

Be Conservative: Enhancing Failure Diagnosis with Proactive Logging

*Ding Yuan, Soyeon Park, Peng Huang, Yang Liu,
Michael Lee, Xiaoming Tang, Yuanyuan Zhou, Stefan Savage*

University of California, San Diego

University of Illinois at Urbana-Champaign

<http://opera.ucsd.edu/errlog.html>

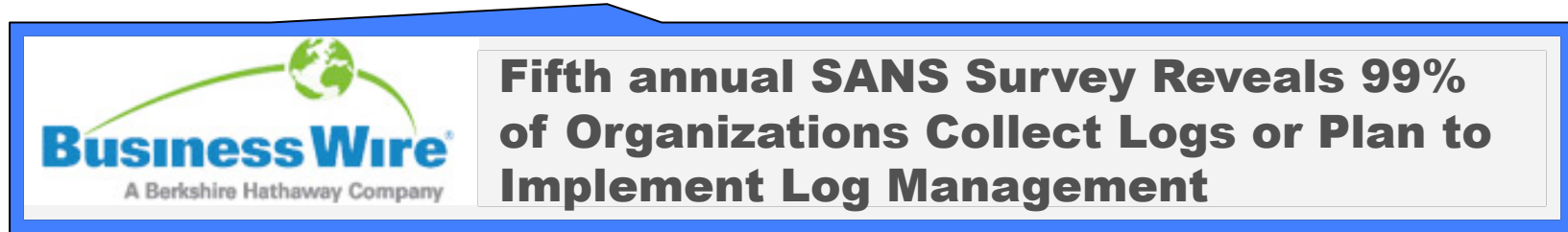


Motivation

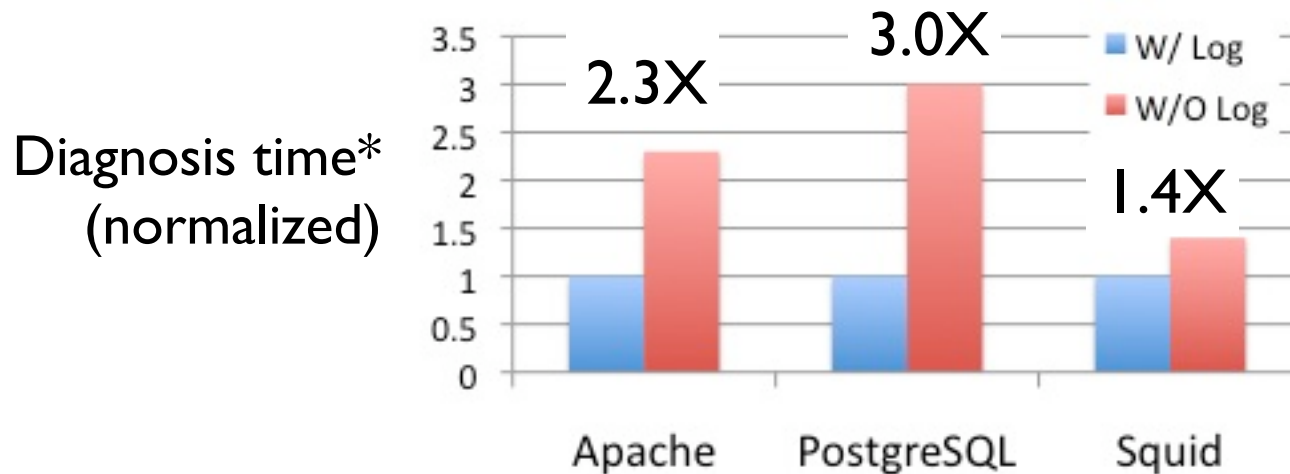
- ▶ **Production failures are hard to reproduce**
 - ▶ Privacy concerns for input
 - ▶ Hard to recreate the production setting

Importance of log messages

- ▶ Vendors actively collect logs



- ▶ Log messages cut diagnosis time by 2.2X



An real-world example of good logging

\$./apachectl start

Could not open group file: /etc/httpd/gorup

No such file or directory

Starting Apache
web server

Typo
misconfiguration

What if there is no such log message?

Real-world failure report

User:

“Apache httpd cannot start.
No log message printed.”

Developer:

Cannot reproduce the failure...
Ask lots of user information...
User’s misconfiguration:
typo in group file name.

Detected error
& terminate

```
if ((status = fopen(grpfile, ..)) != SUCCESS) {  
+ ap_log_error("Could not open group file: %s", grpfile);  
  return DECLINED;  
}
```

Reactive instead of proactive!

Real-world bug in Squid web-cache

User:

“In an array of squid servers, from time to time the available file descriptors drops down to nearly zero.

No log message or anything!”

45 exchanges

Developer:

Cannot reproduce the failure...

Ask user for [debug] level logs...

Ask user for configuration file

Additional log statements.

Ask user for DNS statistics...

Real-world bug in Squid web-cache

User:

“In an array of squid servers, from time to time the available file descriptors drops down to nearly zero.

No log message or anything!”

DNS lookup
error

Not handled
properly

```
if (status != OK) {  
idnsSendQuery (q);  
+ idnsTcpCleanup(q);  
+ error("Failed to connect to DNS server with TCP");  
}
```

