Software Based Fault Injection Framework For Storage Systems

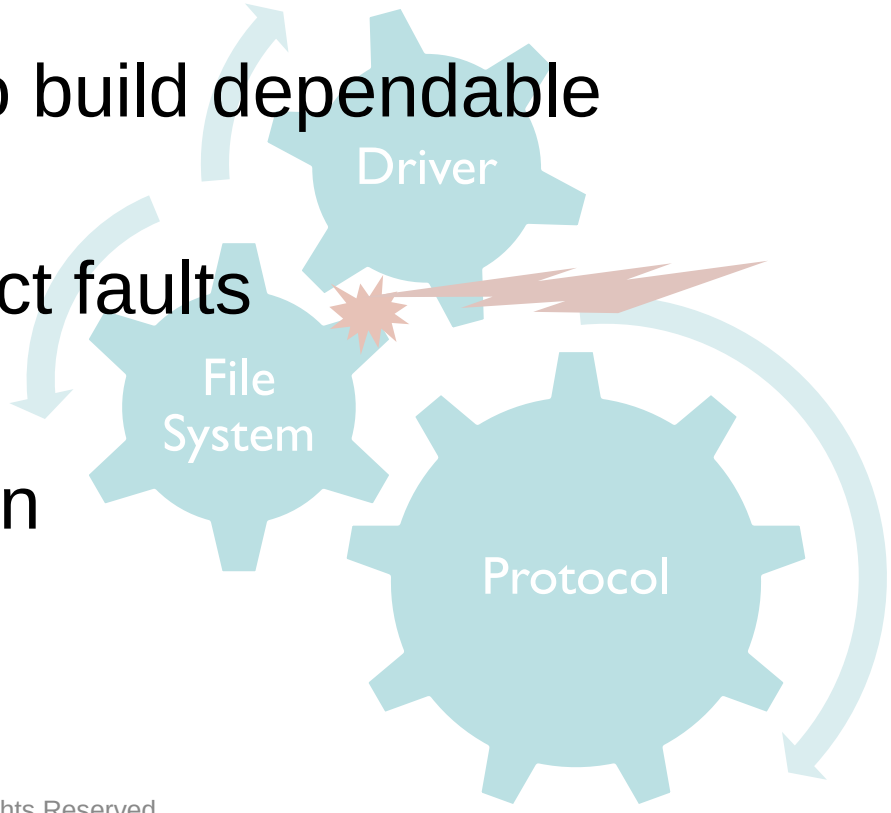
Vinod Eswaraprasad

Smitha Jayaram

Wipro Technologies

The agenda

- Reliability in Storage systems
- Types of errors/faults in distributed storage server
- Layered Fault Injection to build dependable systems
- Unified framework to inject faults at various layers
  - Sample Implementation
- Learning/Conclusion

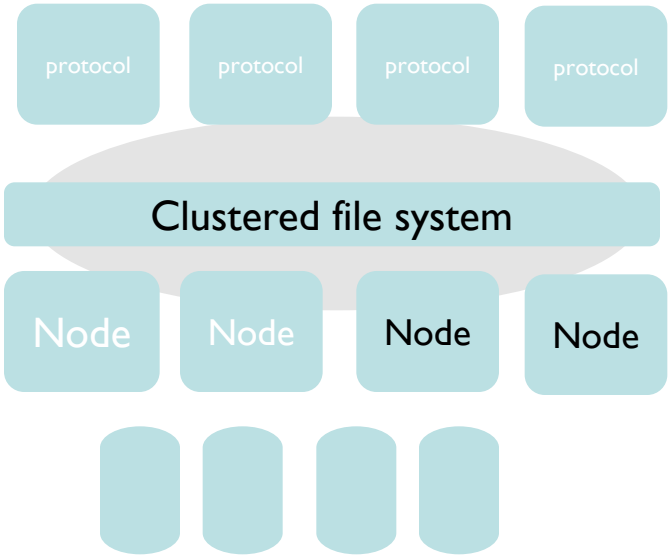


Background and Objective

- Software systems are often delivered with “residual” faults
- Increased complexity in scalable systems; Clustered environment makes it harder.
- Ensuring system robustness - NAS
  - Behavior of layers - protocols, file systems and drivers during certain errors.
  - Ability of filesystem repair tools to handle unexpected failure scenarios.
- Improve the coverage of file system and file system utilities
- Analyze the dependability (tolerance and recovery aspect) of a storage system

