Problem: Security is too expensive

Systems software is written in unsafe languages (for performance reasons)

```
char c = *(buf + offset);
```

⇒ SIGSEGV, ...

Security is retrofitted through instrumentation

```bash
$ gcc -fsanitize=address prog.c -o safeprog
```

automatically generated security check

Performance suffers

```
prog:
```

```
safeprog:
```

typically: 74% overhead, of which

87% due to checks

13% due to other sources

Solution: Given an overhead budget, maximize security

User specifies tolerable overhead

"I want bzip2 with < 10% overhead."

Our tool ASAP automatically recognizes security checks in program bitcode

ASAP performs a cost analysis for every check

ASAP selects checks that maximize security for the desired overhead level.

overhead: **9.2%**

security: > **90%**

Result of optimizing bzip2 with ASAP

Key insights

**Overhead** is dominated by a handful of expensive checks

- Original program runtime
- Metadata management (e.g., keeping track of allocated memory)
- Top 10% most expensive checks
- 90% remaining cheap checks

13% of overhead

81%

6%

**Security** is provided primarily by many cheap checks

Our experiments show that:

- 97% of all memory-related CVE vulnerabilities are in cold code, where checks are cheap
- Checks in buggy code are colder than checks in stable code

Experimental results

Existing tools like AddressSanitizer have a high overhead (avg: 74%)

With ASAP:

Choose perfect **overhead/security** trade-off

ASAP strongly reduces overhead while preserving most of the security

Achieves target overhead of < 10% for 10 out of 15 SPEC benchmarks

ASAP prevents the Heartbleed vulnerability with only 5% reduction in OpenSSL throughput