What we have seen from the examples

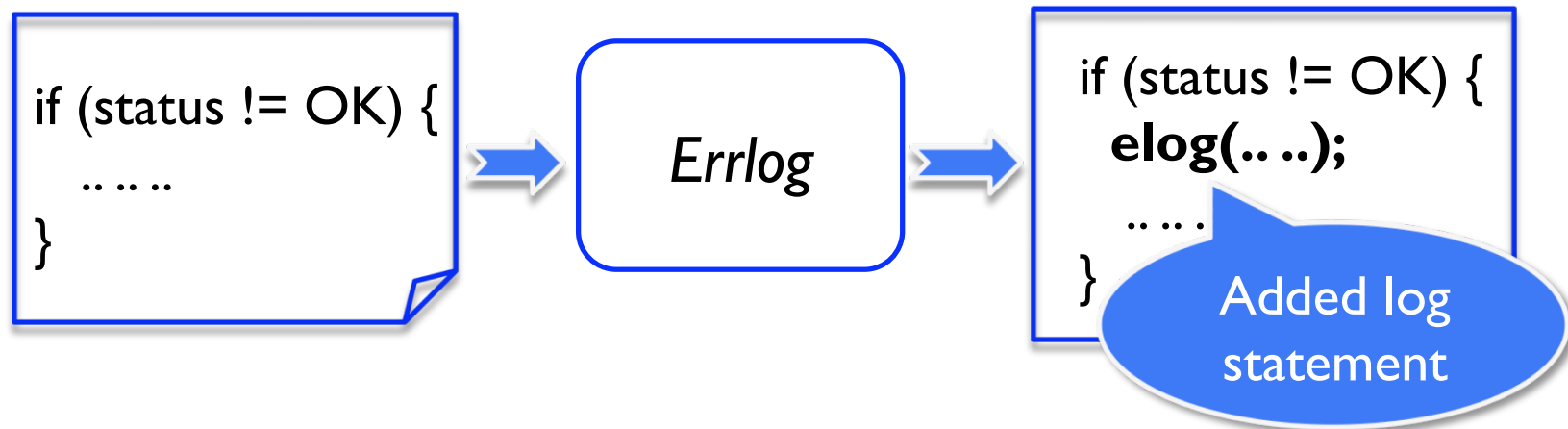
- ▶ Developers miss obvious log opportunities
 - ▶ *Analogy: solving crime without evidence*



- ▶ How many real-world cases are like this?
- ▶ What are other **obvious places to log**?

Our contributions

- ▶ Quantitative evidences
 - ▶ Many opportunities that developers could have logged
 - ▶ *Small set of generic “log-worthy” patterns*
- ▶ *Errlog*: a tool to automate logging



Outline

- ▶ Introduction
- ▶▶ Characterizing logging efficacy
- ▶ *Errlog* design
- ▶ Evaluation results
- ▶ Conclusion

Goals of our study

- ▶ Do real-world failures have log messages?
- ▶ Where to log?

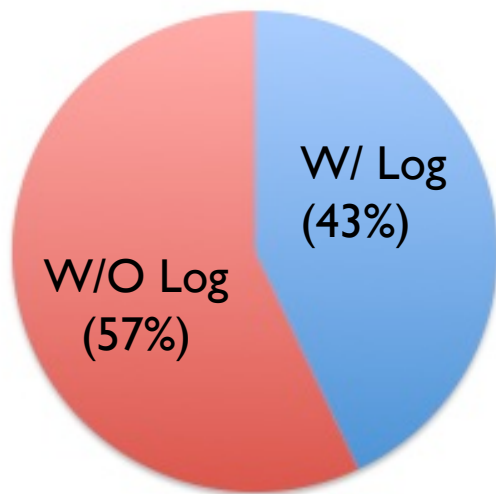
Study methodology

- ▶ Randomly sampled 250 recently reported failures*
 - ▶ Carefully studied the discussion and related code/patch
 - ▶ Study took 4 authors 4 months

Software	Sampled failures
Apache httpd	65
Squid	50
PostgreSQL	45
SVN	45
GNU Coreutils	45
Total	250

How many missed log message?

- ▶ Only 43% failures have log messages



Software	Failures with log
Apache httpd	37%
Squid	40%
PostgreSQL	53%
SVN	56%
Coreutils	33%
Overall	43%

How many missed log message?

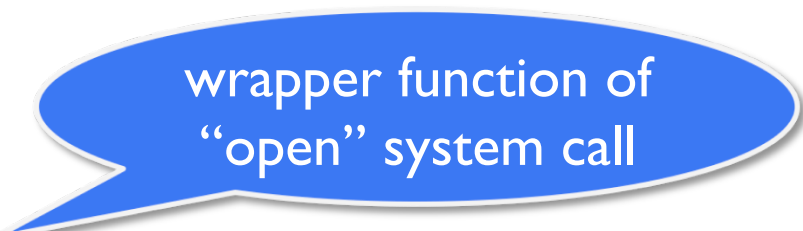
- ▶ Only 43% failures have log messages
- ▶ 77% have easy-to-log opportunities



Software	Failures with log
Apache httpd	37%
Squid	40%
PostgreSQL	53%
SVN	56%
Coreutils	33%
Overall	43%

Pattern I: function return error

No log:



wrapper function of
“open” system call

```
if ((status = fileopen (grpfile, ..)) != SUCCESS) {  
    return DECLINED;  
}  
  
/* Apache httpd misconfiguration. */
```

*Unnecessary user complain and
debugging efforts*

Pattern I: function return error

Good practice:

```
int main (..) {  
    if (svn_export(..))  
        ...  
}
```

print
message once

```
svn_err_t* svn_export(..) {  
    SVN_ERR(svn_versioned(..));  
}
```

```
svn_err_t* svn_versioned(..) {  
    if (!entry)  
        return error_create("%s is not under version control", ..);  
}
```

/* SVN */

Macro to detect all
func. return error

```
#define SVN_ERR(func)  
    svn_err_t* temp=(func);  
    if (temp)  
        return temp;
```

Propagate to caller

Create and return an
error message

Pattern I: function return error

Good practice:

```
int main (..) {  
    if (svn_export(..))  
        ...  
}
```

print
message once

```
svn_err_t* svn_  
    SVN_ERR(s  
}
```

```
svn_err_t* svn_versioned(..) {  
    if (!entry)  
        return error_create("%s is not under version control", ..);  
}
```

Macro to detect all
func. return error

```
#define SVN_ERR(func)  
    svn_err_t* temp=(func);  
    if (temp)  
        return temp;
```

All function return errors in
SVN are logged!

propagate to caller

return an
error message

/* SVN */

Pattern II: abnormal exit

No log:

A bug triggered
this abort

Over 10 discussion messages
btw. user and dev.

```
if (svn_dirent_is_root)
```

```
abort();
```

```
+ svn_error_raise_on_malfunction(_FILE_, _LINE_);
```

```
+ svn_error_raise_on_malfunction(..) {
```

```
+   err=svn_error_create("In file '%s' line '%d': internal malfunction");
```

```
+   svn_handle_error2 (err);
```

```
+   abort();
```

