Go at the DARPA Cyber Grand Challenge

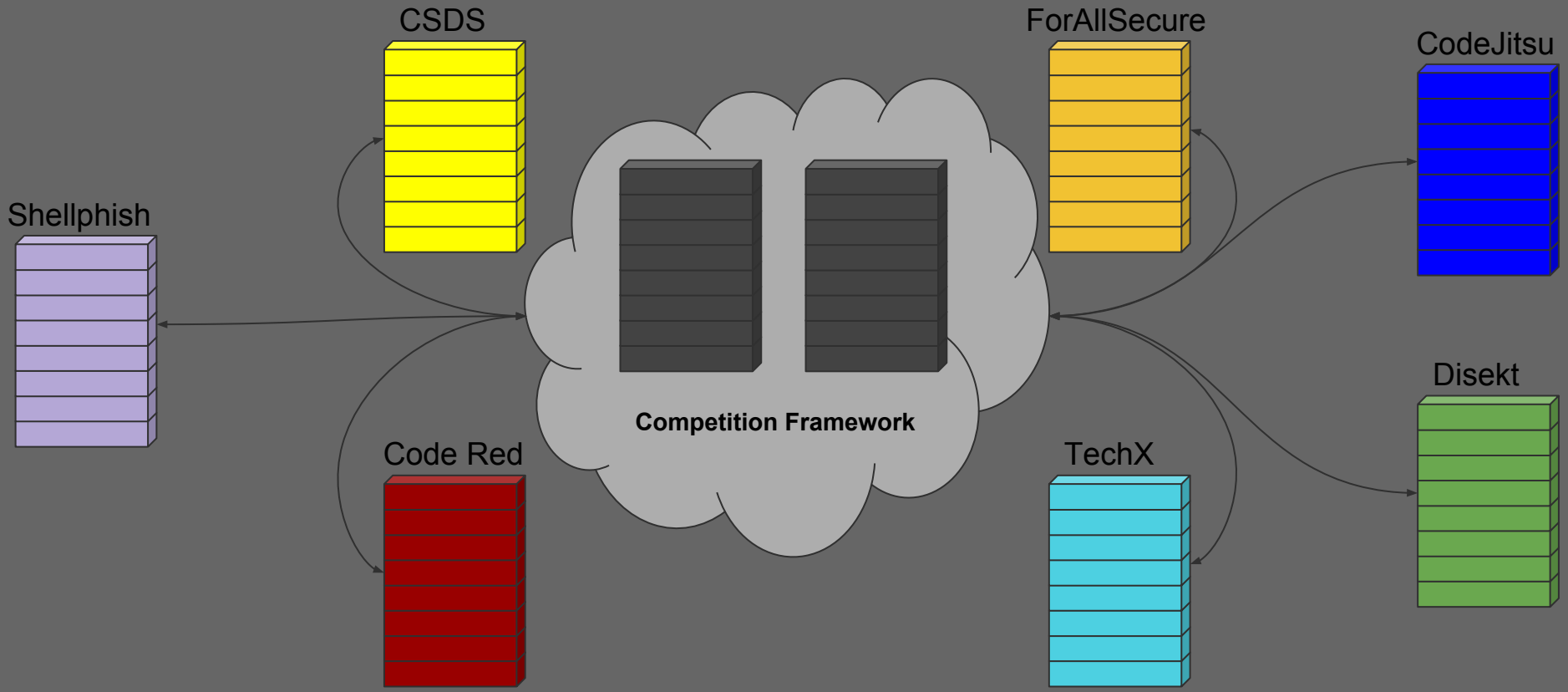
Channels and Parallelism for High Performance Database, Network and File I/O

