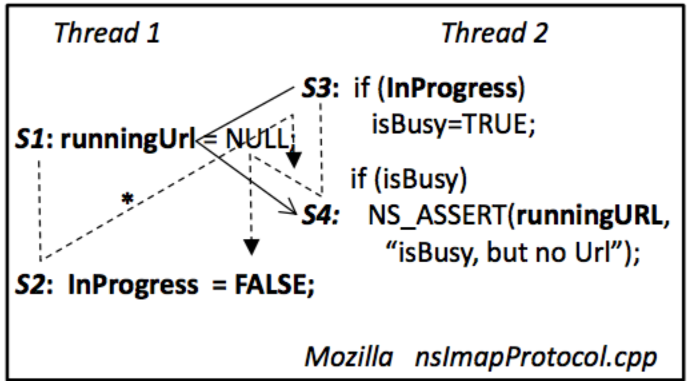
Conseq

- 总的来说，Conseq是一个面向结果的反向分析框架，用来检测concurrency bugs。

Bug 3 phases

3->2->1来检测bug

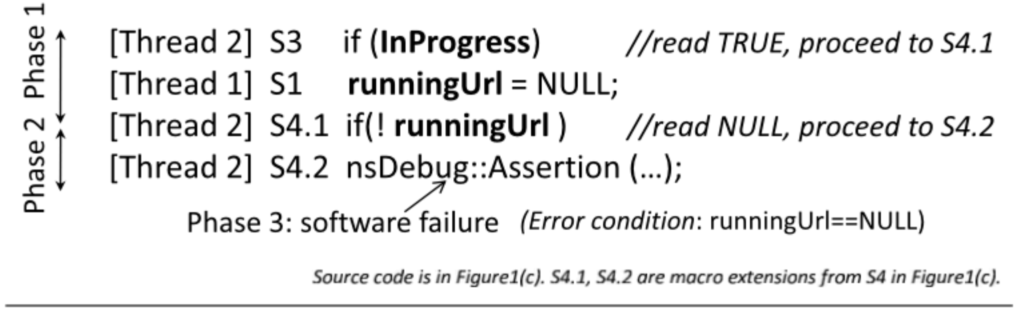
1. Triggering
2. Propagation
3. Failure



```
Thread 1                                Thread 2

                                        S3: if (InProgress)
                                              isBusy=TRUE;
S1: runningUrl = NULL;
                                            if (isBusy)
                                        S4:   NS_ASSERT(runningURL,
                                                "isBusy, but no Url");
S2: InProgress = FALSE;

                                Mozilla   nsImapProtocol.cpp
```



```
Phase 1   [Thread 2]  S3     if (InProgress)        //read TRUE, proceed to S4.1
          [Thread 1]  S1     runningUrl = NULL;
Phase 2   [Thread 2]  S4.1   if(! runningUrl )      //read NULL, proceed to S4.2
          [Thread 2]  S4.2   nsDebug::Assertion (...);
              Phase 3: software failure   (Error condition: runningUrl==NULL)
```

Source code is in Figure1(c). S4.1, S4.2 are macro extensions from S4 in Figure1(c).

Figure 3. Error propagation in a concurrency bug.

