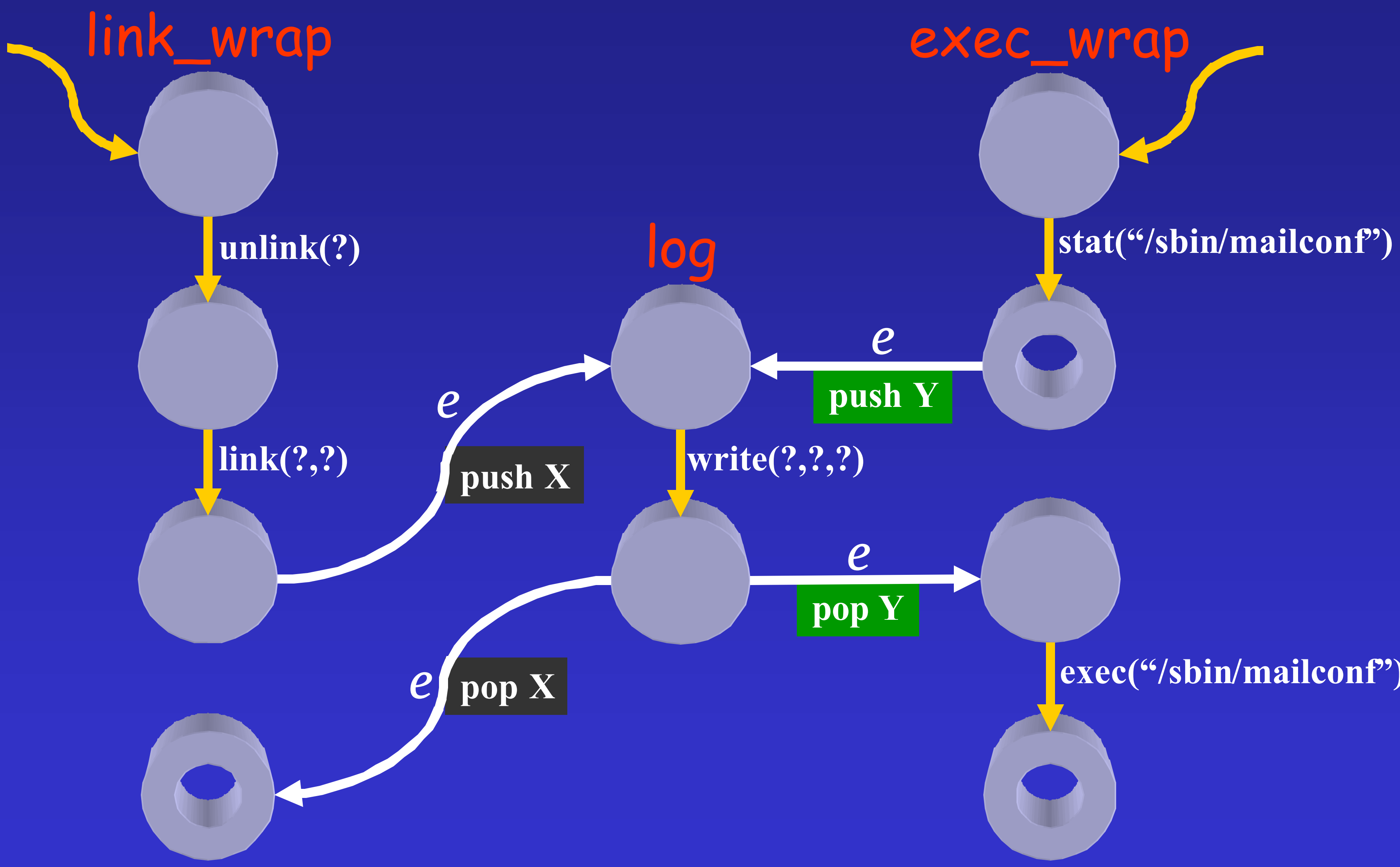
unlink("/sbin/mailconf");
link("/bin/sh", "/sbin/mailconf");
write(-1, 0, 0);
exec("/sbin/mailconf");



Adding Context Sensitivity

- Model call & return behavior of function calls
- Use pushdown automaton (PDA) stack to model program's call stack
- Model is sensitive to calling context of each system call

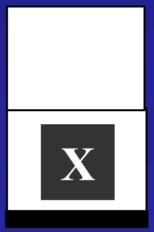
PDA Model



link_wrap
exec_wrap
unlink(?)
log
stat("/sbin/mailconf")
e
push Y
e
push X
link(?,?)
write(?,?,?)
e
pop Y
e
pop X
exec("/sbin/mailconf")