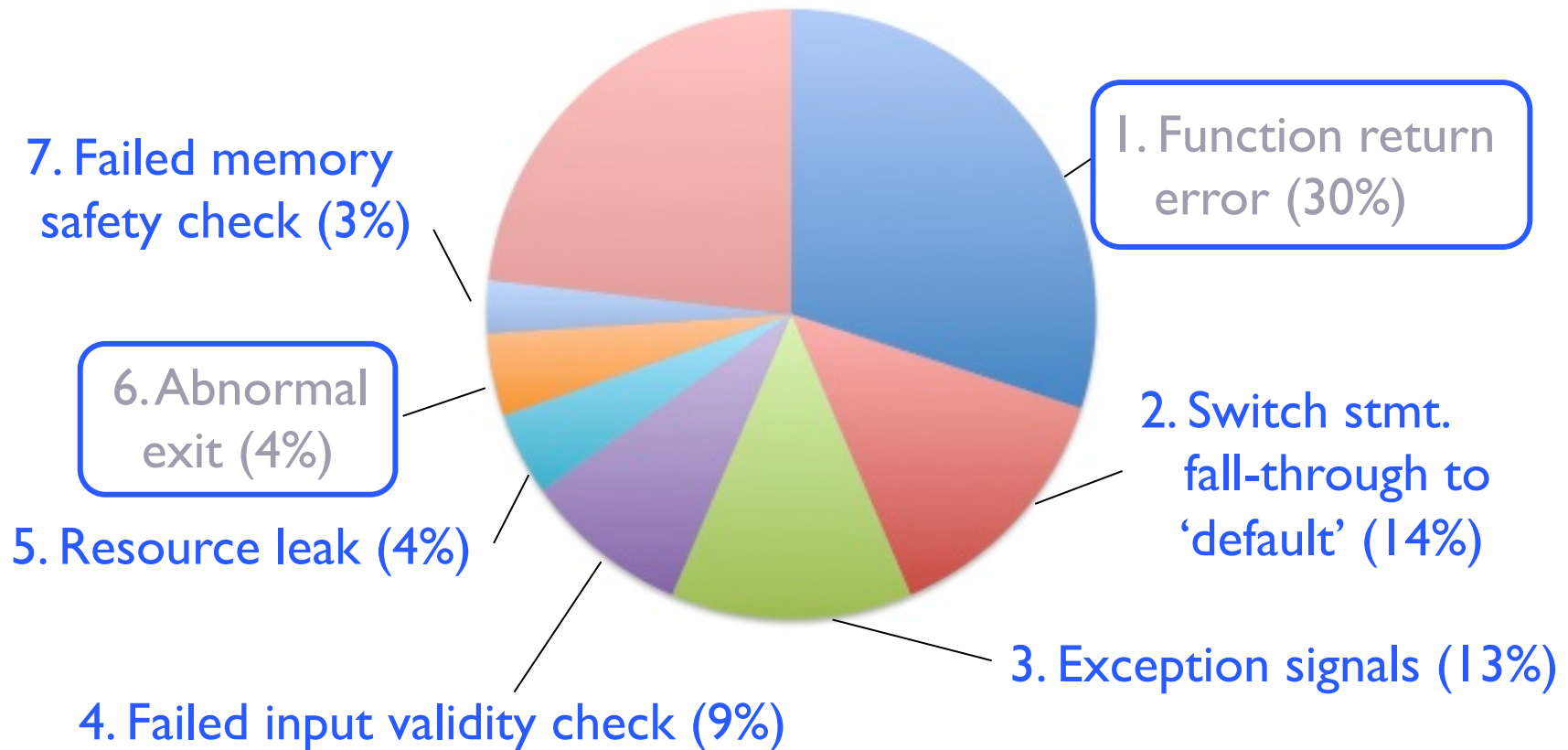
```
+ }
```

print error
message.

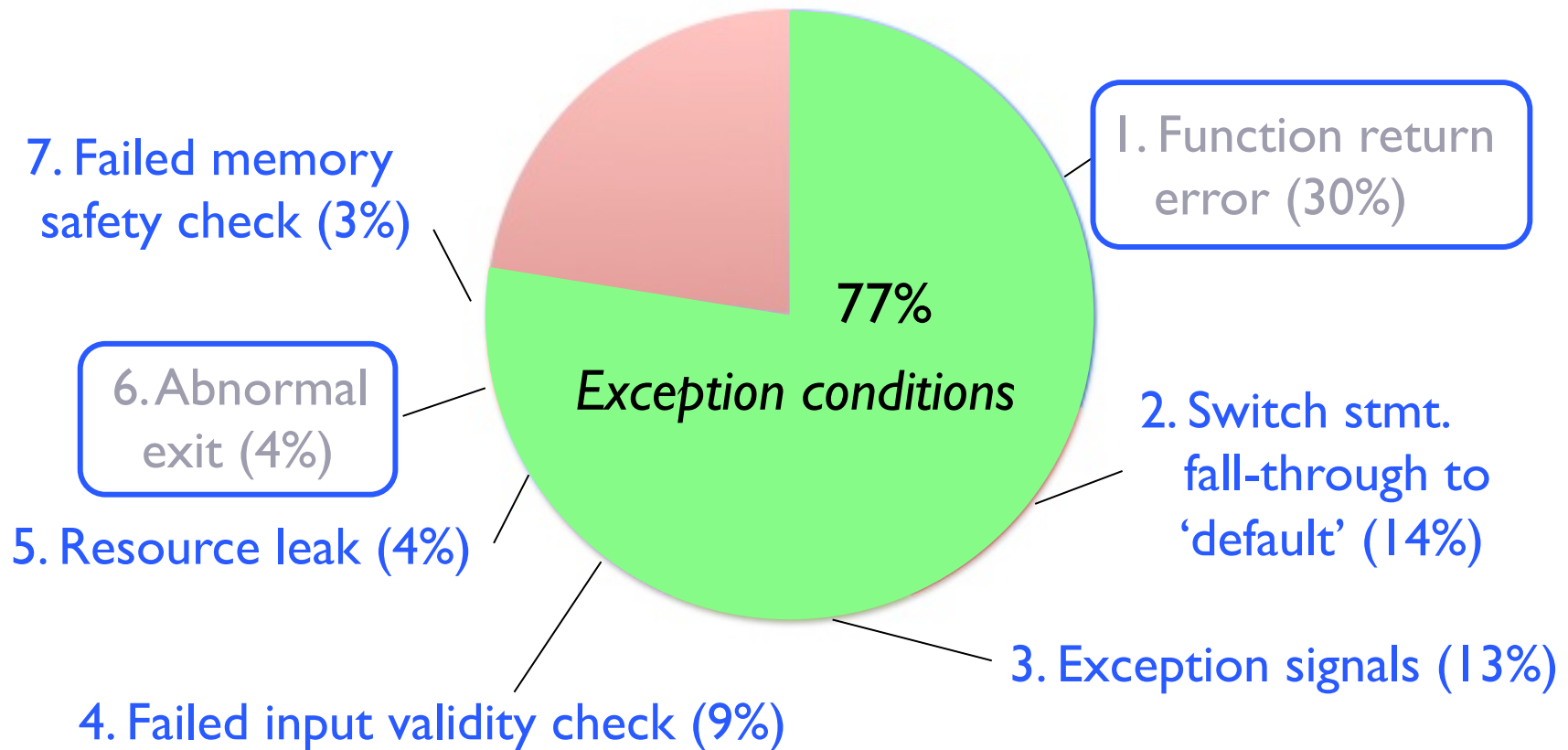
*"I really hate abort() calls! Instead of getting a
usable core-dump, I often got nothing."
— developer's check-in message*

