Recovery Domains

Andrew Lenharth
University of Texas

Vikram Adve, Sam King
University of Illinois

Recovery as a Security Concern

Before SVA: Any exploitable kernel bug allows application complete access to the system.

After SVA: Any exploitable kernel bug brings down the system.

Recovery:

- Increases Availability
- Limits Damage

Recovery Domains

Recover from detected faults in commodity operating systems

Keep operating system state consistent after fault

Limit fault visibility to offending action

Error as a property of a request

Handles transient, timing, and permanent errors

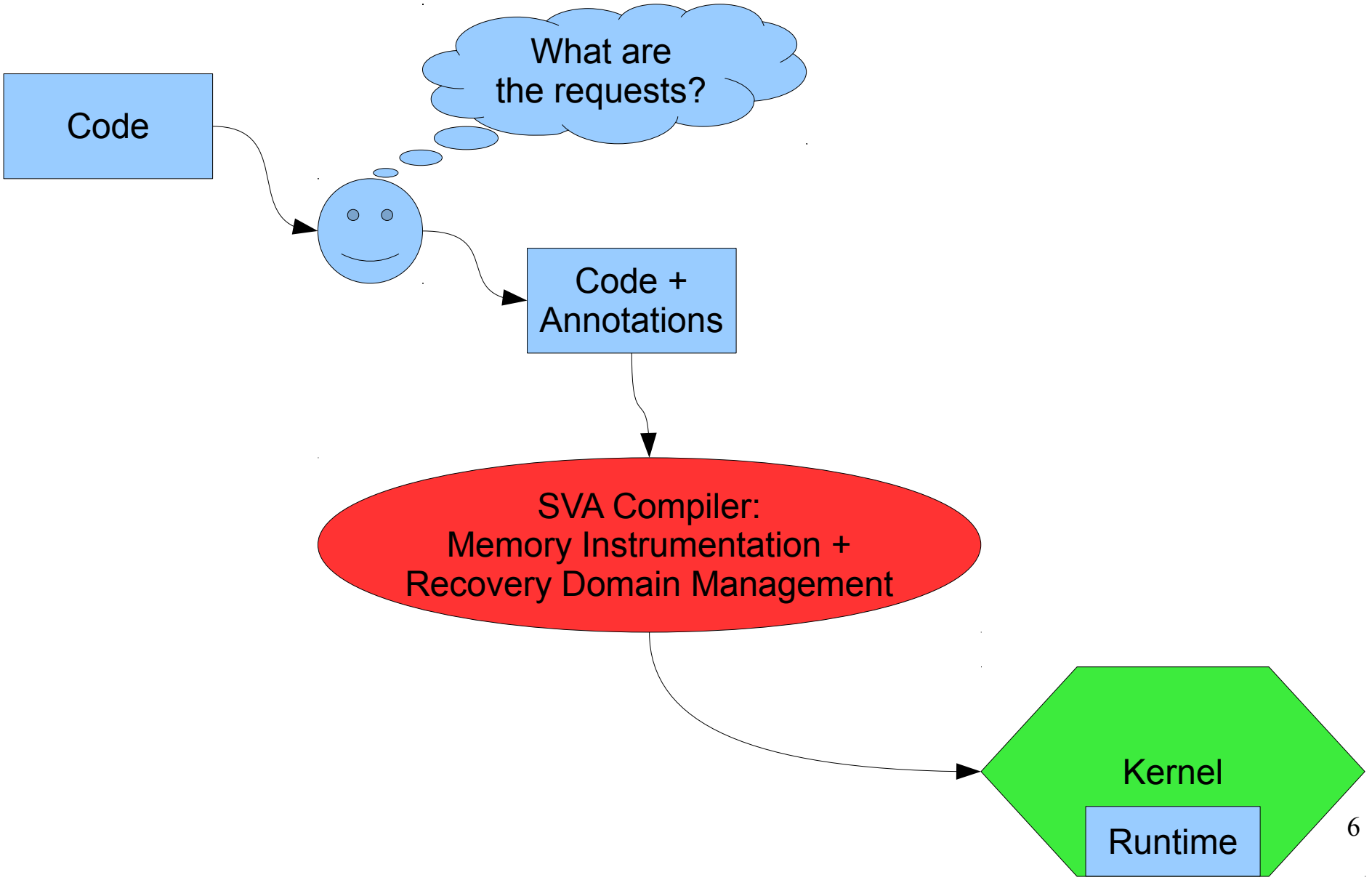
Not statically definable code or heap regions

