Early detection of configuration errors to reduce failure damage

Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, Shankar Pasupathy

UC San Diego *University of Chicago* *NetApp*

This paper is ~~not about bugs~~ about configuration errors.

- bad values inside configuration files
- introduced by sysadmins
- nothing “wrong” in our code

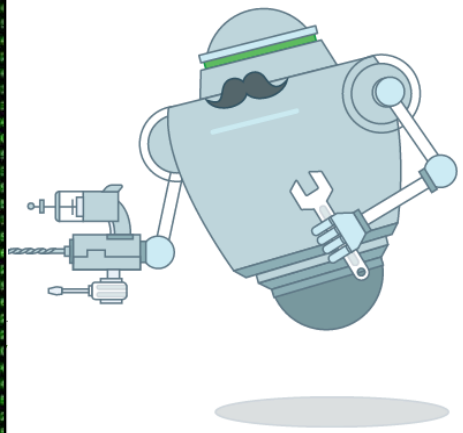
When systems use bad configuration values, the code does report errors.

- throw exceptions
- return error code
- crash with coredumps

correct != timely

Errors are often reported too late!

```
/* sys_24-7.c */

signal(SIGSEGV, call_techsup);

static void call_techsup(int sig) {
  if (fork() == 0) {
    char* args[] = {“0911”, “SOS”};
    int rv = execvp(dial_prog_path, args);
    if (rv != 0)
      fprintf(stderr, “I’m sorry (%d)!”, errno);
  }
}
```



Shoot!

Well, this is unexpected…

Error code: 500

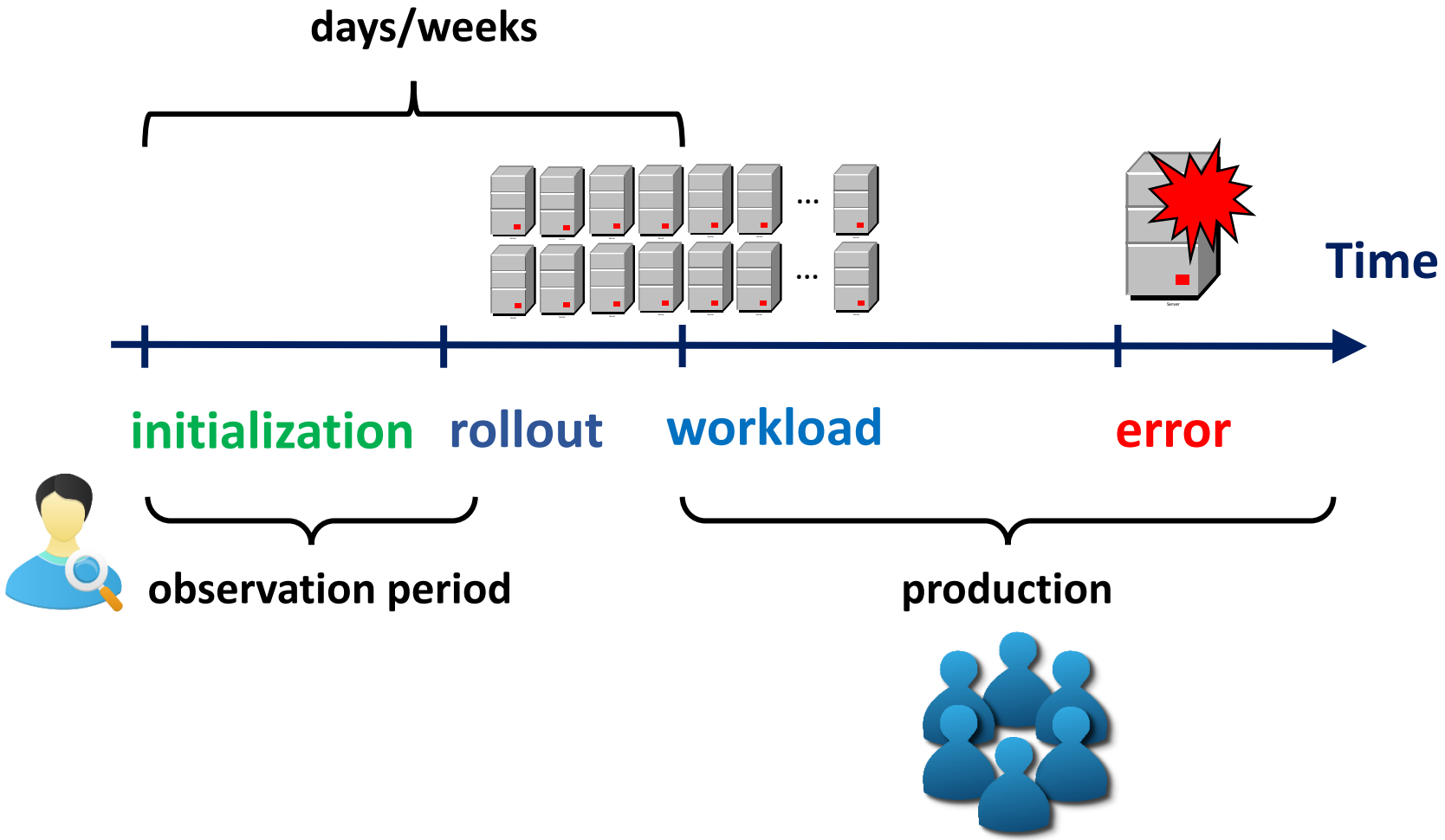
An error has occurred and we're working to fix the problem!
The service will be up and running shortly.

```
static void call_techsup(int sig) {
  if (fork() == 0) {
    char* args[] = {"0911", "SOS"};
    int rv = execvp(dial_prog_path, args);
    if (rv != 0)
      fprintf(stderr, "I'm sorry (%d)!", errno);
  }
}
```