/ SVN */*

Generic log-worthy patterns

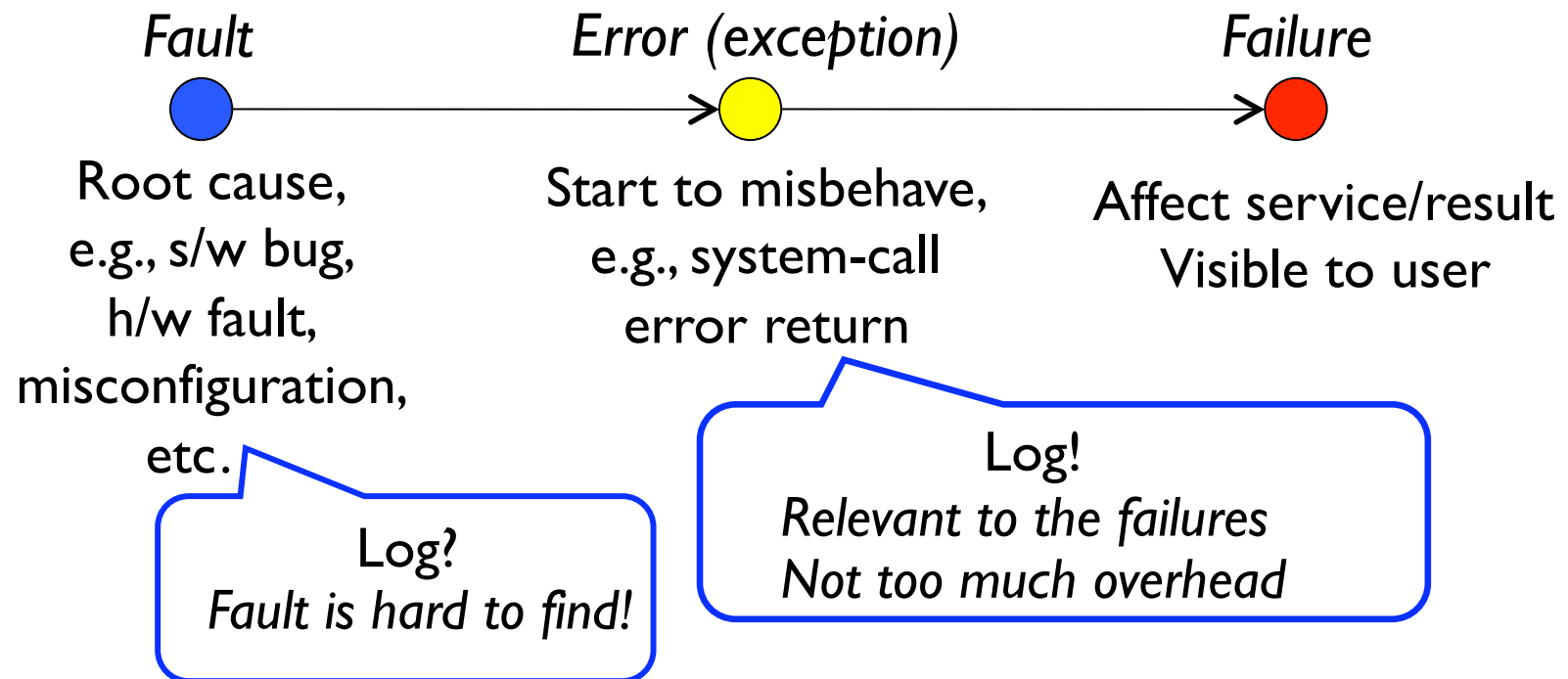


Generic log-worthy patterns

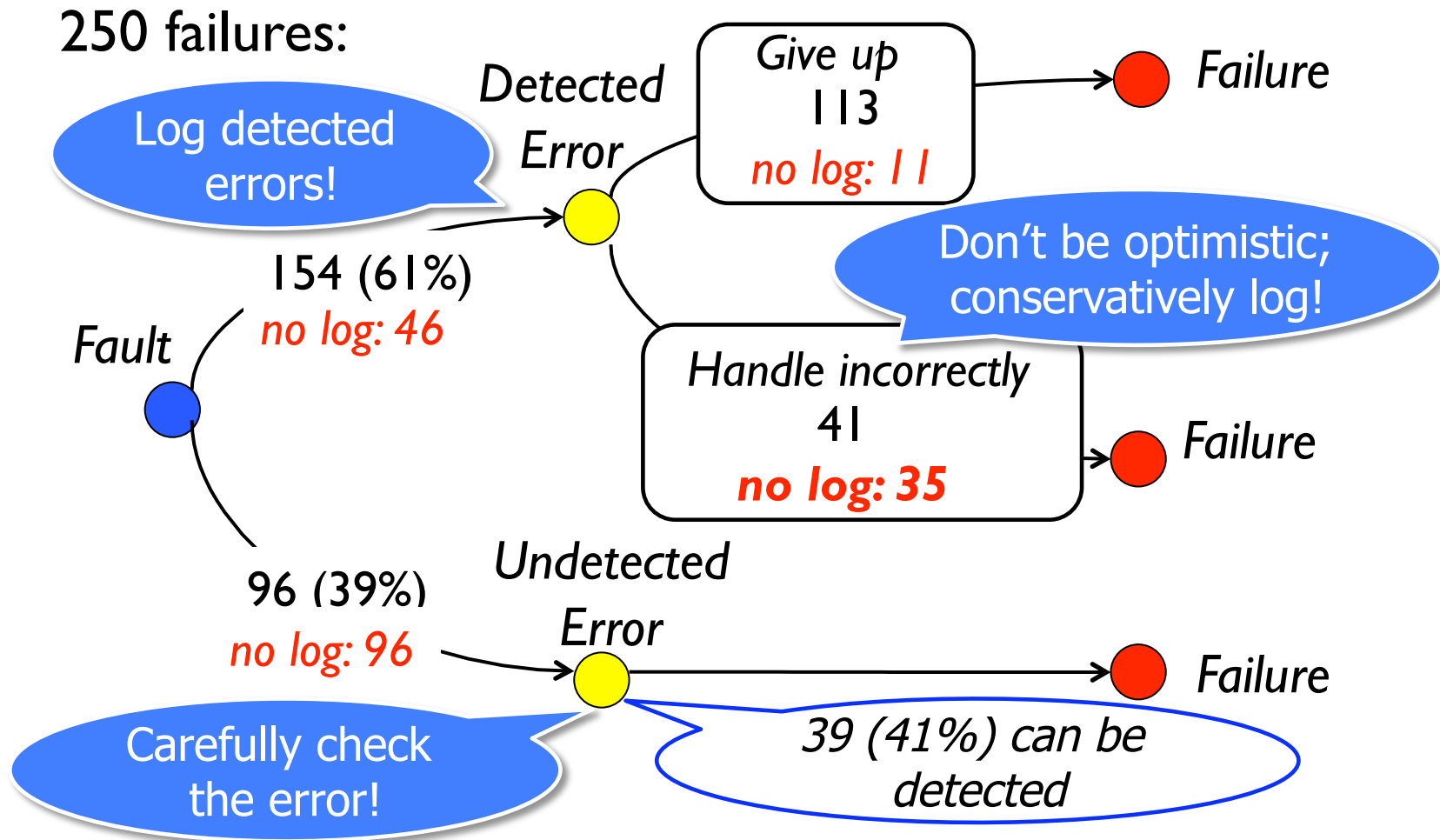


Log the exception

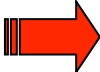
► Classic Fault-Error-Failure model [Laprie.95]



Why developers missed logging?



Outline

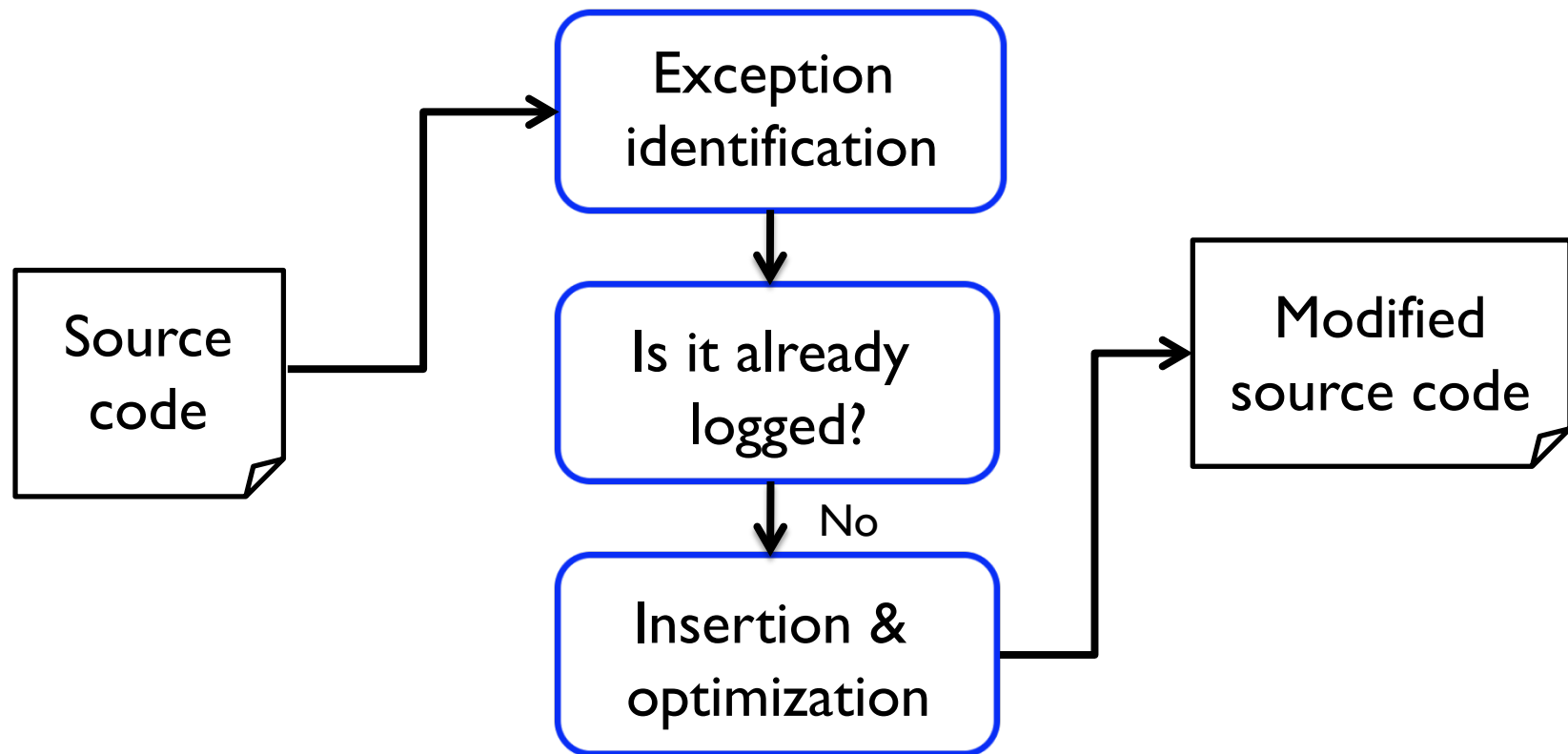
- ▶ Introduction
- ▶ Characterizing logging efficacy
- ▶  *Errlog* design
- ▶ Evaluation results
- ▶ Conclusion



automate such
logging

Errlog: a practical logging tool

```
errlog -logfunc="error" CVS/src
```




Exception identification

- ▶ Mechanically search for generic error conditions
- ▶ Learn domain-specific error conditions