Target - A scalable storage system

- Details about the environment
  - Nodes - Multicore Intel processors
  - Disk – FC array
  - File system - A clustered file system with global namespace
- Large Scale storage system
  - With multiple nodes
  - Increased complexity (hardware/software)
  - Challenged by scale
  - Challenged by failures



Terminology - Definitions

Laprie-Randell Taxonomy

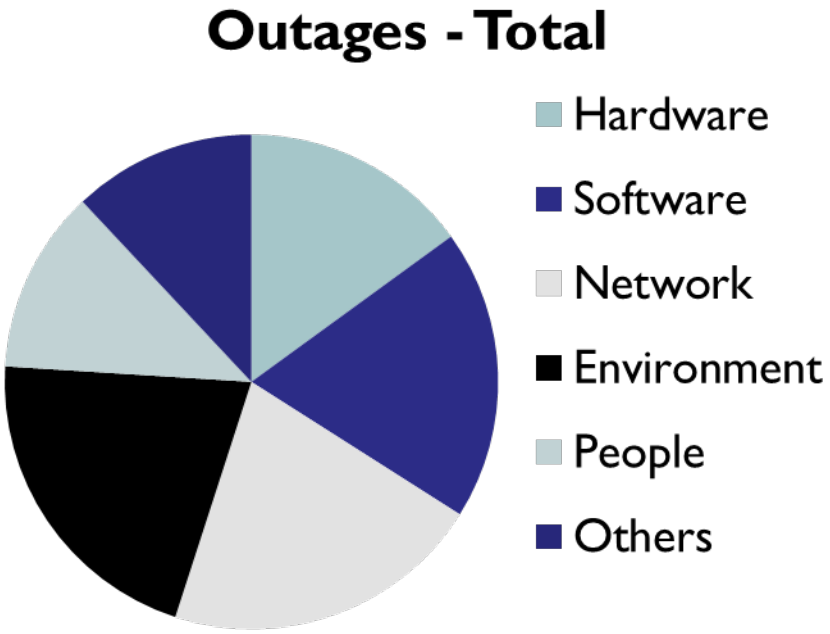
- Dependable and Secure computing 04

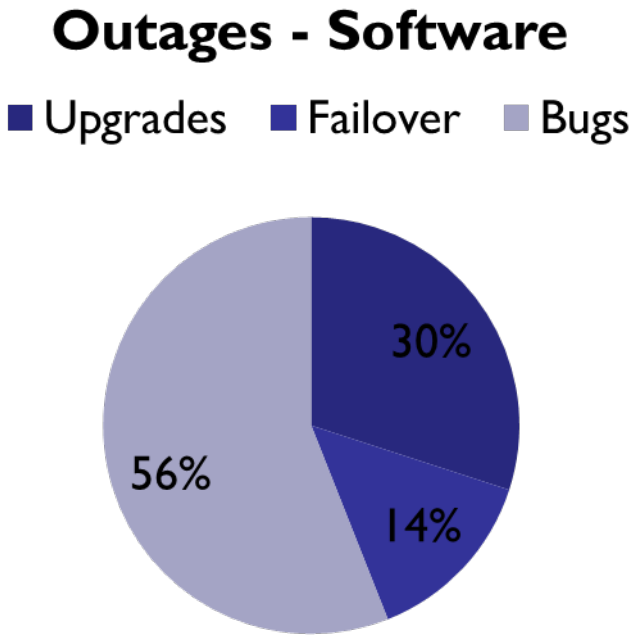
- Fault - a defect in a service, can be “active” or “dormant”
- Error – an “active fault” in a service
- Failure –unsuppressed error, deviates the delivered service from correct service
- Fault Injection

  - Purposeful introduction of faults (errors) into target hardware or software

- Software Fault Injection

A look into the outage causing failures





www.availabilitydigest.com/ - **The causes of Outages**

Hardware failures and software errors contribute equally to data loss

Classification of storage errors

- Nature of the error
  - Deterministic vs. Random
  - Recoverable vs. Non-recoverable
- Frequency of occurrence
  - Transient
  - One-time occurring
  - Repeatedly occurring, measured per second
- Impact of occurrence
  - Data corruption, loss, unavailability
  - Single fault causing multiple failures
    - cascading effect

Classification of storage errors (Cntd.)