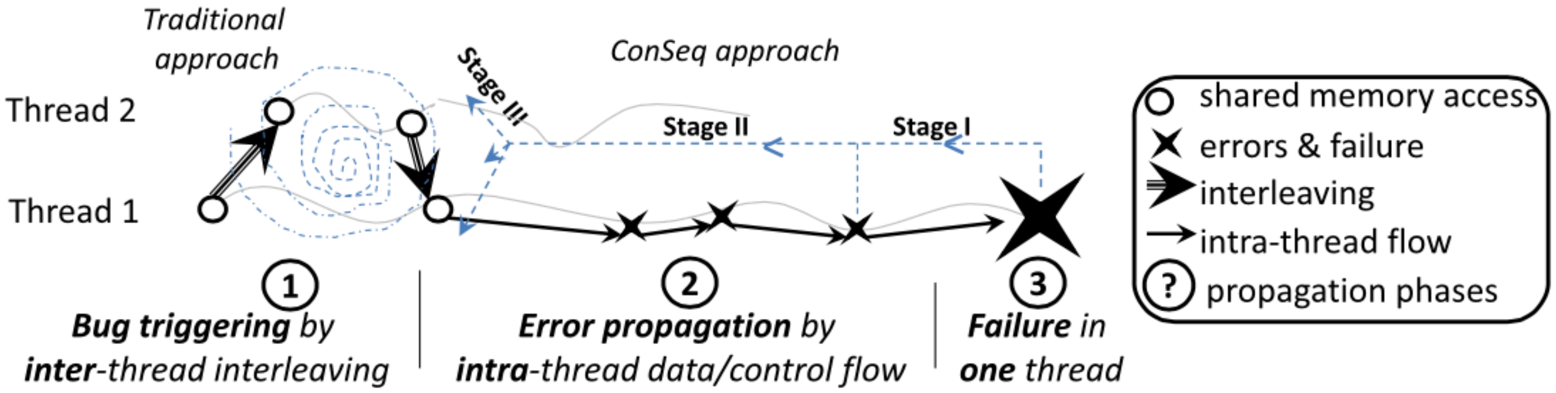
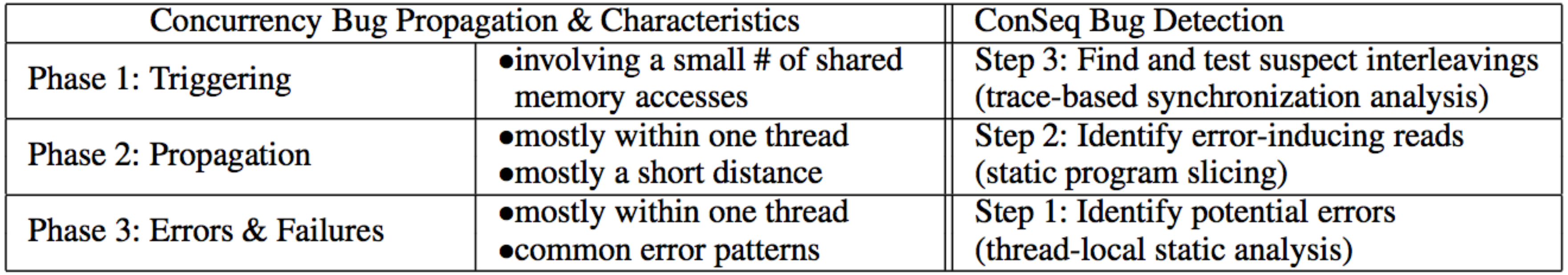


Figure 2. The common three-phase error-propagation process for most concurrency bugs.

Observations

Observation	Implication
bug的传播路径短	容易找到cause
failure通常发生在一个线程里	分为concurrency/sequential analysis
failure pattern和sequential bug相似	容易找到potential failure sites
产生bug的原因是共享内存访问	找读共享内存的指令



Concurrency Bug Propagation & Characteristics		ConSeq Bug Detection
Phase 1: Triggering	●involving a small # of shared memory accesses	Step 3: Find and test suspect interleavings (trace-based synchronization analysis)
Phase 2: Propagation	●mostly within one thread ●mostly a short distance	Step 2: Identify error-inducing reads (static program slicing)
Phase 3: Errors & Failures	●mostly within one thread ●common error patterns	Step 1: Identify potential errors (thread-local static analysis)

Table 1. Observations about concurrency bugs and corresponding components of ConSeq.