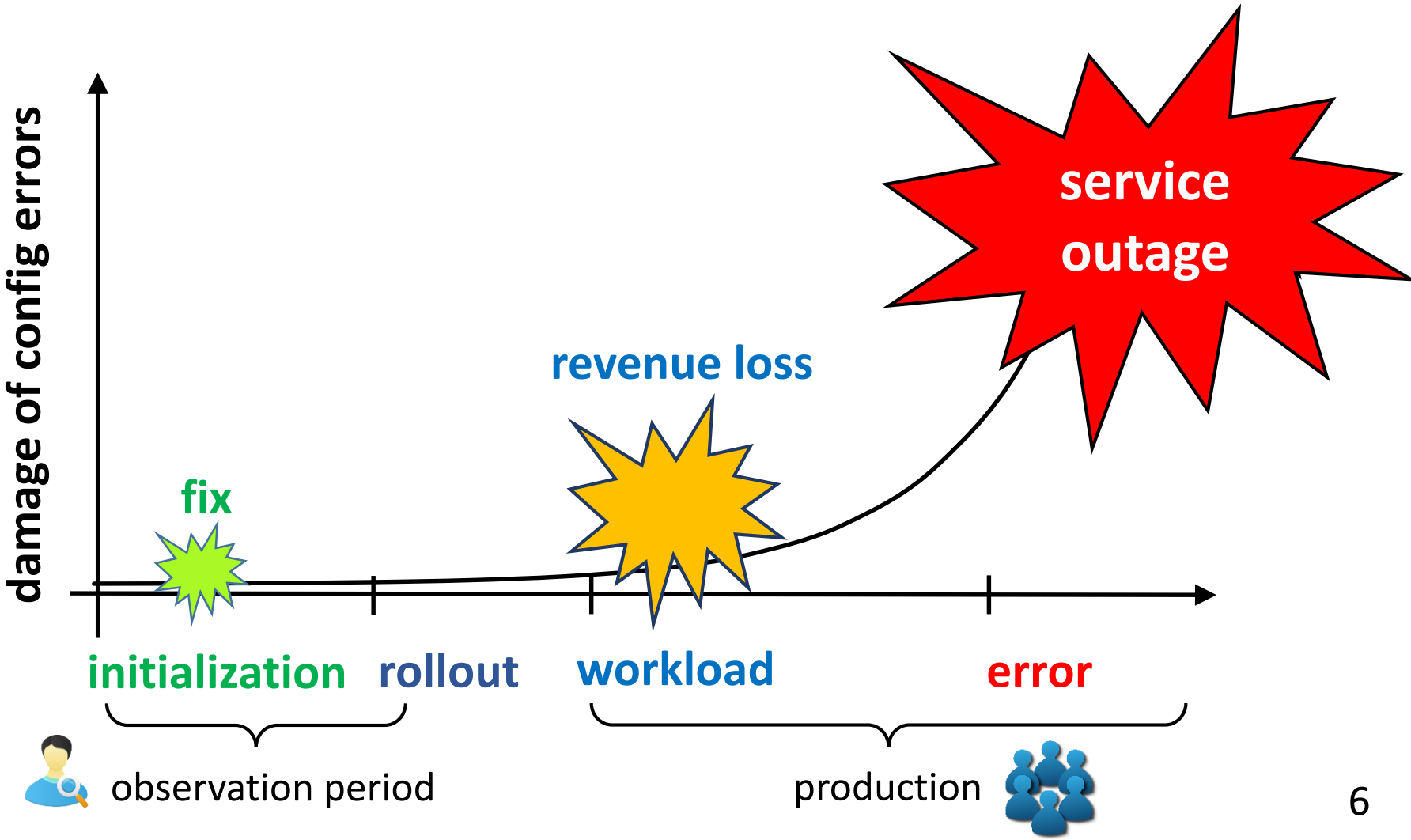
ERROR

"It's too late to apologize!"

Systems execution has stages.



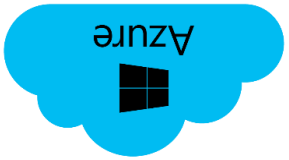
All stages are not created equal.



Does this truly happen?

Faulty failover configurations turned a 