Ties an error to a logical operation not a subsystem

Requests

- System calls
- Interrupts
- Common calls across subsystems

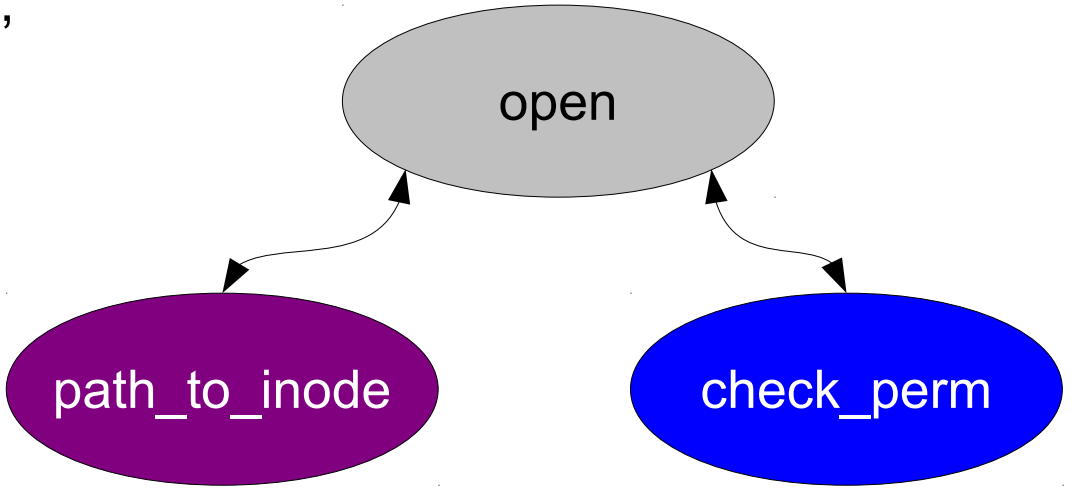
Compiling


What are
the requests?
Code
Code +
Annotations
SVA Compiler:
Memory Instrumentation +
Recovery Domain Management
Kernel
Runtime

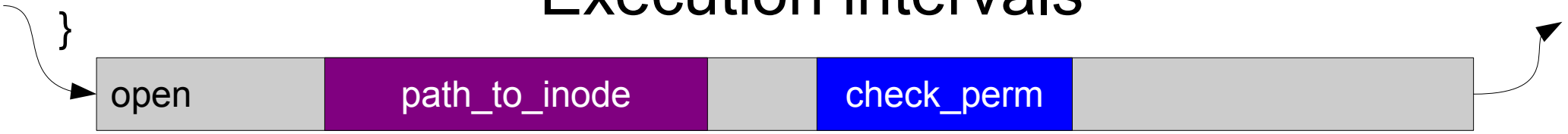
Example

```
int open(char* path) {
  domain_begin();
  inode* n =  path_to_inode(path);
  if(!check_perm(n)) {
     domain_end();
     return false;
  }
  int r = open_fid(n);
  domain_end();
  return r;
}
```

Dependency Graphs



Execution intervals



OS View

Recovery from detected faults

No fault-free change in semantics

- Fine-grain locking *ok*
- Thread communication *ok*

Almost no programmer effort

- Request boundary annotations
- SVA-based compile time instrumentation and transformations

Application View

Visibility – OS notifies application after a fault

- Intelligently continue

Correctness – OS state as-if request didn't happen

- Future actions will yield correct results

Isolation – Faults not visible between applications

- Cannot use corrupted OS state to undermine security

Porting Linux

2.6.27

~600 Lines of Code

- Replace most inline assembly with equivalent C + gcc intrinsics
- Annotate Recovery Domains & locks

2.4.22

132 Lines of Code

- Annotate Syscalls and Interrupts
- Annotate Allocators
- Spinlocks replaced with version in runtime
  - Convert dataflow into control flow

Results - Survival

Inject fault every 300k + rand(0,300k)