Frequently logged

```
if (status != COMM_OK)
    goto ERROR;
.. ..
ERROR:
    error("network failure");
```



```
if (status != COMM_OK){
+   elog ();
}
```

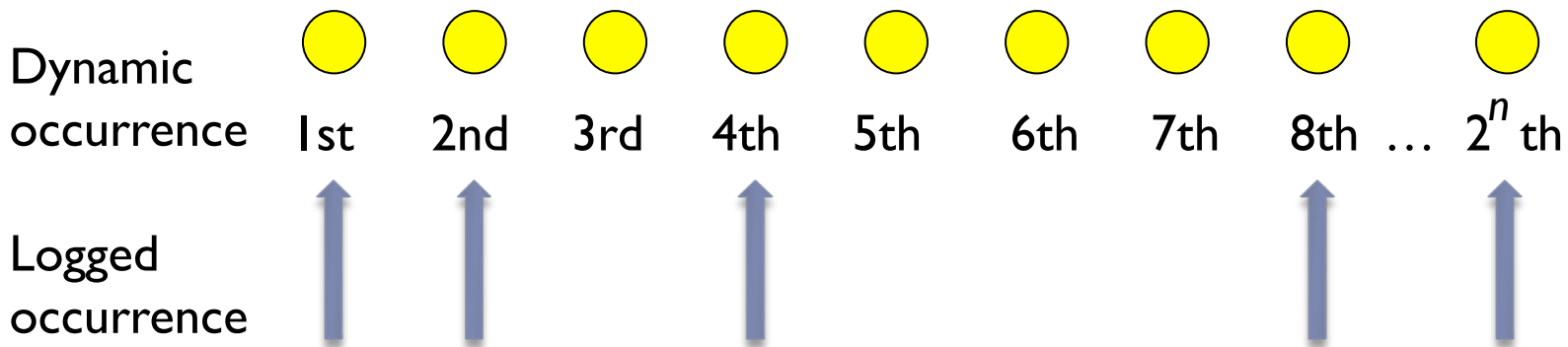
Log statement insertion

- ▶ Check if the exception is already logged
 - ▶ Each log statement contains:
 - ▶ Unique log ID, global counter, call stack, useful variables
- LogEnhancer [TOCS'12]

```
/* Errlog modified code */  
if (status != COMM_OK) {  
+   elog (logID, glob_counter, logEnhancer());  
}
```

Adaptive sampling to reduce overhead

- ▶ Not every identified condition is a true error
 - ▶ E.g., using error return of 'stat' to test the existence of file
- ▶ Adaptive sampling [HauswirthASPLOS'04]
 - ▶ More frequently it occurs, less likely to be a true error
 - ▶ Differentiate run-time log by call stack and errno



Evaluation

- Applied *Errlog* on ten software projects

Software	LOC
Apache httpd	317K
Squid	121K
PostgreSQL	1029K
SVN	288K
Coreutils	69K