10 minute outage into a 2.5 hour ordeal.

Misconfigured backup DNS (used upon attacks) made LinkedIn inaccessible for half a day.



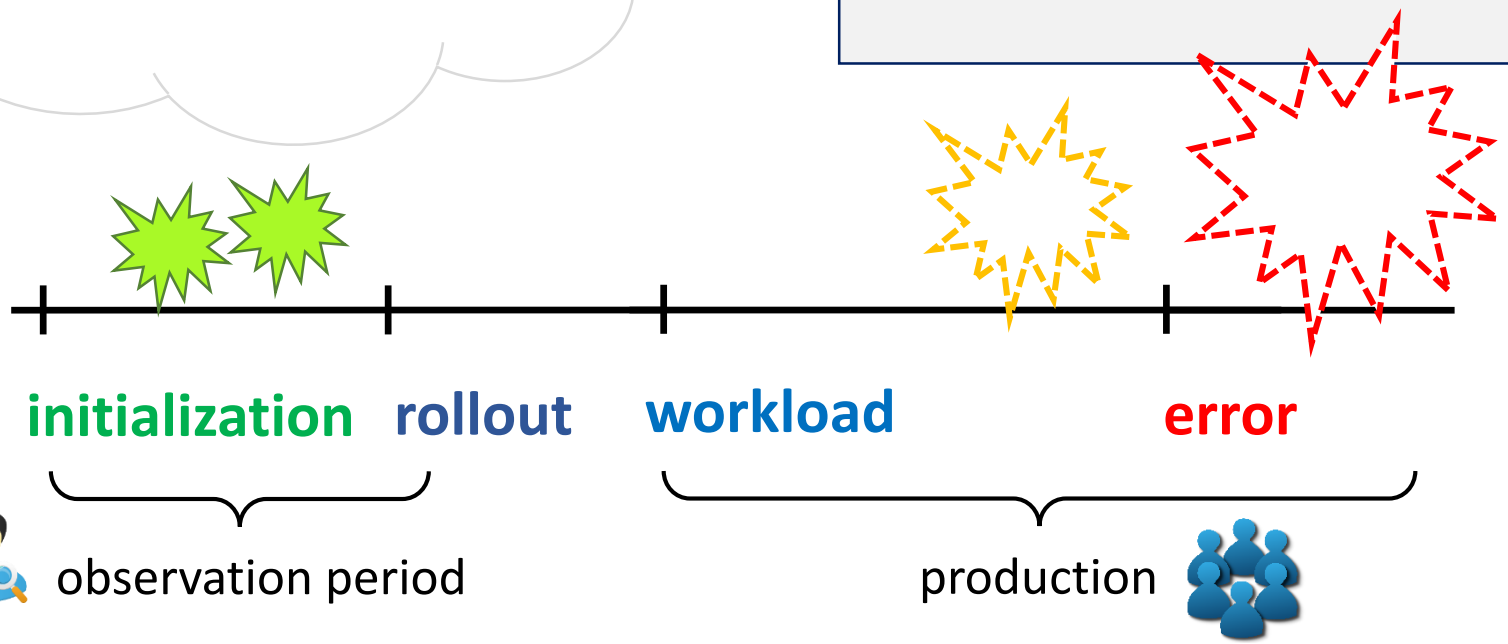
Misconfigured data protection allowed a bug to wipe out 10% of the storage nodes.