Will Hawkins

Introduction

DARPA Cyber Grand Challenge

DARPA Cyber Grand Challenge



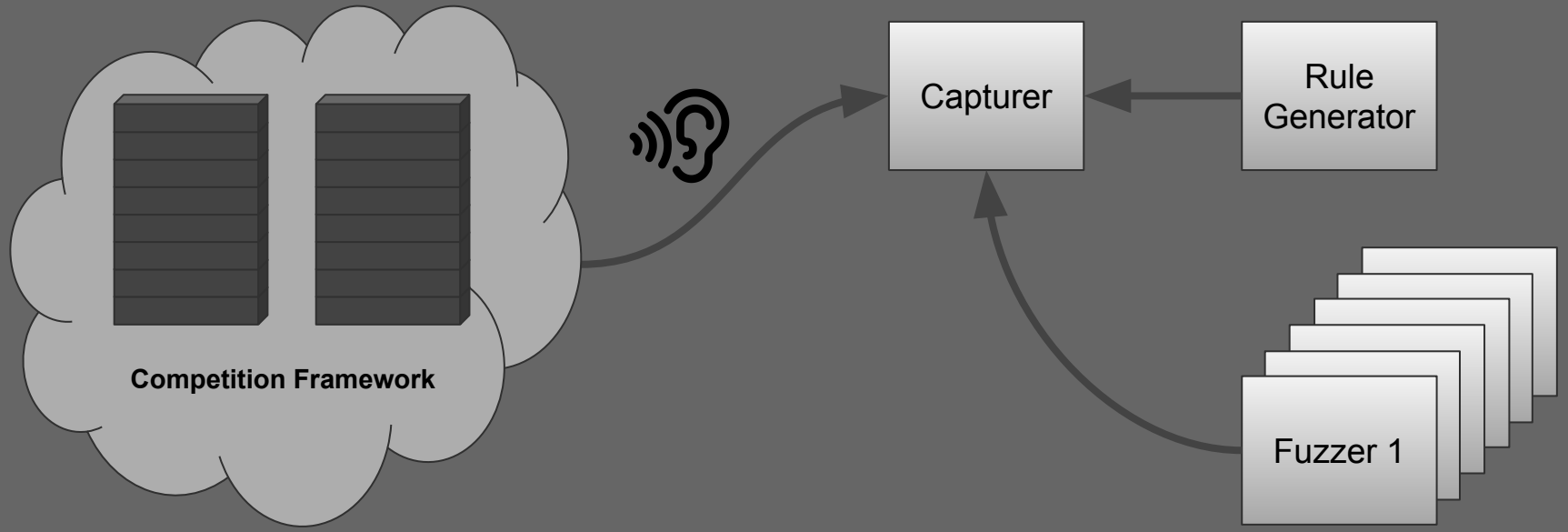
DARPA Cyber Grand Challenge

Our Goal: Finish 1000

DARPA Cyber Grand Challenge

Our Goal: Finish 1000 (= ^0111 = “Not 7th”)

DARPA Cyber Grand Challenge



Ear by Gregor Cresnar, Noun Project

*Go*al

- Get it? The go-al?
- Capturer will capture …
  - … the fields from the packets …
    - Timestamp
    - Cbid
    - Conversation
    - Side - either client or server
    - Message ID
    - Contents
  - … into a data store that can be queried by different components.
    - The Fuzzer
    - The Rule Generator

Results

- This ended up being a significant component of our defensive systems
- The Fuzzer
  - Used data captured from the network tap to generate inputs for our Fuzzers
- The Rule Generator
  - Rules were deployed in concert with hardened binaries.
  - Hard to tell which was the effective defense.
  - However, there were two specific cases where replacement binaries were vulnerable *but* the IDS rules generated by the Rule Generator protected the binary from successful attack.

Design, Architecture and Implementation

- Capturing
  - Simple tcpdump
  - libpcap
- Filtering
  - BPF
  - Custom
- Storing
  - Flat file
  - Database
    - SQL
    - NoSQL

- Off-the-shelf? Custom?
  - BASH pipeline: tcpdump ‘port 1993’ | sed | awk | … > file
  - Custom program
    - C
    - C++
    - Go