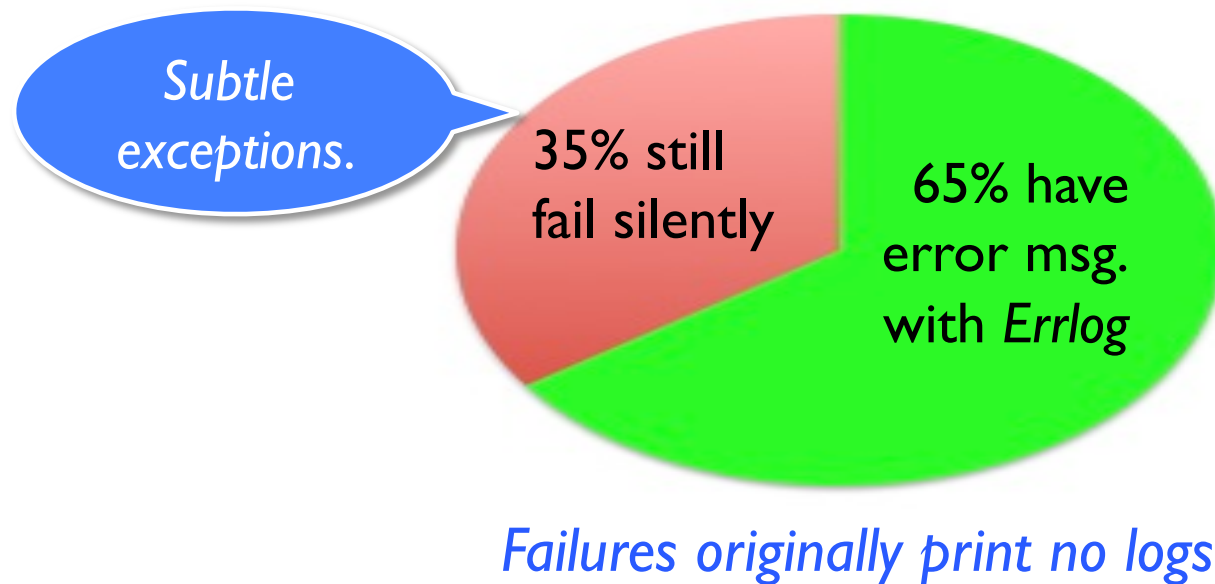
Software used in our
empirical study

Software	LOC
CVS	111K
OpenSSH	81K
Lighttpd	56K
gzip	22K
GNU make	29K

New software not used
in our empirical study

Reducing silent failures

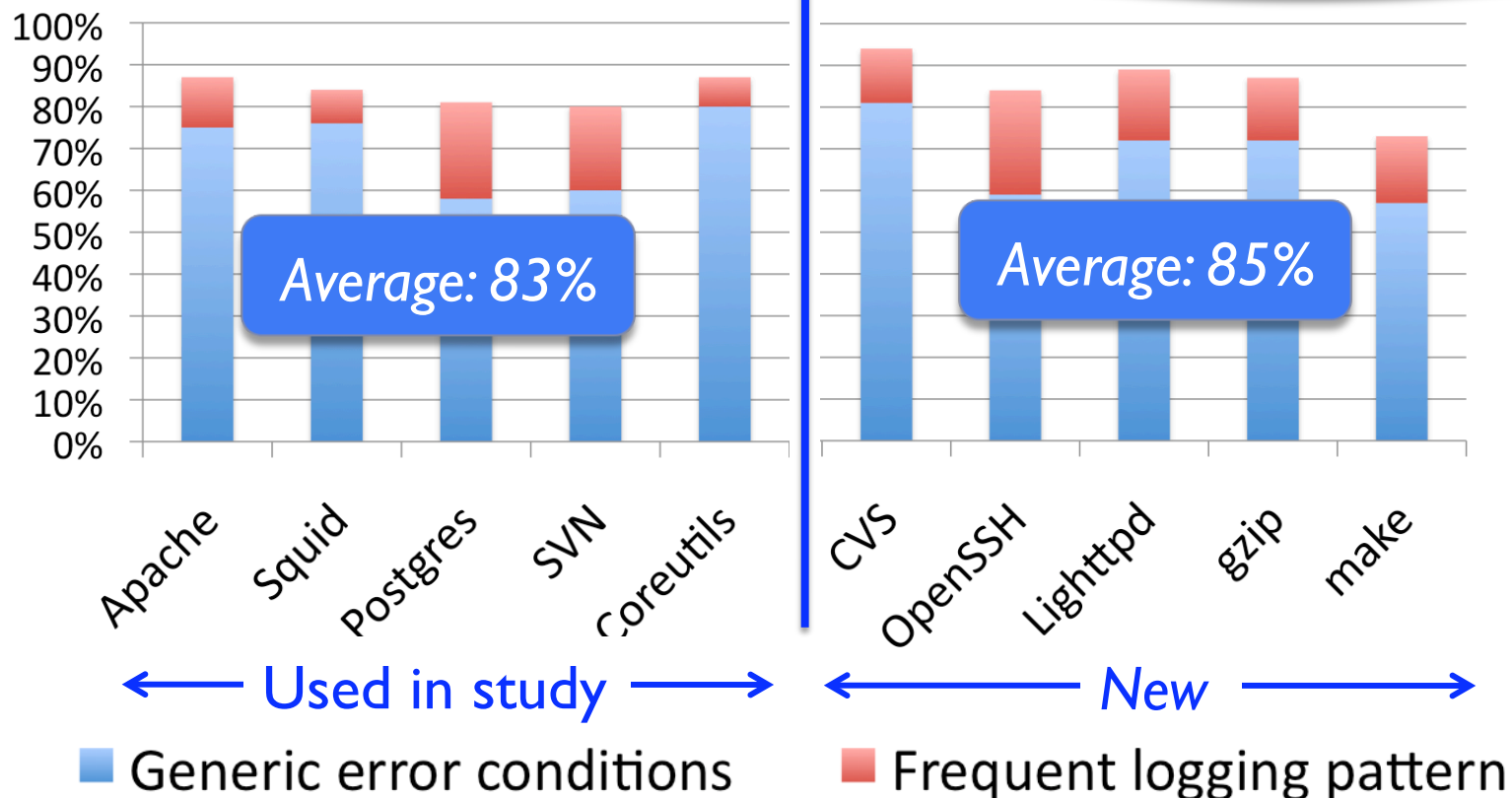
- ▶ *Errlog* adds 0.60X extra log printing statements
- ▶ What is the benefit?
 - ▶ Evaluated on 141 silent failures



Comparison with manual logging

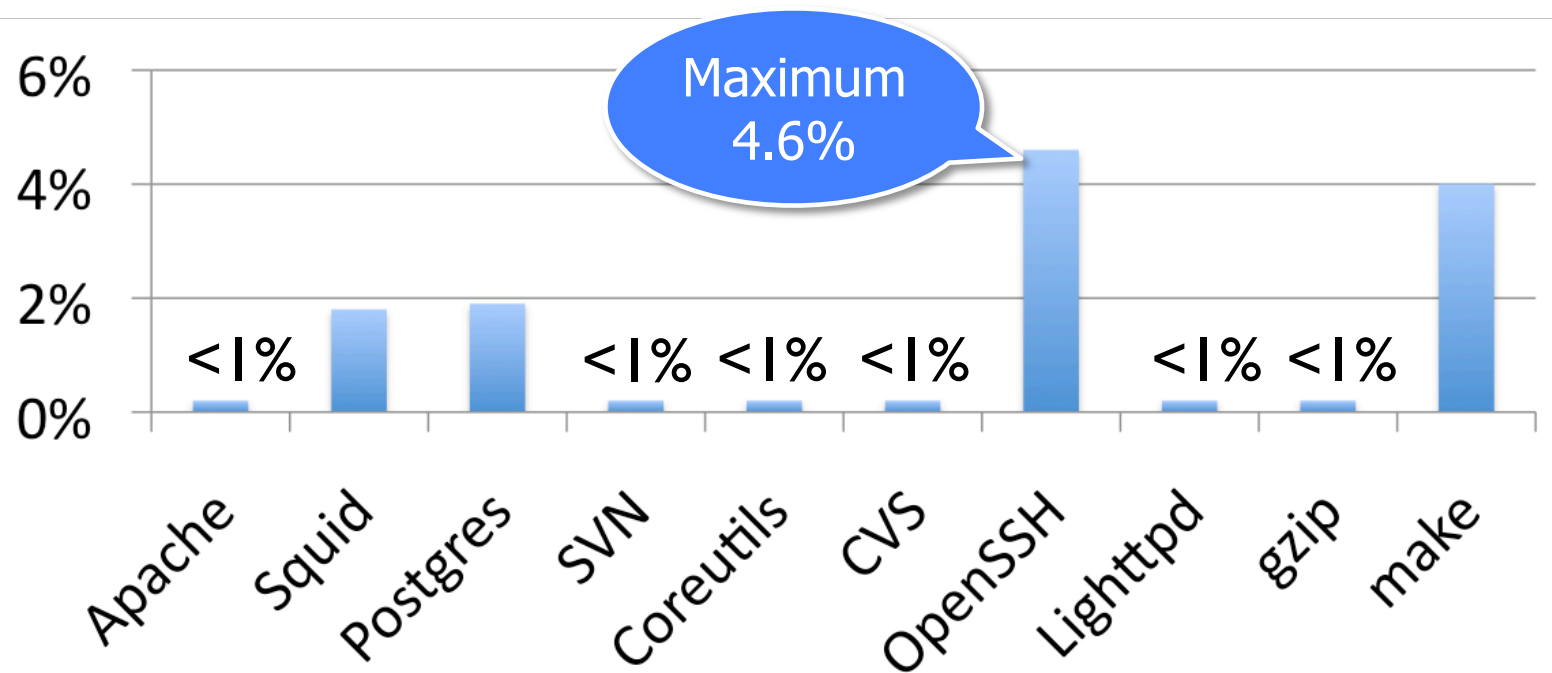
► 16,065 existing log stmt. in ten systems