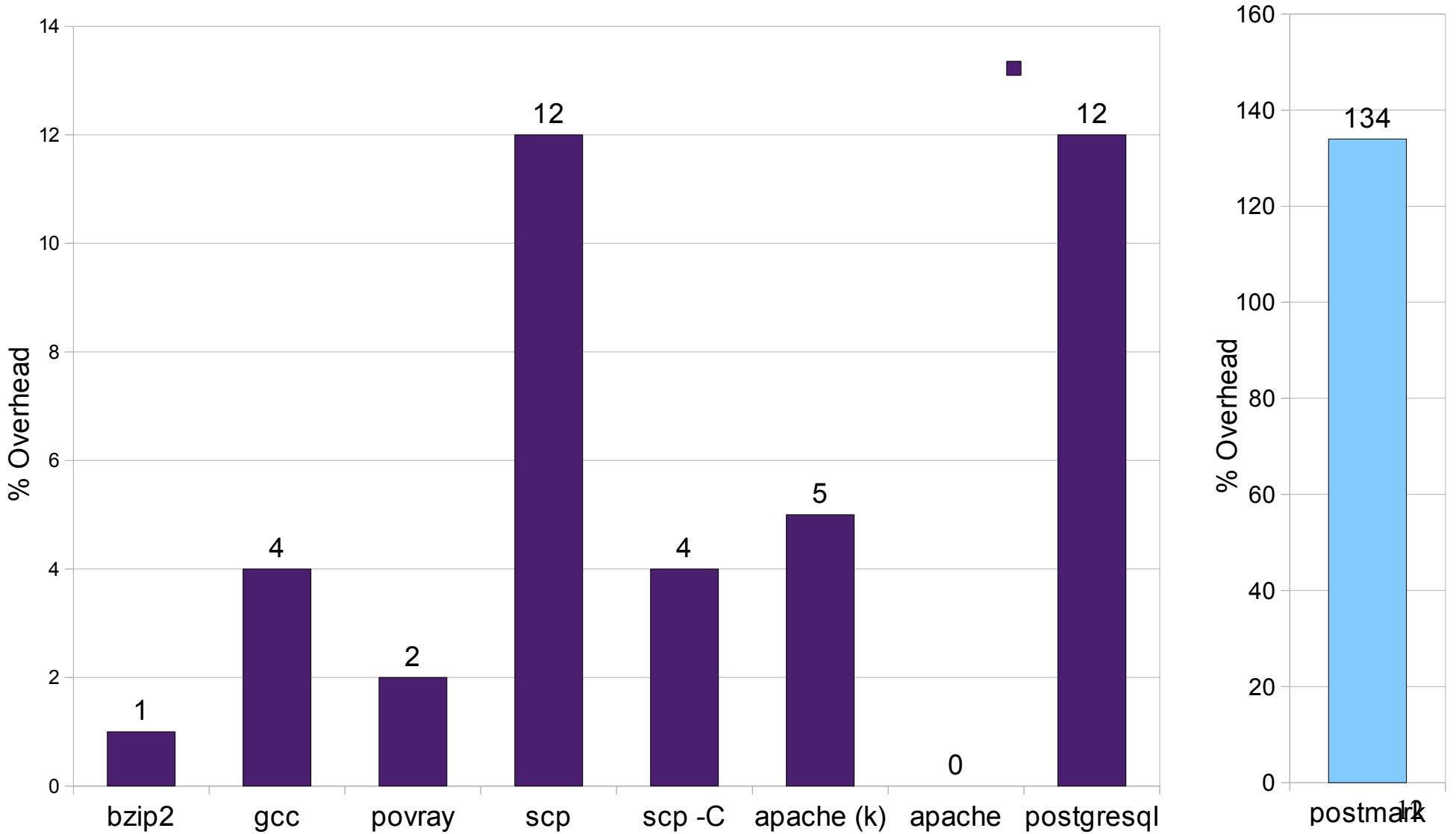
Start Apache and inject faults

– Avg 21 consecutive successful recoveries

Start Postgresql and inject faults

– Avg 17 consecutive successful recoveries

Results - Performance



Recovery Domains

Low level recovery, high level semantics

Key Ideas:

- SVA makes whole OS transforms and analysis “easy”
- Logical requests as unit of recovery
- Transactional memory techniques
- Error virtualuzation

Transactions

- Transaction-like
  - Failure atomicity
- Not transactions
  - No isolation
  - No additional durability
  - Consistency assumed to be memory state
  - May not commit or abort at end of domain

A C ~~I~~ ~~D~~

Summary of Implementation

Significant complications:

- Logging
- Versioned memory
- Dependence graphs
- Register and stack state saving
- Lock and allocation optimizations