Architecture

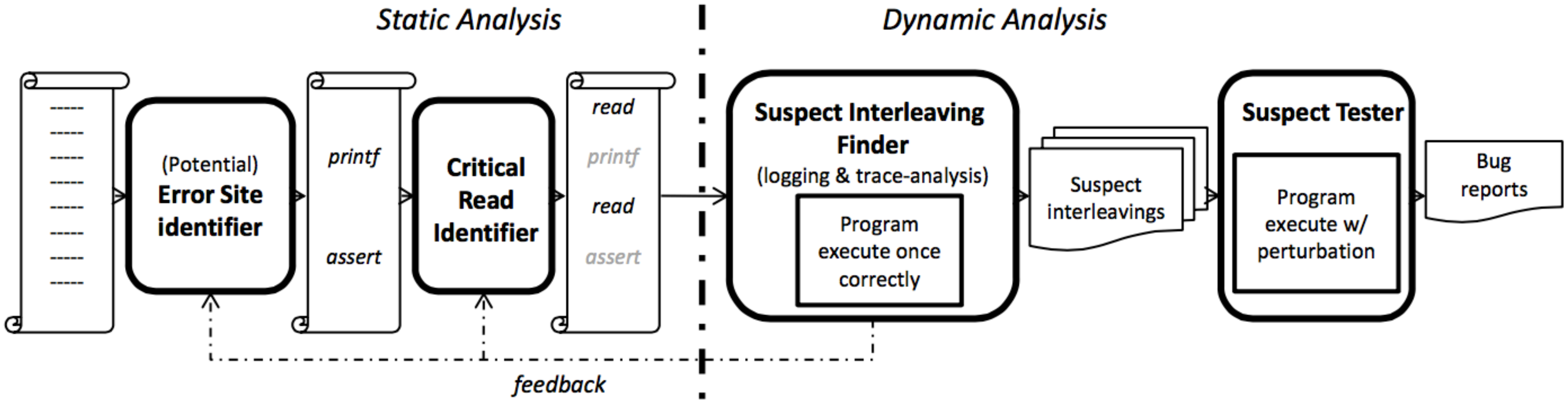


Figure 4. An overview of the ConSeq architecture.

Architecture example(1/4)

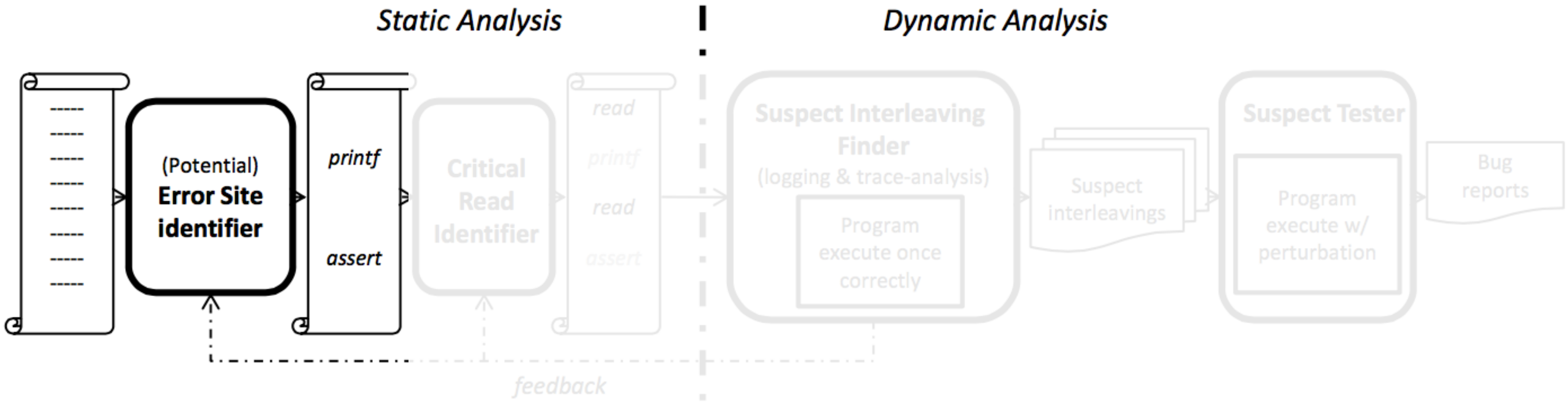
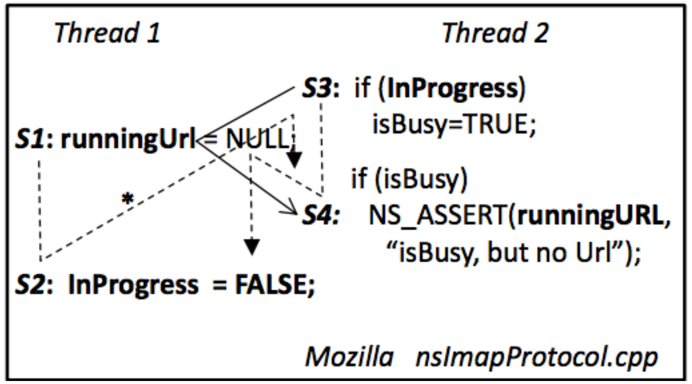