► Many added reactively



Performance overhead

- ▶ Why *Errlog* has overhead?
 - ▶ A few noisy log messages in normal execution
- ▶ *Errlog* adds 1.4% overhead



User study

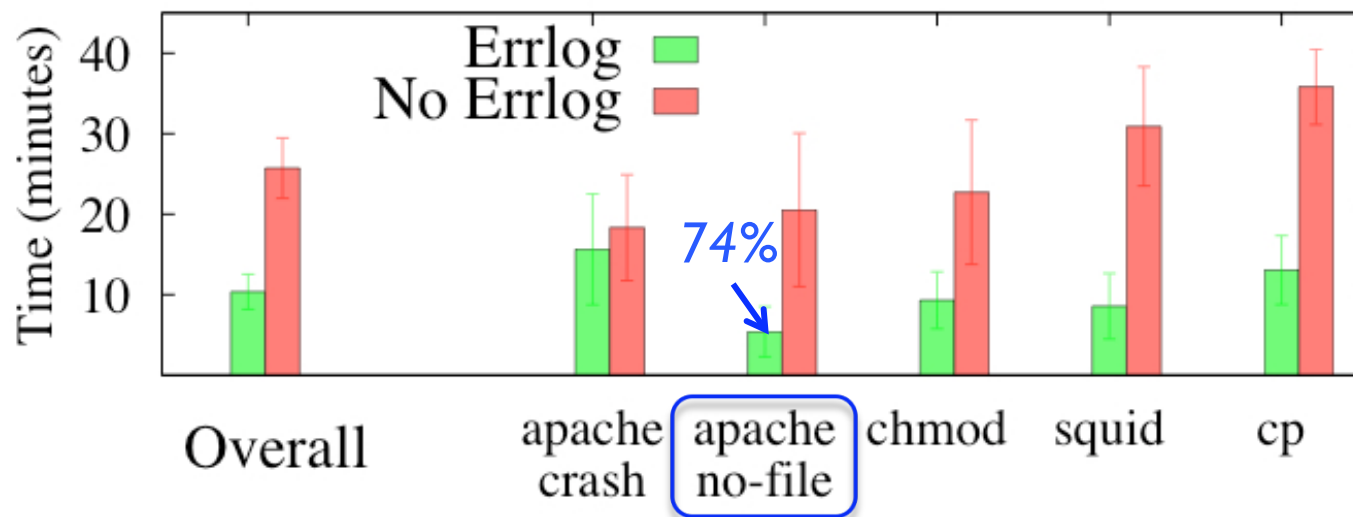
- ▶ 20 programmers from UCSD
- ▶ 5 *real-world* failures

GDB can be used.

Failure	Repro?	Description
apache crash	Yes	NULL ptr. dereference caused by user misconfiguration.
apache no-file	Yes	Misconfiguration caused apache cannot find the group-file
chmod	No	Silently fail on dangling symbolic link
cp	Yes	Fail to copy the content of /proc/cpuinfo
squid	No	Denies user's valid authentication when using an external authentication server

User study result

- ▶ On average, *Errlog* reduces diagnosis time by 61%



“(Errlog added) logs are in particular helpful for debugging complex systems or unfamiliar code where it required a great deal of time in isolating the buggy code path.”

— from a user’s feedback

Limitations

- ▶ Study result might not be representative
 - ▶ Only five software projects
 - ▶ All written in C/C++
- ▶ Not all failures can benefit from *Errlog*
 - ▶ Still 35% of the silent failures remain silent
- ▶ Semantic of the log message is not as good

Related work

- ▶ **Detecting bugs in exception handling code**
[RenzelmannOSDI'12][GunawiFAST'08][GonzalesPLDI'09]
[MarinescuTOCS'11][GunawiNSDI'11][YangOSDI'04]
 - ▶ Different: logging instead of bug detection
 - ▶ Complementary: exception patterns can benefit previous work
- ▶ **Deterministic replay** [VeeraraghavanASPLOS'11][AltekarSOSP'09]
[DunlapOSDI'02][SubhravetiSIGMETRICS'11]
 - ▶ Overhead and privacy
- ▶ **Log enhancement** [Yuan TOCS'12][Yuan ICSE'12]
 - ▶ Unique challenges: Shooting blind and overhead
 - ▶ Different approaches: failure study, exception identification, check if exception is logged, adaptive sampling, etc.

Conclusions

- ▶ Many obvious exceptions are not logged
 - ▶ *Carefully write error checking code*
 - ▶ *Conservatively log the detected error, even when it's handled*
- ▶ **Errlog: practical log automation tool**
 - ▶ User study: *Errlog* shortens the diagnosis by 61%
 - ▶ Adding only 1.4% overhead

"As personal choice, we tend not to use debuggers beyond getting a stack trace or the value of a variable... We find stepping through a program less productive than thinking harder and adding output statements and self-checking code at critical places. More important, debugging statements stay with the program; debugging sessions are transient."

Thanks!

— Brian W. Kernighan and Rob Pike
“The Practice of Programming”

Failure diagnosis reports can be found at:
<http://opera.ucsd.edu/errlog.html>