Sysadmins' wish

All the configuration errors can be exposed at **initialization**.

Reality

Difficult for sysadmins to test out **latent** configuration errors ☹



Contribution

- **A perspective of checking configurations early and detect errors timely**
- **A study on real-world configuration checking practices**
  - deficiency of built-in configuration checks
  - prevalent threats of **latent** configuration errors
- **PCheck: tooling support for automatically generating configuration checking code**
  - help systems detect configuration errors **early**
  - effective, safe, and efficient

How ~~well~~ terribly do systems check their configurations?

- Studied configuration parameters of R.A.S. (**R**eliability, **A**vailability, **S**erviceability) features

Software	RAS Param.
HDFS	44
YARN	35
HBase	25
Apache	14
Squid	21
MySQL	43

- **mission critical**
- **not** really **needed** for initialization
  - 12%–39% are **not** used during in[illegible]tion

w/o early checking, errors would become latent & catastrophic.

How ~~well~~ terribly do systems check their configurations?

- 14%–93% of the studied parameters do **not** have any special checking code at initialization

  → *rely on usage code for checking/reporting errors*

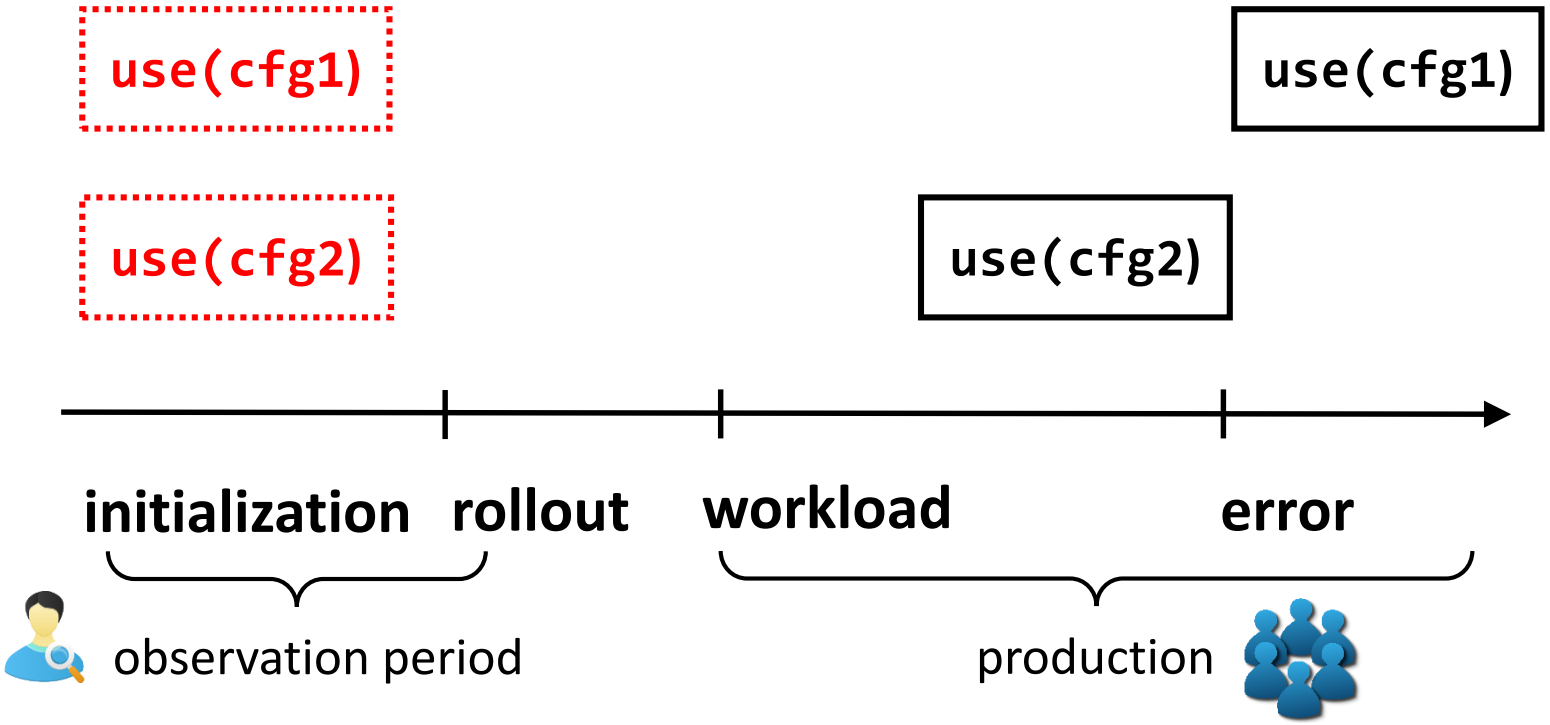
5%–39% of R.A.S. parameters are subject to latent errors.