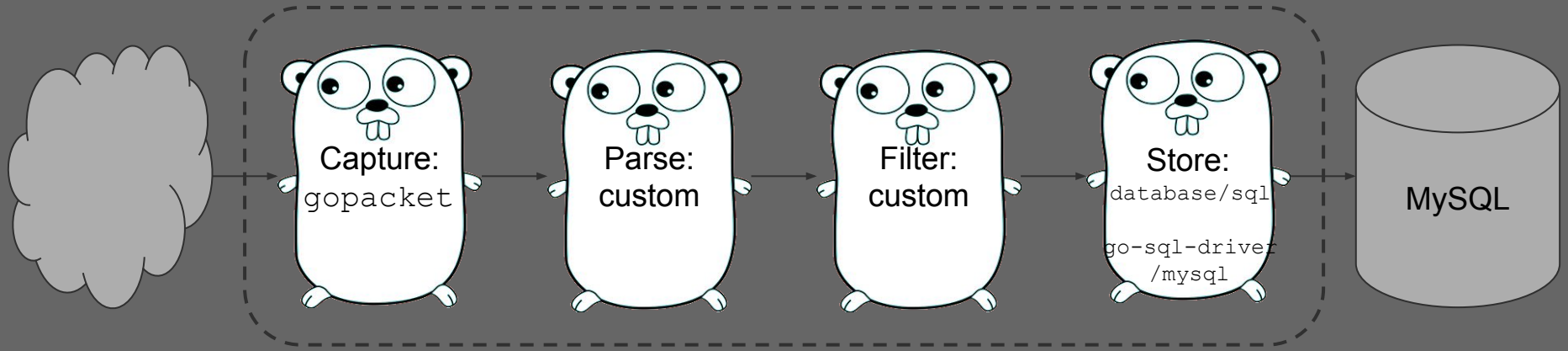
Design, Architecture and Implementation

- Capturing
  - Simple tcpdump
  - **libpcap**
- Filtering
  - BPF
  - **Custom**
- Storing
  - Flat file
  - **Database**
    - **SQL: MySQL**
    - NoSQL

- Off-the-shelf? Custom?
  - BASH pipeline: tcpdump ‘port 1993’ | sed | awk | … > file
  - Custom program
    - C
    - C++
    - **Go**

Design, Architecture and Implementation

- Capturing
- Filtering
- Storing
- Off-the-shelf? Custom?



Capturing

- gopacket
  - API for
    - accessing different packet sources in a consistent manner (`PacketDataSource` interface), and
    - accessing data in packets in a consistent manner
  - Sources include: byte array, files (from, e.g., `tcpdump` output), live devices (from, e.g., `libpcap`, `pfring`)
- Three steps for use:
  - Handle
    - `OpenLive()`
    - `OpenOffline()`
  - Source from Handle
    - `NewPacketSource()`
  - Capture
    - `Packets()`

Capturing

```
live_handle, err = pcap.OpenLive(*iface, 1024, false, pcap.BlockForever)
…
defer live_handle.Close()
…
live_source := gopacket.NewPacketSource(live_handle, live_handle.LinkType())
…
for stopping == false {
     select {
          …
          case p, ok := <- live_source.Packets():
               if !ok {
                    stopping = true
                    break
               }
     }
}
```

Filtering

- gopacket (again)
- Two Steps For Use:
  - From a packet, get the Layer: `Layer()`
  - Use the Layer

Filtering

```
var ipv4_layer gopacket.Layer
var udp_layer gopacket.Layer
var udp *layers.UDP
var ip *layers.IPv4

if ipv4_layer = i.Layer(layers.LayerTypeIPv4); ipv4_layer == nil {
     …
}
if udp_layer = i.Layer(layers.LayerTypeUDP); udp_layer == nil {
     …
}
udp, _ = udp_layer.(*layers.UDP)
ip, _ = ipv4_layer.(*layers.IPv4)
if filter.DstPorts != nil && !cgc_utils.MatchAnyPort(udp.DstPort,filter.DstPorts){
     …
}
```

Parsing

```
func ParseCgcPacket(packet []byte) (cgc_packet IdsPacket, err error) {
     packet_length := len(packet)
     packet_offset := 0

     if (packet_offset+4) > packet_length {
               err = errors.New("Could not parse past first field.")
               return
     }
     csid := binary.LittleEndian.Uint32(packet[packet_offset:])
     cgc_packet.Csid = fmt.Sprintf("%x", csid)
     packet_offset += 4

     if (packet_offset+4) > packet_length {
               err = errors.New("Could not parse past csid field.")
               return
     }
     cgc_packet.ConnectionID = binary.LittleEndian.Uint32(packet[packet_offset:])
     packet_offset += 4
     …
     return
}
```