- Data Availability
  - Hardware down – disk controller, disk drive, storage array fails
  - OS crash, server crash, system hang
  - Heart beat monitoring failure
  - Takeover/Giveback failure
  - Errors that affect latency and throughput
- Data Accessibility
  - Denial of access to files with permissions
  - Illegal access to files without permissions
- Limits or boundary errors
  - Writes/reads beyond disk boundaries
  - System overload

Typical Errors - Causes

- File system response to disk failures
  - Latent sector faults
  - Silent corruption in blocks
  - Misdirected writes
    - How does file system handle this?
    - How the check-and-recover tools behave?
- Software errors
  - Logical errors
  - Race conditions
  - Cluster fail-over issues
- Clustered file system behavior during interconnect/network failure
  - Interconnect failure affect data path latency
    - delays, out of order messages, lost messages
  - Effect of interconnect failure in local caching
  - Distributed Lock management during failure

Typical Errors - Effects

- Data Corruption
  - Metadata – on-disk and in-memory, stale metadata cache.

    Loss of complete file or directory
  - User Data – incorrect block translation, buffer cache corruption, bad checksum

    Partial or complete corruption of user data
- Data Loss
  - Partial or complete loss of metadata/data blocks
  - Lost chunks in a distributed file
- Data Unavailability
  - Synchronization and lock ordering violations
  - System or sub-system hangs
  - System crash

Fault Injection framework

- Ability to inject fault at software level and monitor the effect in protocol layer, file system, and driver.
- The essential properties
  - Simple
  - Non-intrusive
  - Versatile
  - Reproducible
  - Distributed
  - Real-time

The layers of interest

- Protocol
  - File serving protocol implementations
  - User/Kernel space
- File system
  - Virtual and physical file systems
  - File system utilities
- OS kernel
  - Page caches
  - VM and management
- Storage device drivers
- Firmware

Options for fault injection

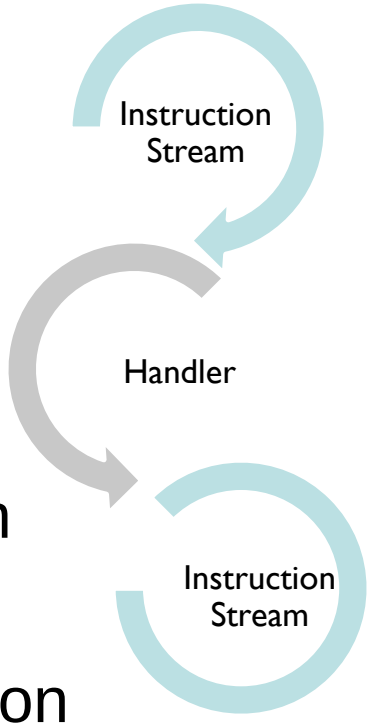
- Software vs. Hardware
- Software
  - Flexibility
  - Expansion
  - Ability to control timing - delay, races
  - Distributed environment
  - Less risk – no damage
  - Standard environment
- Mechanism based on software probes
  - Works across layers

Kernel level

- ActonAT – Perform certain action when execution flow hits an instruction.
- Implementation specific to the processor hardware and the underlying kernel
- Mostly done with the use of
  - Special purpose instructions – break
  - Hardware assisted breakpoint registers
- Software probes
  - Several implementations
    - Kprobe
    - Dtrace
    - Etc.

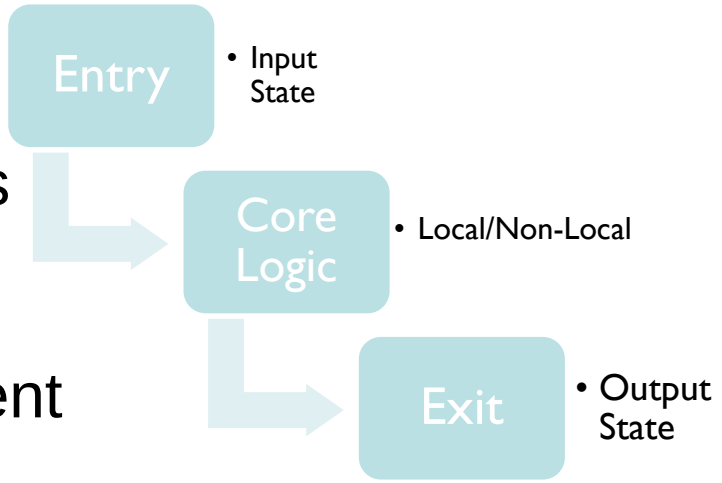
Kprobe – As a FI mechanism

- Traditionally used to gather performance and debugging information
- With a little instrumentation, this can be used to inject errors
  - Live system, Non intrusive
- Works on standard product build – no special purpose build steps
- Specify the instruction point (IP) along with a pre and post handles
- Handler functions gets called when execution hits the IP
- The fault injection logic in the handlers.