Figure 4. An overview of the ConSeq architecture.



```
S1: runningUrl = NULL;
S2: InProgress = FALSE;

S3: if (InProgress)
      isBusy=TRUE;
    if (isBusy)
S4:   NS_ASSERT(runningURL,
        "isBusy, but no Url");
```

Error site identifier: 从binary 中找到可能发生error的指令。

在这个例子中，找到200个assertion, 其中一个是S4中的assertion

Architecture example(2/4)

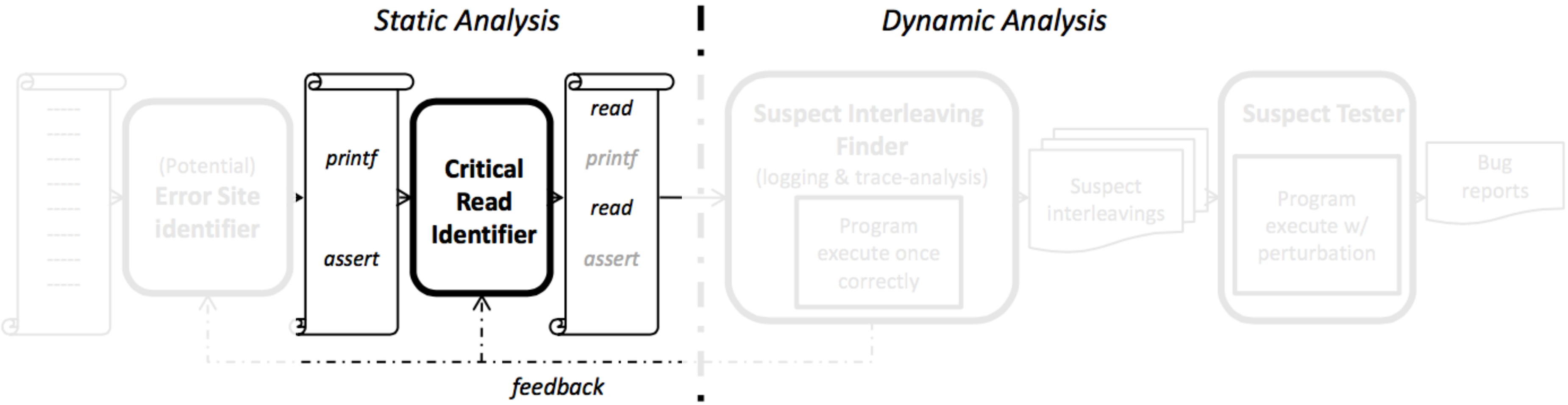
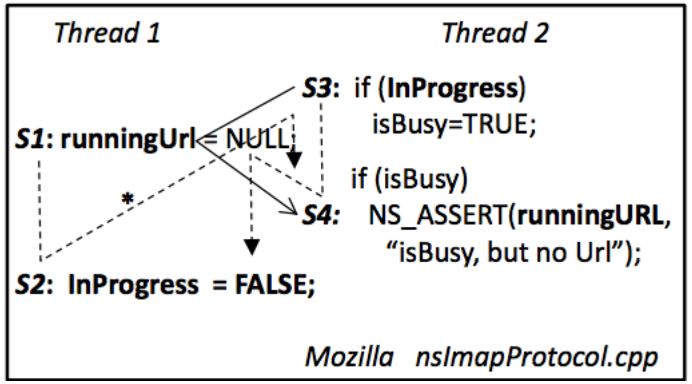