Storing

- database/sql in combination with Go-MySQL-Driver
- Three steps for use:
  - Connect to the database: `Open()`
  - Prepare statements (optional): `Prepare()`
  - Execute statements: `Exec()`
  - Check/Retrieve Results:
    - Check return value, or
    - `Next()`

Storing

```
db, err := sql.Open("mysql",
                    user+":"+password+"@unix(/var/run/mysqld/mysqld.sock)/"+database)
if err != nil {
     return nil, err
}
if err = db.Ping(); err != nil {
     return nil, err
}
statement, statement_err = database.Prepare("insert into pcap
                                             (cbid, conversation, side, message, contents)
                                      VALUES (?,    ?,            ?,    ?,       ?)")

if _, err := statement.Exec(packet.Csid,
                            packet.ConnectionID,
                            fmt.Sprintf("%v", packet.ConnectionSide),
                            packet.MessageID,
                            packet.Message); err != nil {
     fmt.Printf("Exec() error: %v\n", err)
}
```

Testing

- DARPA played games with us
- `tcpdump`: Capture the traffic from the simulation
  - All
  - Some
- `tcpreplay`
  - Real time
  - Accelerated
- Storage required
  - Fixed
  - Variable

Testing

- DARPA played games with us
- `tcpdump`: Capture the traffic from the simulation
  - All
  - **Some:** Filter meta traffic (ssh, etc); Use only 200,000 packets.
- `tcpreplay`
  - Real time
  - **Accelerated**: `-t`; Playback all 200,000 packets in 15s; That's roughly 7.43Mbps.
- Storage required
  - Fixed
  - Variable
  - **N/A**