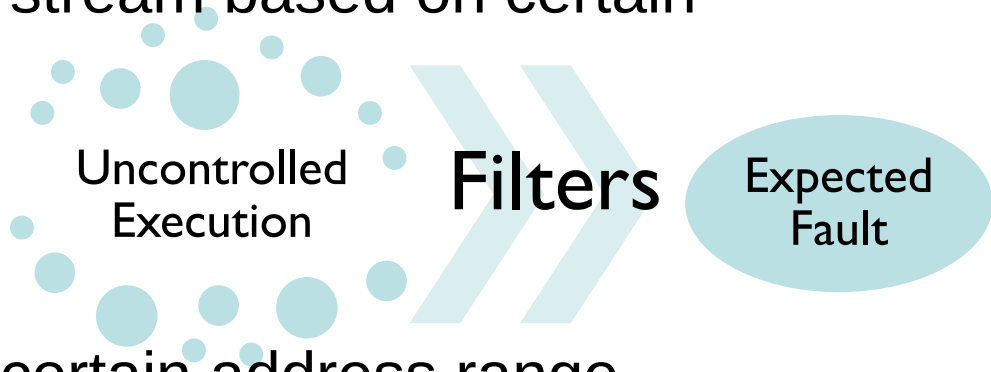
Fault Injection points

- Control the code execution in a given instruction stream; real-time.
  - Entry State - Arguments
  - Exit State – Return values
  - Partial execution with early returns
- Non local state changes
  - State change not caused by current stream
- Timing control
  - Synchronized events
  - Multi tasking and Multi threaded environments