PDA State Explosion

- e-edge identifiers maintained on a stack
  - Stack non-determinism is expensive
  - Unbounded stacks add complexity
  - Best-known algorithm: cubic in automaton size



- Unusable as program model
  - Orders of magnitude slowing of application
    - [Wagner et al. 01, Giffin et al. 02]
  - Conclusion: only weaker NFA models have reasonable performance

Dyck Model

- Efficiently tracks calling context

- As powerful as full PDA
- Efficiency approaches NFA model

- Implication: accuracy & performance can coexist
  - Invalidates previous conclusion

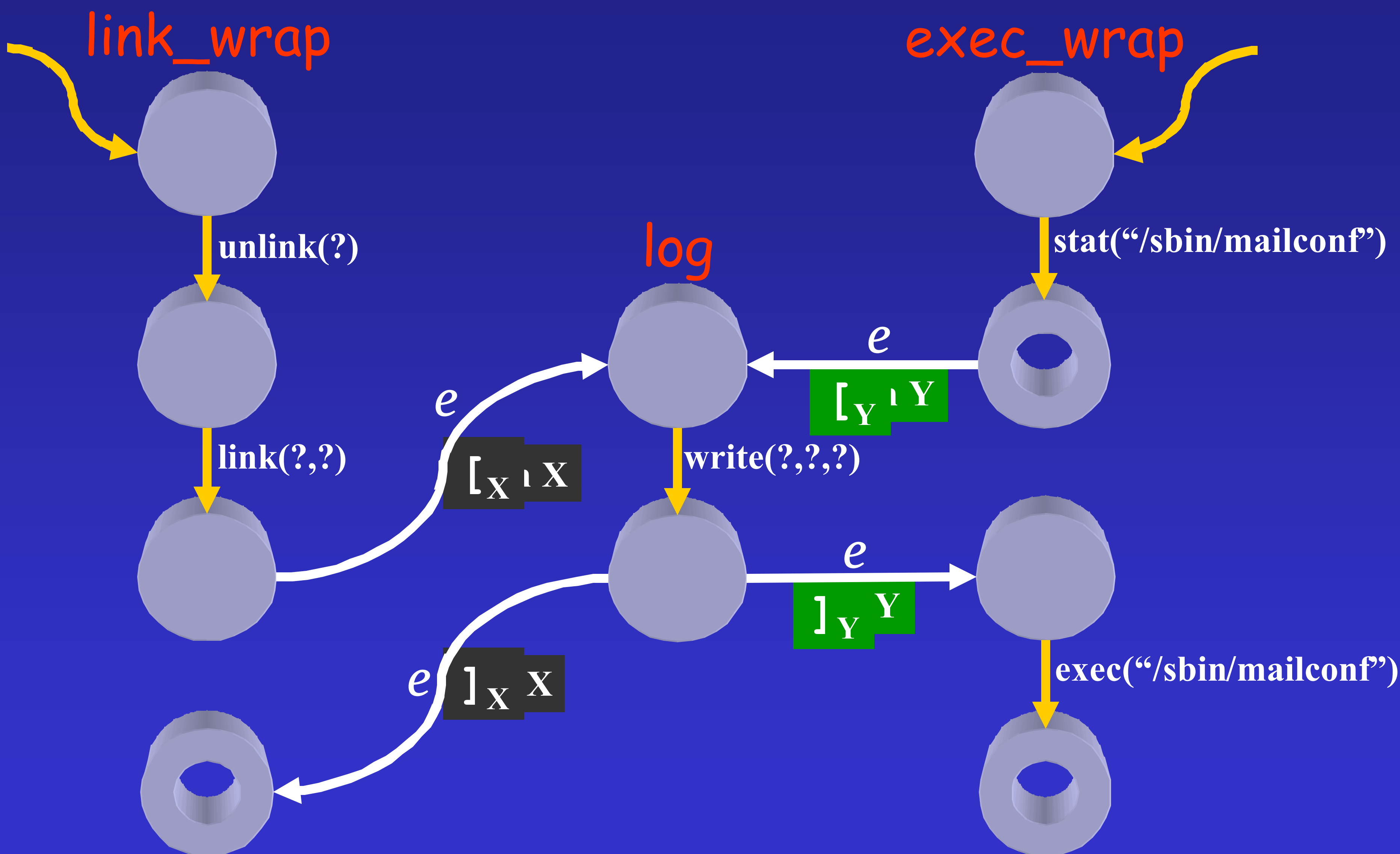
Dyck Model

- Bracketed context-free language
  - [Ginsberg & Harrison 67]