Initial Result

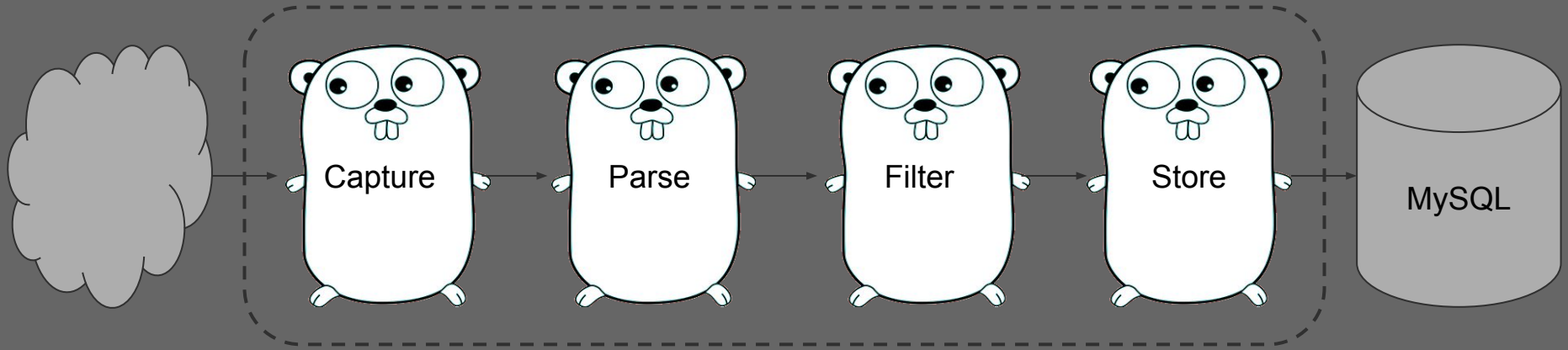
200,000pkts/200,000pkts=**100%**

Questions

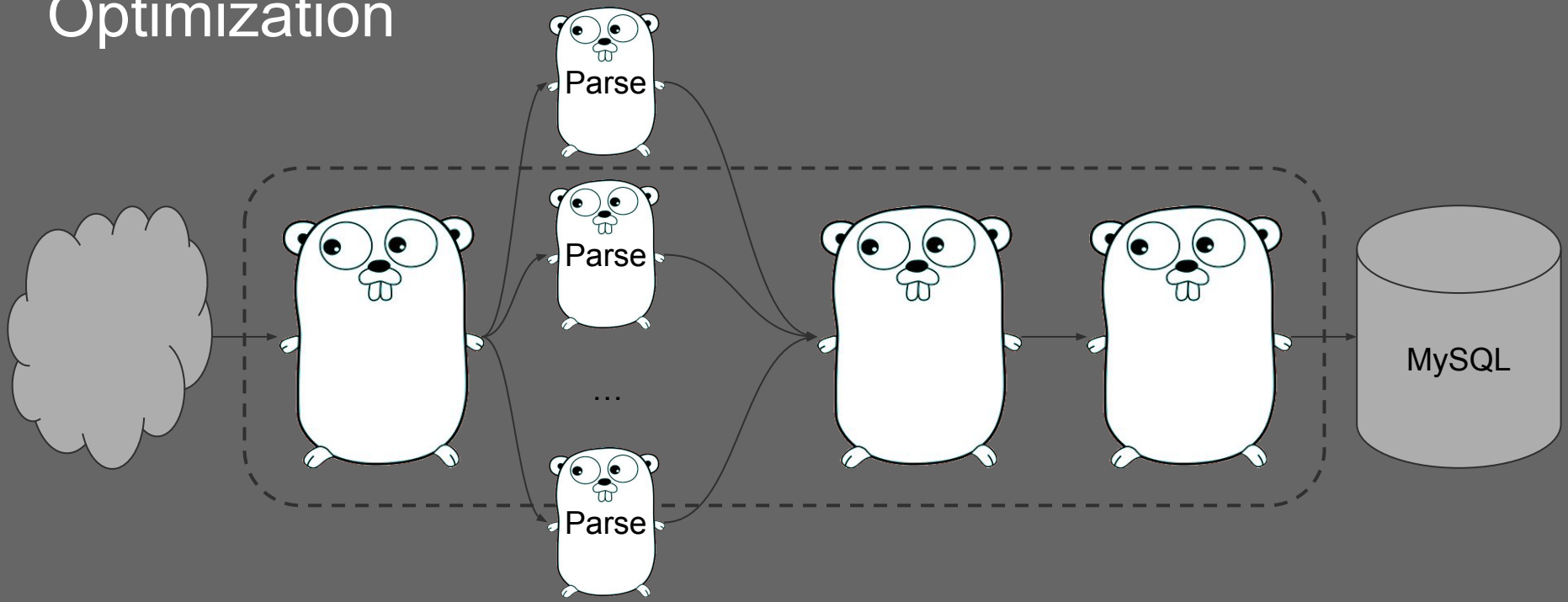
Initial Result

844pkts/200,000pkts=**0.42%**

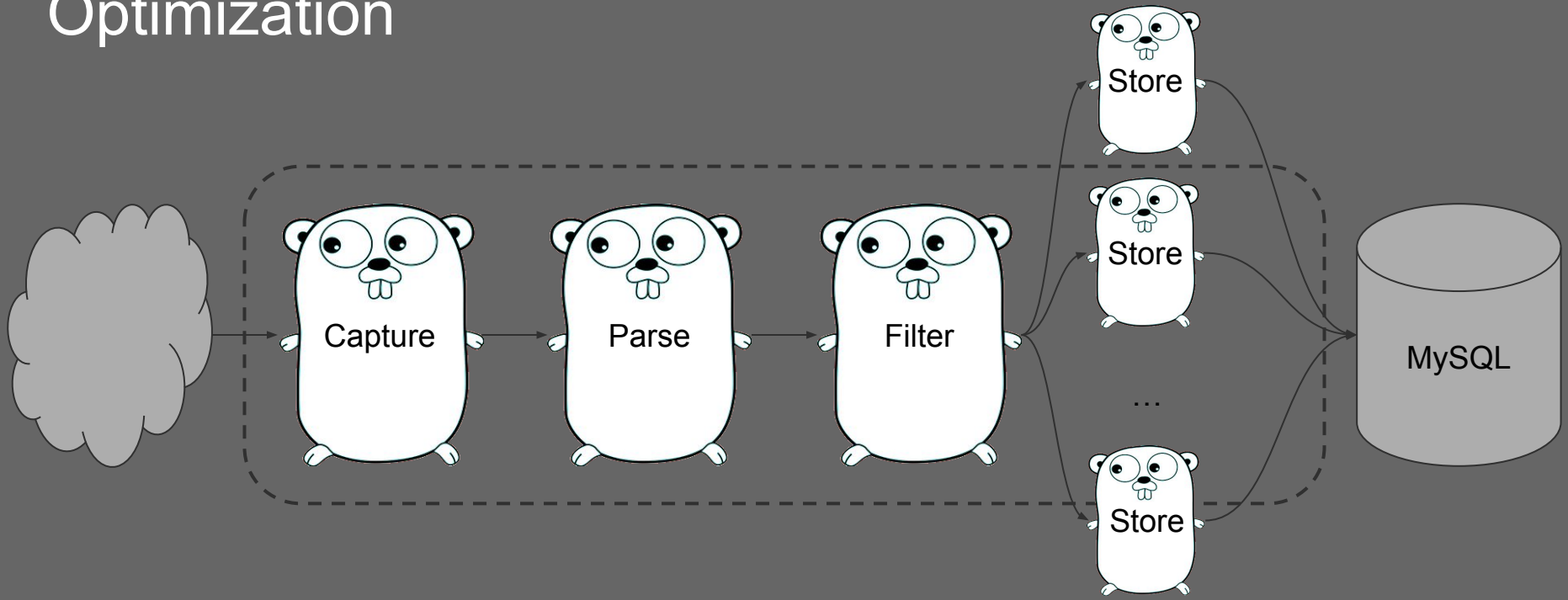
Optimization



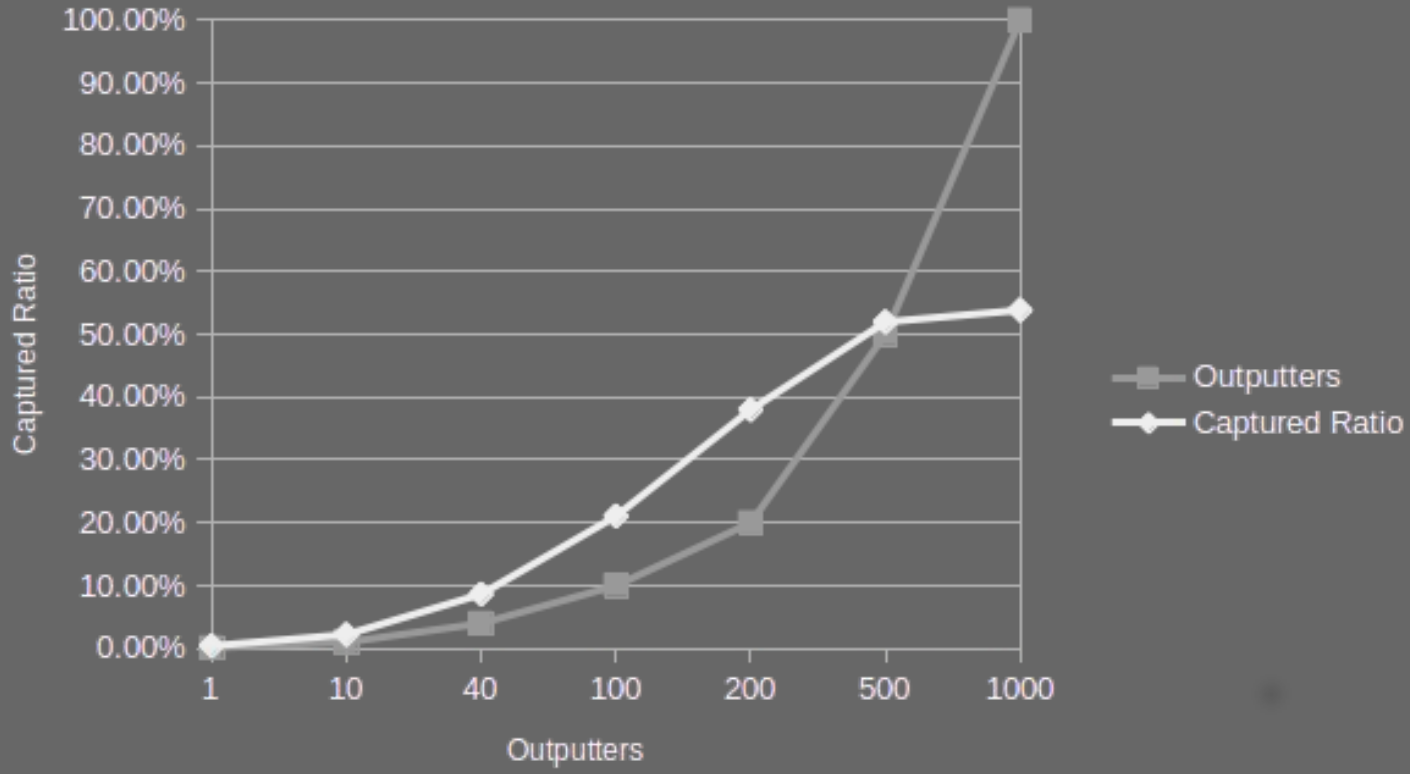
Optimization



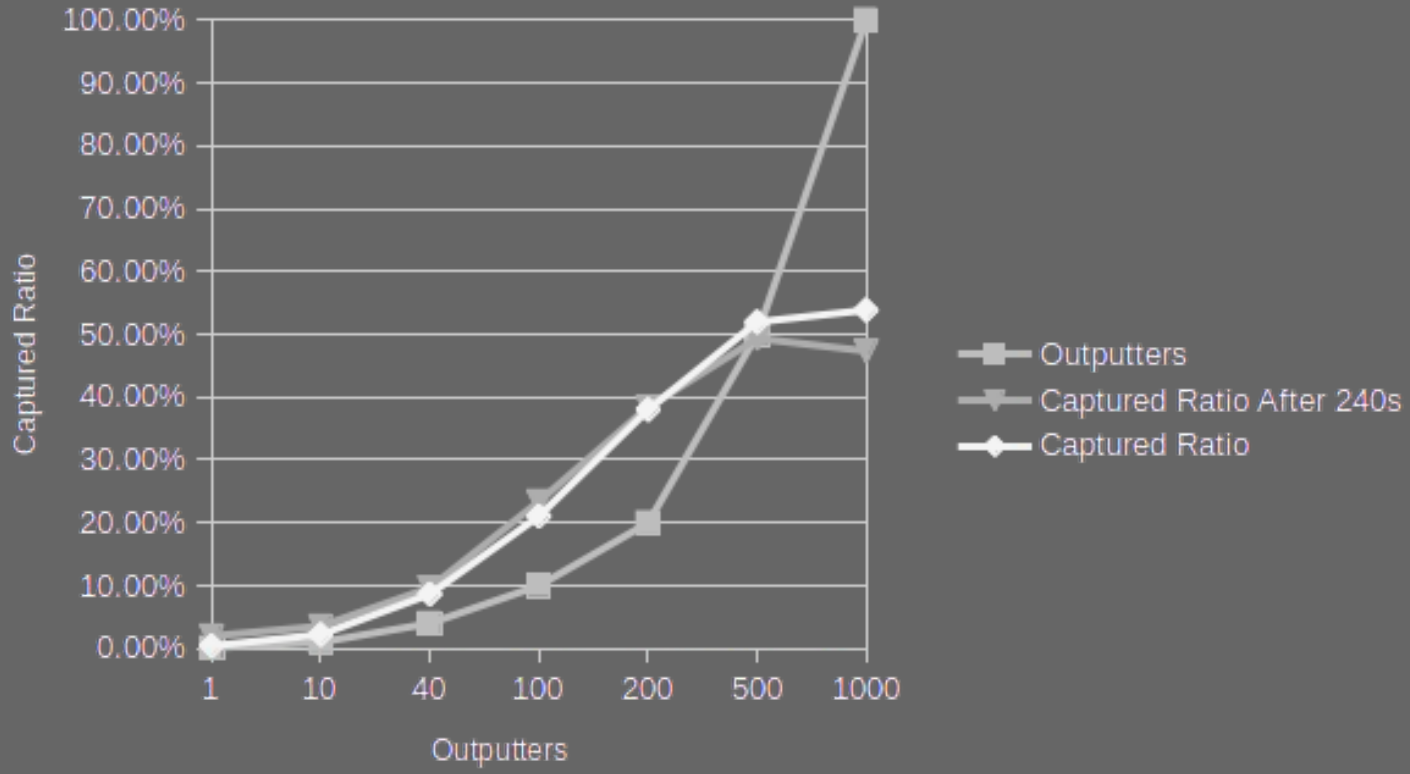
Optimization



Outputters' Effect on Capture Ratios



Outputters' Effect on Capture Ratios Over Time



Buffer(s)

- There appears to be buffering
- Sources
  - Capturer itself
  - `gopacket` / `libpcap`
  - Operating system

Buffer(s)

- There appears to be buffering
- Sources
  - **Capturer itself: Nope**
  - `gopacket` / `libpcap`
  - Operating system

Buffer(s)

- There appears to be buffering
- Sources
  - Capturer itself
  - **`gopacket` / `libpcap`**