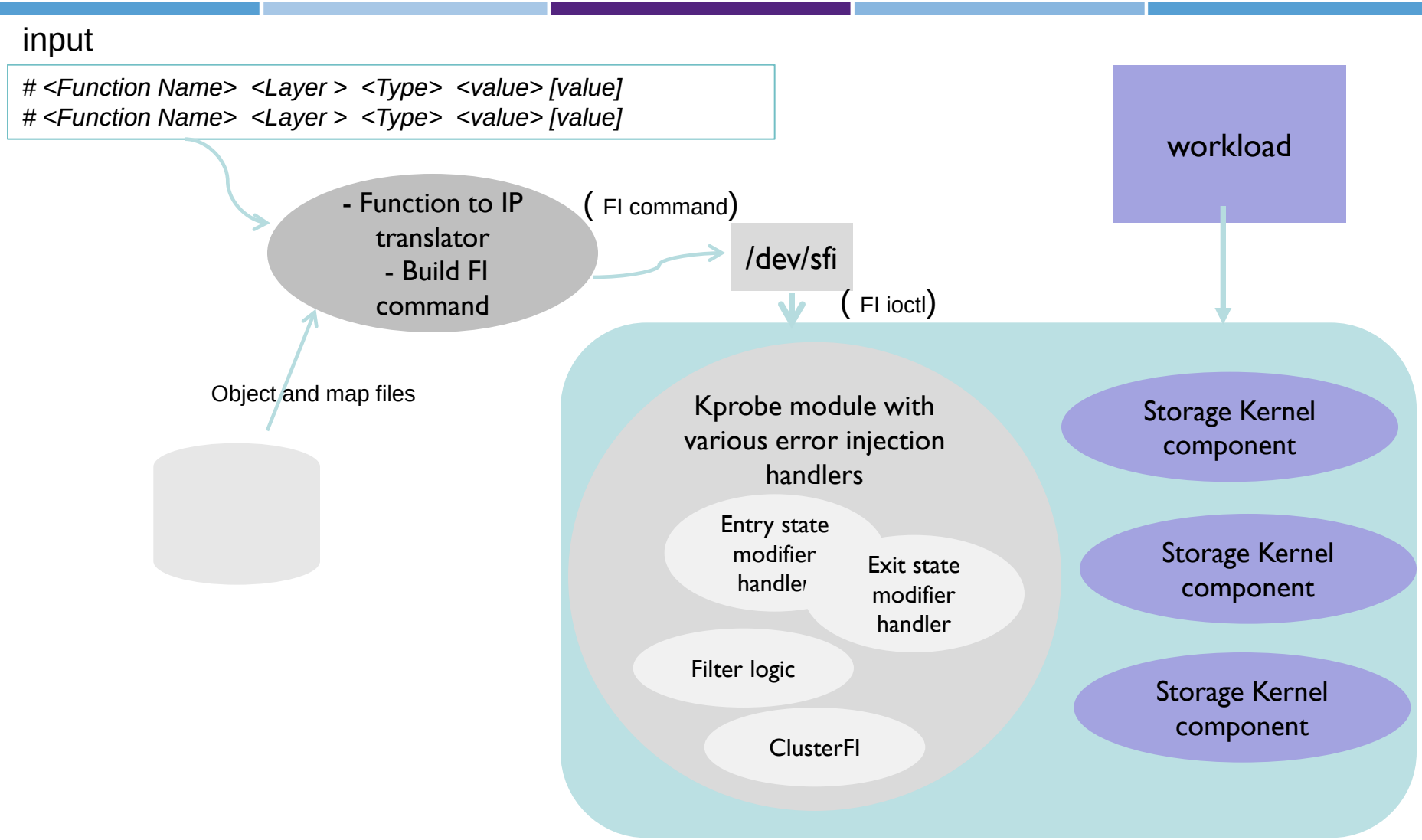
Filters – for selective faults

- Filter based on PID
  - Select target task/thread of action
- Event based filter
  - Inject error on execution stream based on certain events



- Address range filter
  - Error injection based on certain address range
- Call Chain filter
  - Error injection only during specific call sequence

The automated framework



Error injection samples

- Sample Code
- Demonstrates:
  - Filtering
  - Fault injection
  - Manipulation of execution sequence

Sample FI Probe

Sample Code flow

```
getObject (type1 arg1, type2 object)
{
      read_from_disk() and set as object
      return object and status of read

}

getAndUseObject(type1 arg1, type2 arg2)
{
    Allocate and Initialize Object
    retval = getObject(arg1, arg2, &object);
    if (retval == ok)
    {
        if (condition1)
           Do something1
       else
           Do something2

       UseObject(&object, arg2, arg3);
    }

     Cleanup and return
}
```

Sample Probe and snippet of an FI command

```
// Module registration
kprobe_init
{
      kp.pre_handler = handler_pre;
      kp.post_handler = handler_post;
      kp.fault_handler = handler_fault;
      kp.sym = fi_context.func_name      // ex., getObject ()

      register_kprobe(&kp);

}

// Module de-registration
kprobe_exit
{
    unregister_kprobe(&kp);
}

// Components of an FI command:

probe_func_name                 filter_addr_val
probe_func_line_num             filter_caller_func_name
exit_val                        filter_caller_func_line_num
input_val                       partial_exec_func_name
filter_event_val                partial_exec_func_line_num
```

Sample FI Probe handler - filtering and modify input/exit condition


```
handler_<xxx>(struct kprobe *p, struct pt_regs *regs)
{

    1. Apply Filter Policy

    2. Inject fault: Modify Input condition

    3. Inject fault: Modify Exit condition

}
```

1. Event based filter
   if (fi.filter_event_val == fs.op_context)

2. PID based filter
   if(fi.filter_pid_val == current_task.pid)

3. Address based filter
   if(fi.filter_addr_val == fs_context.obj_addr)
   ex, inode or dentry based filtering

4. Caller based filter
   if(fi.filter_caller_func_addr == regs->sp)

Modify registers that hold input parameters:
   regs->rdx = fi.input_val

1. Modify registers that hold return values:
   regs->rax = fi.exit_val

2. Modify the caller's stack directly
   *(sp+offset) = fi.exit_val

Manipulating of execution sequence

Sample Code flow

```
getObject (type1 arg1, type2 object)
{
        read_from_disk() and set as object
        return object and status of read

}

getAndUseObject(type1 arg1, type2 arg2)
{
    Allocate and Initialize Object
    retval = getObject(arg1, arg2, &object);
    if (retval == ok)
    {
        UseObject(&object, arg2, arg3);

        // Do something else

    }
    Cleanup and return
}
```