Detecting latent configuration errors would require separate checking code at systems' initialization phase.

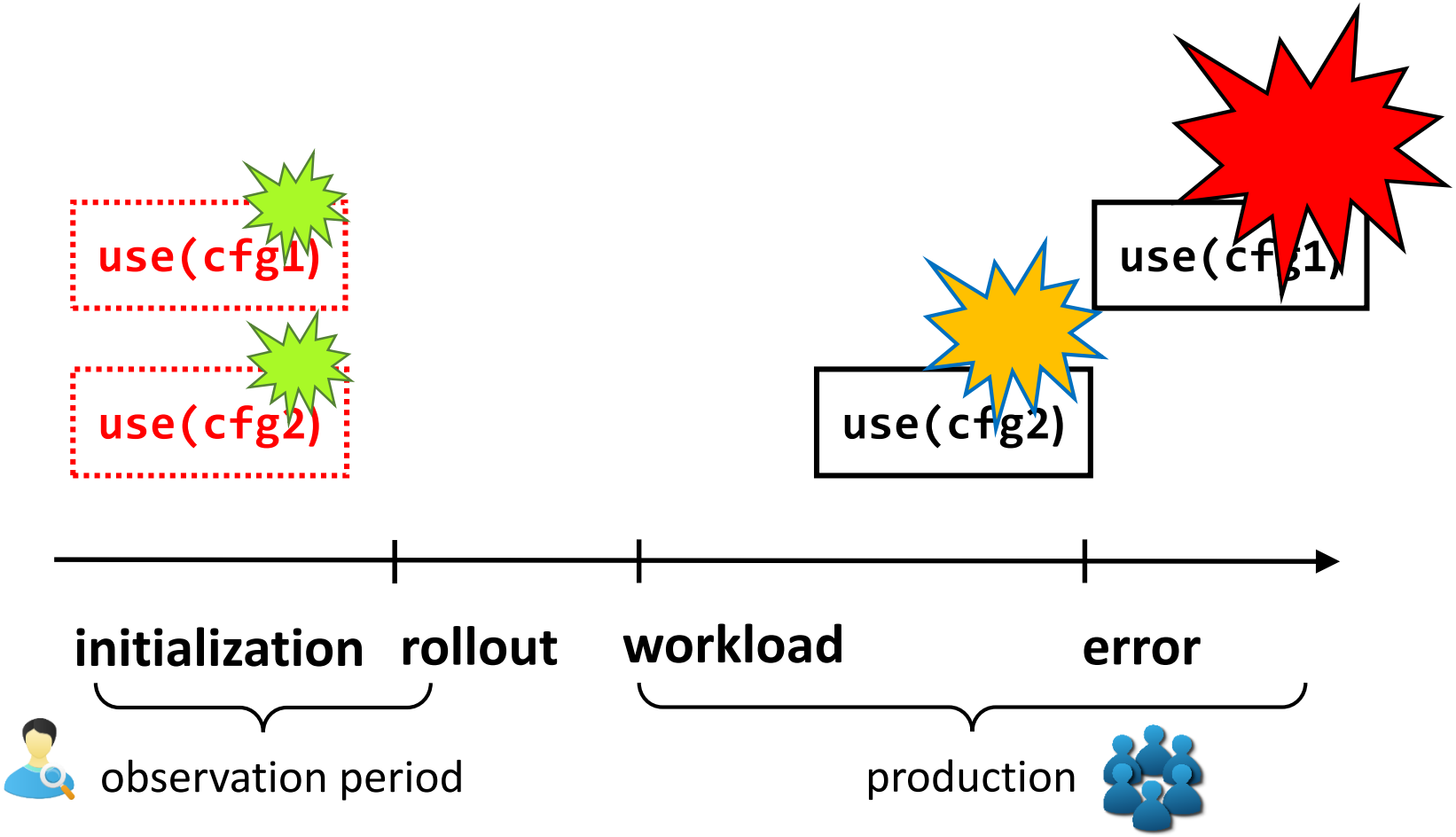
Systems already have checking logic implied by the usage of configuration values (though usage code often comes late).