stat $[_y$ write $]_y$ exec
unlink link $[_x$ write $]_x$

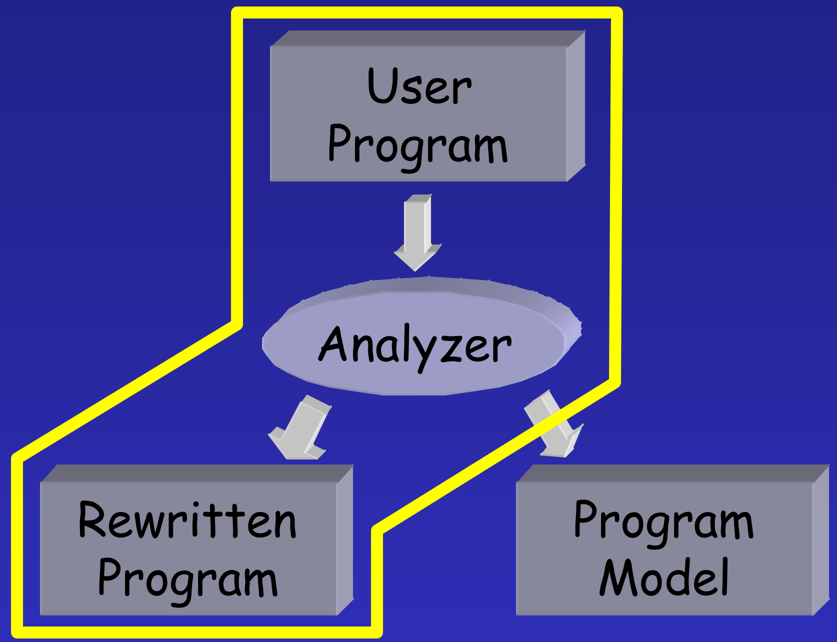
- Matching brackets are alphabet symbols
  - Exposes stack operations to runtime monitor
  - Rewrite binary to generate bracket symbols
  - [Giffin et al. 04]

Dyck Model



link_wrap
exec_wrap
unlink(?)
log
stat("/sbin/mailconf")
e
[Y Y
e
link(?,?)
[X X
write(?,?,?)
e
]Y Y
e
]X X
exec("/sbin/mailconf")

Binary Rewriting



Binary Program → Rewritten Binary

Binary Rewriting

- Insert code to generate bracket symbols around function call sites
- Notify monitor of stack activity

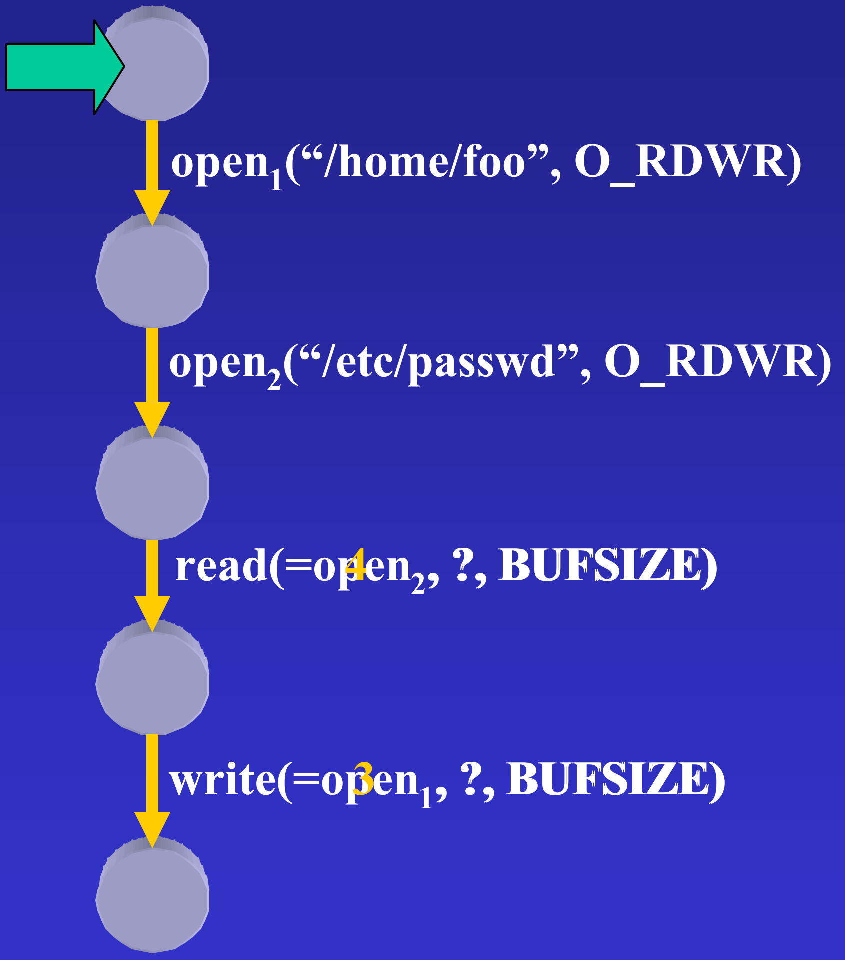
```
void
link_wrap(char *f, char *t)
{
  char msg[BUFFSIZE];

  unlink(t);
  link(f, t);
  snprintf(msg, BUFFSIZE,
    "Linked %s to %s, f, t);
  leftX();
  log(msg);
  rightX();
}
```