Figure 4. An overview of the ConSeq architecture.



```
Thread 1                          Thread 2
                                  S3: if (InProgress)
S1: runningUrl = NULL;                  isBusy=TRUE;
                                      if (isBusy)
                                  S4:   NS_ASSERT(runningURL,
                                          "isBusy, but no Url");
S2: InProgress = FALSE;

                              Mozilla   nsImapProtocol.cpp
```

Critical read identifier: 找到影响这些error site的read

这里找到读runningURL指令

Architecture example(3/4)

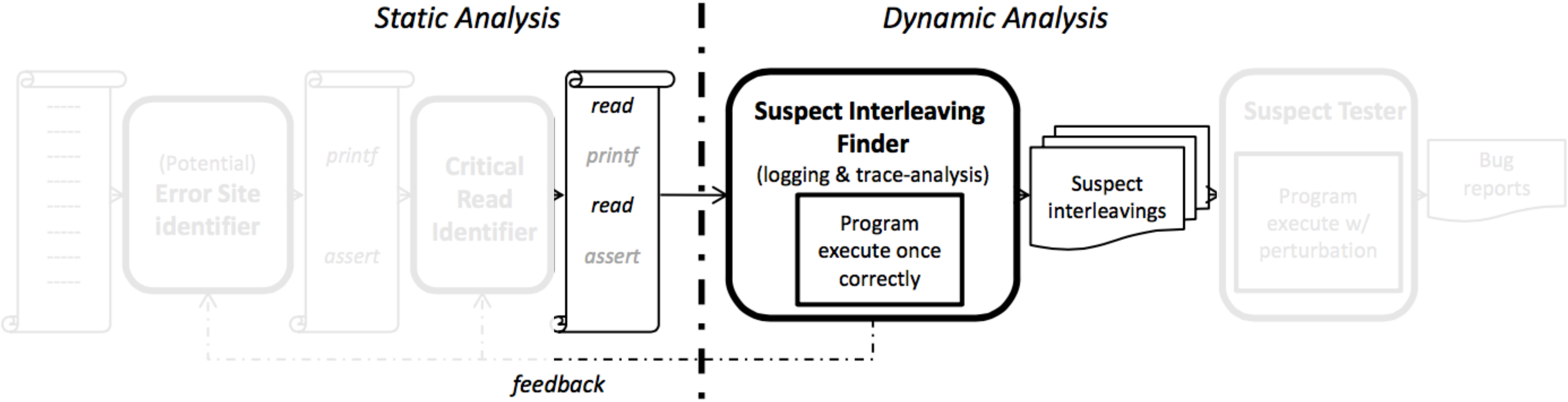
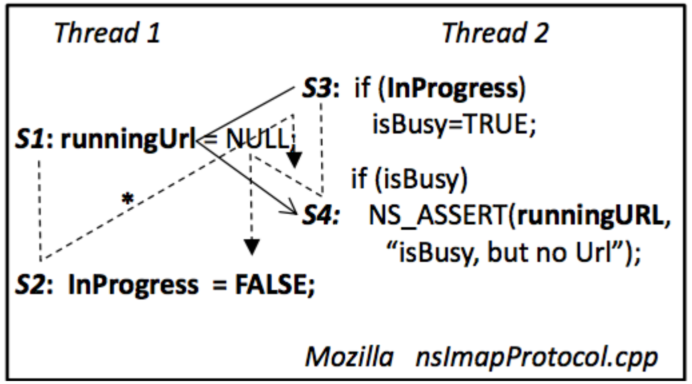

Static Analysis
Dynamic Analysis
(Potential) Error Site identifier
printf
assert
Critical Read Identifier
read
printf
read
assert
Suspect Interleaving Finder
(logging & trace-analysis)
Program execute once correctly
Suspect interleavings
Suspect Tester
Program execute w/ perturbation
Bug reports
feedback

Figure 4. An overview of the ConSeq architecture.



```
S3:  if (InProgress)
       isBusy=TRUE;
S1: runningUrl = NULL;
     if (isBusy)
S4:    NS_ASSERT(runningURL,
         “isBusy, but no Url”);
S2: InProgress = FALSE;
```

Mozilla nsImapProtocol.cpp

Suspect interleaving finder: 监测程序的一次执行，找到可能导致错误的Interleaving

这里找到interleaving: S4可能读S1中设置的NULL

Architecture example(4/4)