Can we leverage usage-implied checking to detect configuration errors early?

Why not copy+paste the code that uses configuration values into initialization?

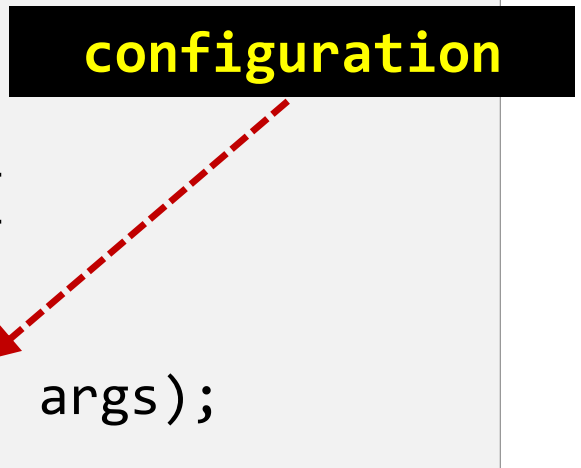


Why not copy+paste the code that uses configuration values into initialization?



Demo

```
/* sys_24-7.c */

signal(SIGSEGV, call_techsup);

static void call_techsup(int sig) {
  if (fork() == 0) {
    char* args[] = {“0911”, “SOS”};
    int rv = execvp(dial_prog_path, args);
    if (rv != 0)
      fprintf(stderr, “I’m sorry (%d)!”, errno);
  }
}
```



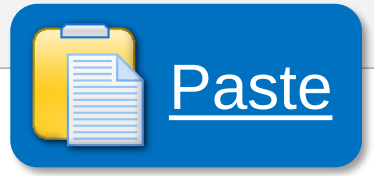
Demo

```
/* sys_24-7.c */

signal(SIGSEGV, call_techsup);

static void call_techsup(int sig) {
  if (fork() == 0) {
    char* args[] = {“0911”, “SOS”};
    int rv = execvp(dial_prog_path, args);
    if (rv != 0)
      fprintf(stderr, “I’m sorry (%d)!
  }
}
```

configuration

Demo

```
/* sys_24-7.c */

static int sys_init() {
  load_config();
  ...
  ...
  ...
  int rv = execvp(dial_prog_path, args);
}
```



Copy+paste code does not work!

```
int rv = execvp(diag_prog_path, args);
```

Problem 1 **Executing code needs context**

- `args` is undefined

Problem 2 **Execution can have side effects**

- prank calls are **crimes**.
- `exec()` removes the current process image

Produce necessary context

```
+     char* args[] = {“0911”, “SOS”};
      int rv = execvp(diag_prog_path, args);
```

- ✓ **Backtrack to determine values of undefined variables**
  - best effort: may not always be able to determine the values
  - configurations often have relatively simple context