Data-Flow Analysis

- Can use knowledge of argument values to make model more precise.
- Use data-flow analysis of arguments:
  - Argument recovery
    - Sets of constant values
    - Sets of regular expression strings
  - Argument dependencies upon system call return values
  - System call return values that control branching

Argument Dependencies

```
. . .
fd1 = open("/home/foo",
           O_RDWR);
fd2 = open("/etc/passwd",
           O_RDWR);
read(fd2, buf, BUFSIZE);
write(fd1, buf, BUFSIZE);
. . .
```



$open_1() = 3;$

$open_2() = 4;$

Test Programs

Program	Number of Instructions
procmail	107,246
gzip	56,710
eject	70,177
fdformat	67,874
cat	54,028

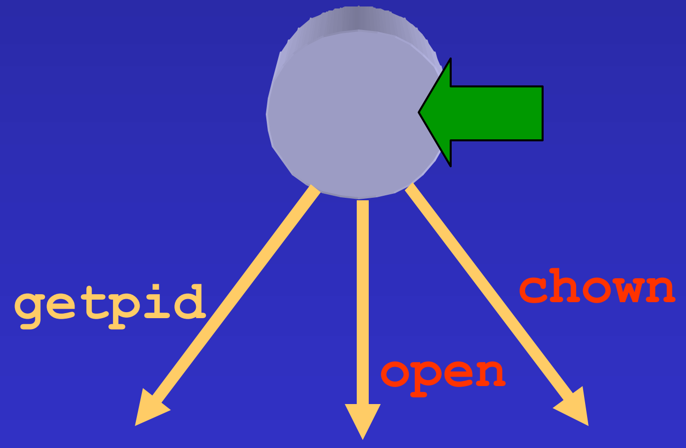
Runtime Overheads

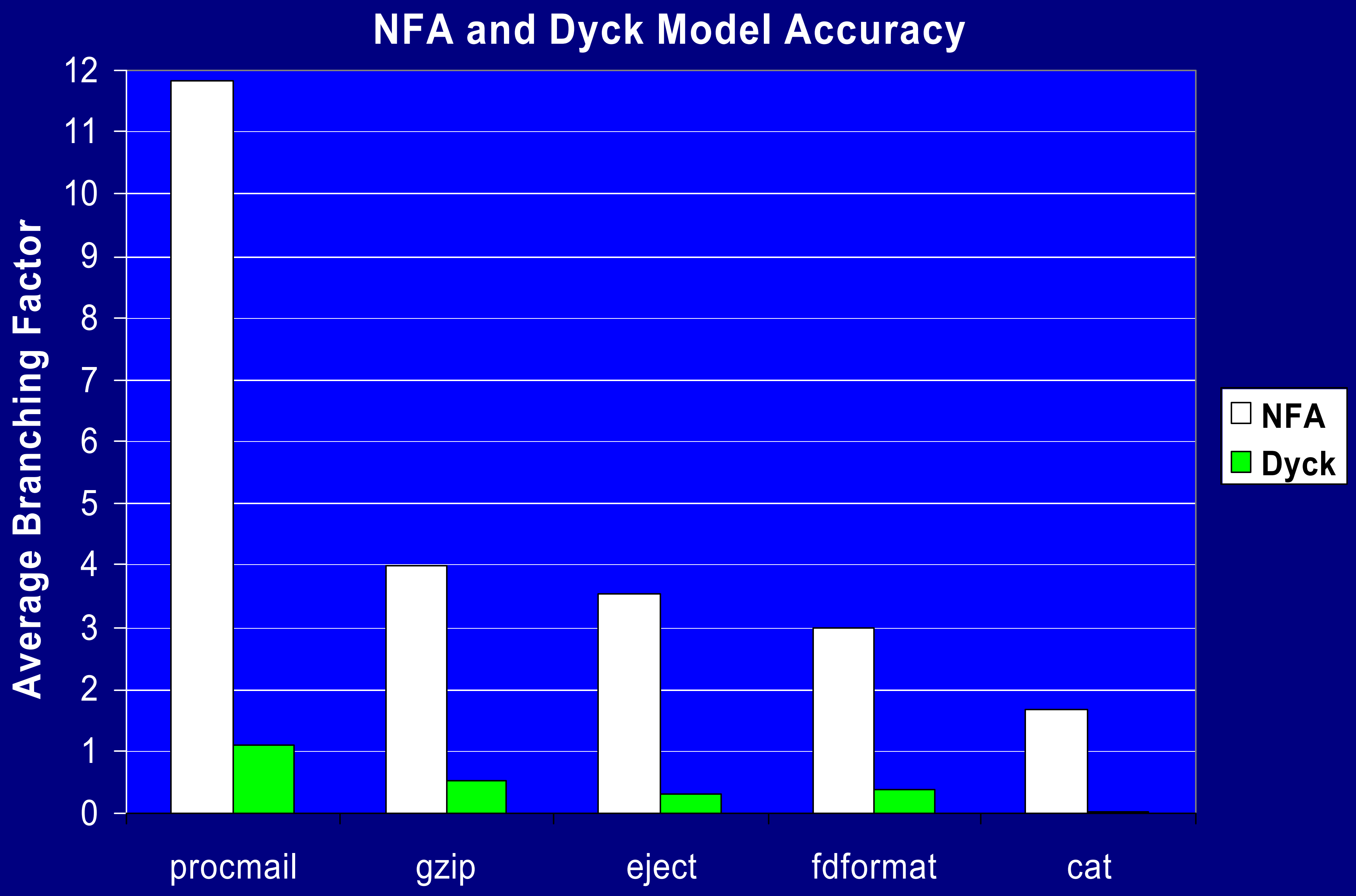
Execution times in seconds

Program	Base	NFA	Increase	Dyck	**Increase**
procmail	0.42	0.37	0%	0.40	0%
gzip	7.02	6.61	0%	7.16	2%
eject	5.14	5.17	1%	5.22	2%
fdformat	112.41	112.36	0%	112.38	0%
cat	54.65	56.32	3%	80.78	48%

Accuracy Metric

- Average branching factor




NFA and Dyck Model Accuracy
Average Branching Factor
12
11
10
9
8
7
6
5
4
3
2
1
0
procmail
gzip
eject
fdformat
cat
NFA
Dyck

Important Ideas

- Model-based intrusion detection forces execution behavior to match model.
- Statically constructed program models historically compromise accuracy for efficiency.
- The Dyck model is the first efficient context-sensitive specification.

Milestones

Nov 01:
- NFA & PDA construction
- Call-site renaming

Jan 02:
- Bounded stack PDA
- Static argument recovery
- Binary code instrumentation

May 02:
- Began shared object analysis

Aug 02:
- USENIX Security paper
- Interprocedural data-flow analysis
- Argument dependency recovery

Oct 02:
- Data-flow-sensitive analysis

Jan 03:
- Smart binary code instrumentation

March 03:
- Invented Dyck model

July 03:
- Implemented kernel trap monitoring

Feb 04:
- NDSS Paper

Milestones

- Two conference papers
  - J.T. Giffin, S. Jha, and B.P. Miller. Detecting manipulated remote call streams. In 11th USENIX Security Symposium, San Francisco, California, August 2002.
  - J.T. Giffin, S. Jha, and B.P. Miller. Efficient context-sensitive intrusion detection. In 11th Annual Network and Distributed Systems Security Symposium (NDSS), San Diego, California, February 2004.