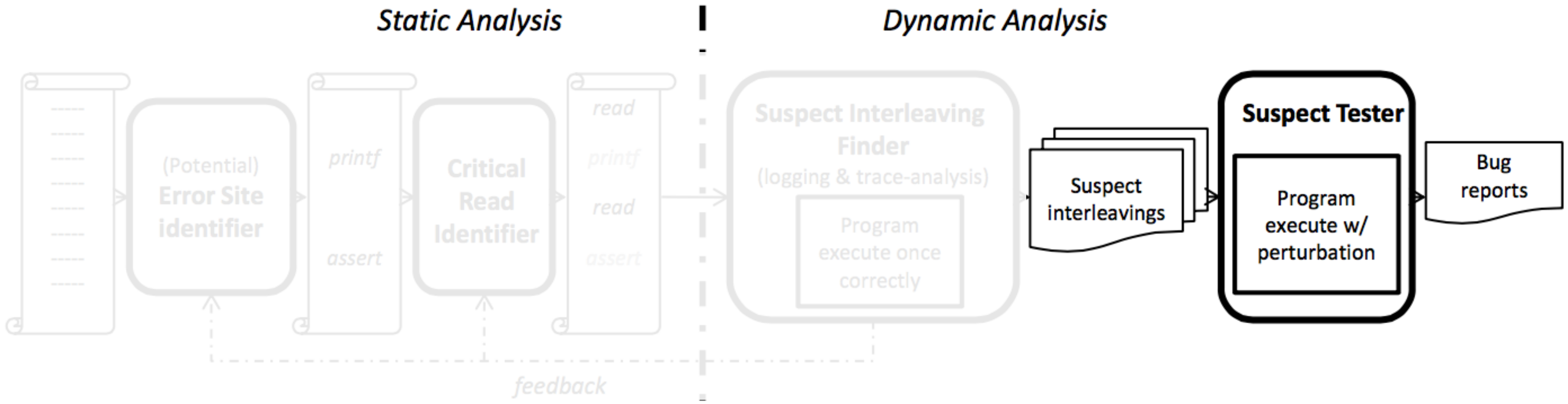
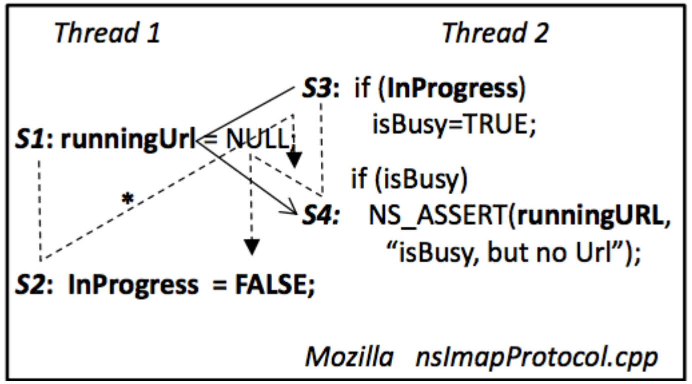


Figure 4. An overview of the ConSeq architecture.



```
S1: runningUrl = NULL;      S3: if (InProgress)
                                  isBusy=TRUE;
                                if (isBusy)
                            S4:   NS_ASSERT(runningURL,
                                   "isBusy, but no Url");
S2: InProgress = FALSE;
```

Suspect tester: 执行测试，对程序进行微小调整，看是否产生bug

这里，发现S3->S1->S4执行时，产生bug。最终将其放入bug reports里

5种bug pattern

- Infinite loop: back-edge in a loop
- Assertion violation： assertions
- Memory Errors: invalid memory access例如Null-pointer dereference
- Incorrect outputs: printf and fprintf
- Error messages: 有一些输出error message的指令， 例如fprintf, NS_WARNING等

Evaluation

- 在7个C/C++开源项目中找11个concurrency bugs，能够成功检测到10个。
- 新的bug
  - Aget:2
  - Click:2
  - Cherokee:1个non-deterministic output问题
  - MySQL: infinite loop