Prevent side effects

```
-     char* args[] = {“0911”, “SOS”};
      int rv = execvp(diag_prog_path, args);
```

```
      int rv = check_execvp(diag_prog_path);
```

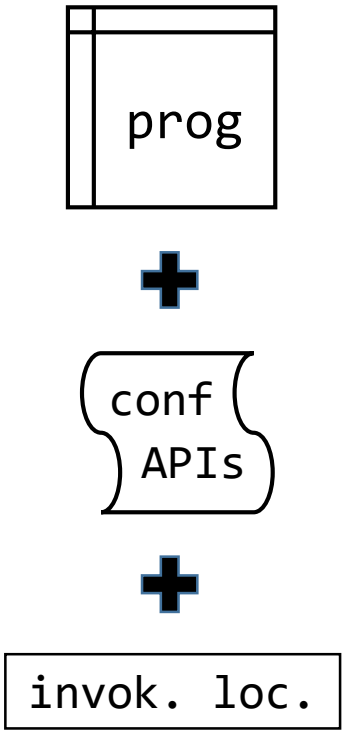
- ✓ **“Sandbox” the checking code**
  - Rewrite instructions based on ***check utilities*** that validate the operands w/o executing the instructions

PCheck implementation

- **Works for both C and Java programs**
  - LLVM compiler framework for C code
  - Soot compiler framework for Java code

PCheck implementation

Input

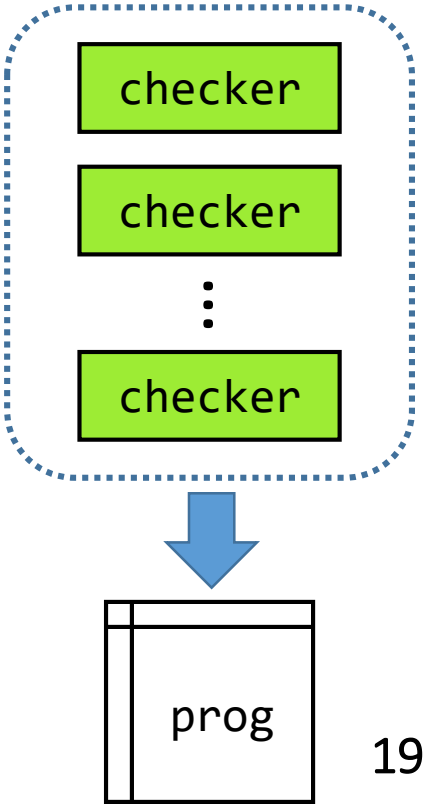


Analysis

1. Extract instructions that use configuration values
2. Produce necessary context
3. Prevent side effects
4. Capture runtime anomalies