Milestones

Feb 04:
- Support dynamic linking

Ongoing 2004:
- Formal modeling of attacks & defenses
- Investigating tool to construct attacks for better IDS testing

June 04:
- Move to BREW infrastructure

July 04:
- Recover arguments at additional program points

Ongoing 2004:
- Investigating hybrid static/dynamic techniques

Collaboration with Wenke Lee

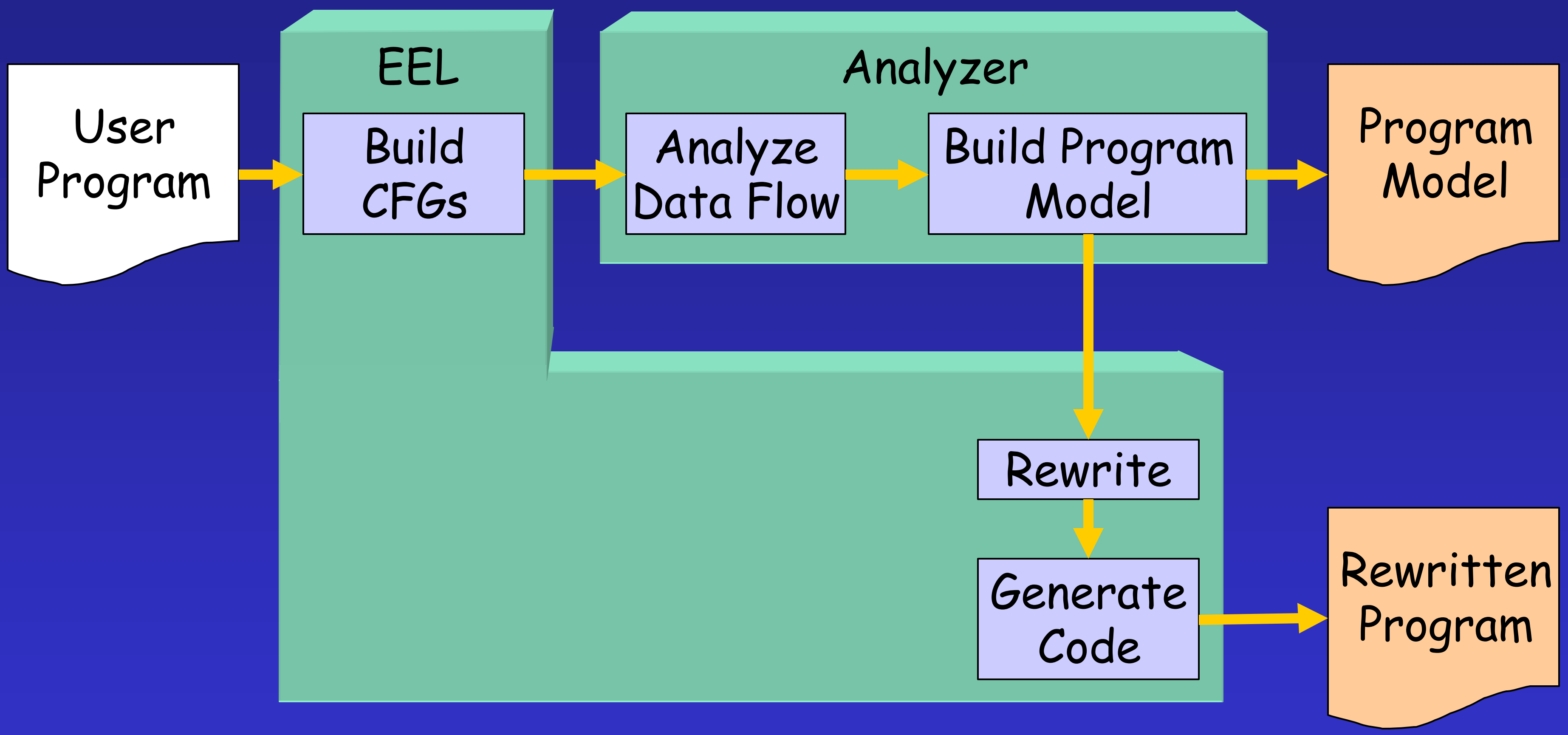
- Collaborated on static version of his dynamic analysis work
  - Compared with our Dyck model
  - Developed static model formalisms
  - Under submission: "Formalizing Sensitivity in Static Analysis for Intrusion Detection"
- Future: research hybrid techniques
  - New methods to recover calling context
  - Combine static & dynamic analysis

Attacks and Defenses

Barton Miller
Jonathon Giffin, Somesh Jha
University of Wisconsin
{bart,giffin,jha}@cs.wisc.edu

WiSA - Wisconsin Safety Analyzer
http://www.cs.wisc.edu/wisa

Architecture



Architecture