A

B

```
handler_post(struct kprobe *p, struct pt_regs *regs)
{
    Skip code parts within a function to avoid
    state changes caused by the function where
    fault is being injected
     (modifying IP does not work!!)
}
```

Probe inserted at getObject():

Jump from A to B:

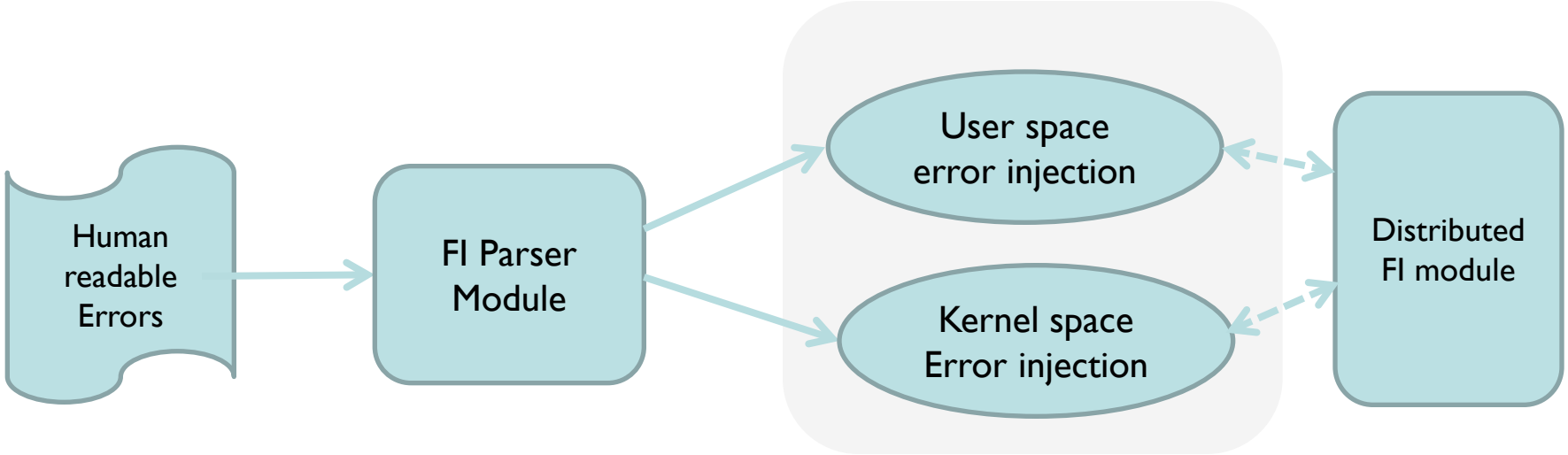
1. *Modify exit condition at the post handler to a value which is !(ok).*
2. *Modify sp to point to the instruction addr pointing to B*

Errors at Protocol Layer

- User space or kernel space implementations
- Typical errors:
  - Malformed packets
  - File system errors
  - Response timeouts
  - Lost requests
  - Race conditions
  - Reaching/exceeding limits
- Cluster failover
  - Failover timing, detection delays
  - Split brain issues - protocol

The Unified Framework

- A single interface for all layers
- User specifies the error in a human readable form
  - Layer, Function, Type of Fault, Configurable values
  - The FI structure formed will be directed to the appropriate module
    - Kernel Space - Kprobe
    - User Space – Library intercept, GDB interface
    - Firmware Interface - Inserted Module



Learning and Conclusions (1)

Behavior of Clustered Filesystem under Errors

- Deficiency in handling random, un-common errors
- Simulated range of 35 error - Outage causing scenarios
- Transient outage causing errors

Effectiveness of data recovery tools

- Analysis of check and recovery tools - error scenarios
- Detected around 6 unhandled common FS errors in FS check tools
- Incorrect recovery procedure
- Distributed Metadata handling issues during network failure

Learning and Conclusions (2)

The unified framework

- Layered and Flexible framework
- Human readable, scriptable error scenarios
- Analyze robustness by injecting simultaneous faults at different layers
- Currently on x86 and Linux based systems – Portable

Questions?

Thank You