5. Insert + invoke checking code in program bitcode/bytecode

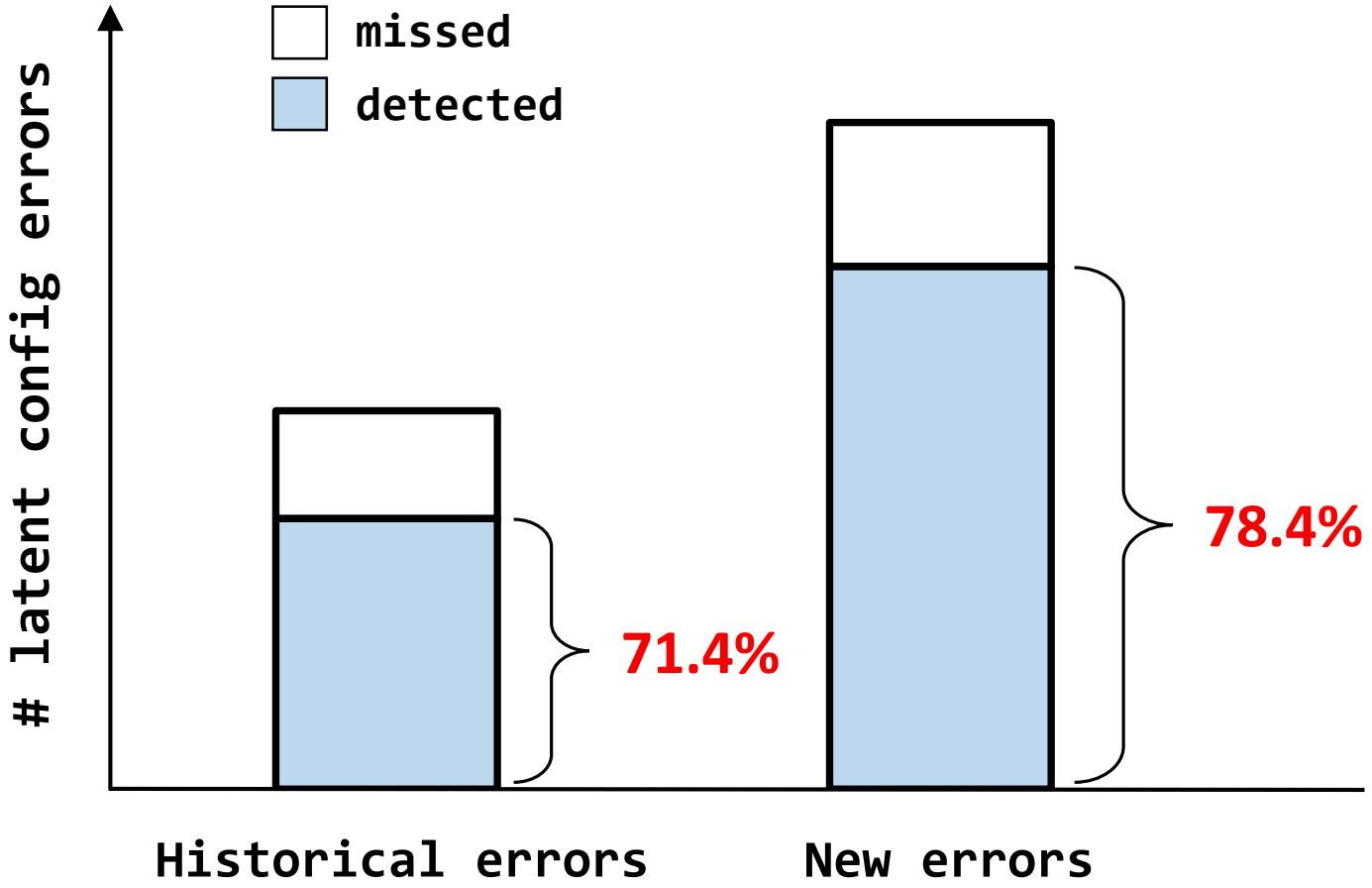
Output



Evaluation

- **Evaluate on 58 real-world latent configuration errors**
  - 37 **new errors** (discovered in our study)
  - 21 **historical errors** (caused failures in the past)

How many errors can be detected?



How many errors can be detected?

Type of errors	# (%) errors detected	
	Historical	New
Type/format errors	1/1 (100.0%)	13/13 (100.0%)
Invalid options/ranges	2/2 (100.0%)	4/4 (100.0%)
Incorrect files/dirs	9/12 (75.0%)	5/7 (71.4%)
Miscellaneous errors	3/6 (50.0%)	7/13 (53.8%)
Total	**15/21 (71.4%)**	**29/37 (78.4%)**

What errors are missed?

- **Cannot generate the checking code**
  - fail to produce the execution context
  - e.g., values from runtime requests
- **Cannot safely execute the checking code**
  - unknown side effects
  - e.g., used as bash command

Caveats

- **Errors manifested via a long running period**
  (we cannot run checks for too long.)
  - resource misconfigurations (exhaustion)
  - performance misconfigurations (degradation)

- **Errors not exposed via "obvious" anomalies**
  (we report errors based on exceptions, error code, etc.)
  - semantic errors (e.g., backup data to wrong files)

What is the cost?

- **Checking overhead**

HDFS/YARN/HBase	**less than 1000 msec**
Apache/MySQL/Squid	**less than 100 msec**

- **False positives**
  - tested with the default values and real-world settings collected from **830** configuration files
  - only **3** parameters have false alarms reported
    - caused by imprecise analysis that missed control-flow dependencies (the exposed anomalies are unreal)

Conclusion

- Treat configuration errors like fatal diseases
- PCheck: auto-generating & invoking configuration checking code to enforce early detection.

Check your configurations early!