```
func (p *PacketSource) Packets() chan Packet {
    if p.c == nil {
            p.c = make(chan Packet, 1000)
            go p.packetsToChannel()
    }
    return p.c
}
```

  - Operating system

Buffer(s)

- There appears to be buffering
- Sources
  - Capturer itself
  - **`gopacket` / `libpcap`**

```
func (p *PacketSource) Packets() chan Packet {
    if p.c == nil {
        p.c = make(chan Packet, 1000)
        go p.packetsToChannel()
    }
    return p.c
}
```

  - Operating system

Buffer(s)

- There appears to be buffering
- Sources
  - Capturer itself
  - `gopacket` / `libpcap`
  - **Operating system**

On Linux, `rmem_default`:

```
$ cat /proc/sys/net/core/rmem_default
212992
```

On BSD, `net.bpf.size/maxsize`.

Optimization

Profiling

- Misleading
- Helpful
- In Between

Profiling

- Misleading

```
go tool pprof -cum -top cap pcap.prof | head
210ms of 210ms total (  100%)
      flat  flat%   sum%        cum   cum%
         0     0%     0%          140ms 66.67%  runtime.goexit
         0     0%     0%          120ms 57.14%  github.com/google/gopacket.(*PacketSource).NextPacket
         0     0%     0%          120ms 57.14%  github.com/google/gopacket.(*PacketSource).packetsToChannel
         0     0%     0%          110ms 52.38%  github.com/google/gopacket/pcap.(*Handle).ReadPacketData
         0     0%     0%          110ms 52.38%  github.com/google/gopacket/pcap.(*Handle).getNextBufPtrLocked
      10ms  4.76%  4.76%          110ms 52.38%  github.com/google/gopacket/pcap._Cfunc_pcap_next_ex
      80ms 38.10% 42.86%          100ms 47.62%  runtime.cgocall
         0     0% 42.86%    40ms 19.05%   runtime._System
```

- Helpful
- In Between

Profiling

- Misleading
- Helpful

```
$ go tool pprof -cum -top cap pcap.prof | head
80ms of 80ms total (  100%)
      flat  flat%   sum%      cum   cum%
      0     0%    0%          50ms 62.50%  runtime.goexit
      0     0%    0%          30ms 37.50%  runtime.gcDrain
      0     0%    0%          20ms 25.00%  runtime.gcBgMarkWorker
      20ms 25.00% 25.00%      20ms 25.00%  runtime.scanobject
      10ms 12.50% 37.50%      20ms 25.00%  runtime.systemstack
      0     0% 37.50%   10ms 12.50%  database/sql.(*Stmt).Exec
      0     0% 37.50%   10ms 12.50%  database/sql.resultFromStatement
      0     0% 37.50%   10ms 12.50%  github.com/go-sql-driver/mysql.(*buffer).fill
```

- In Between

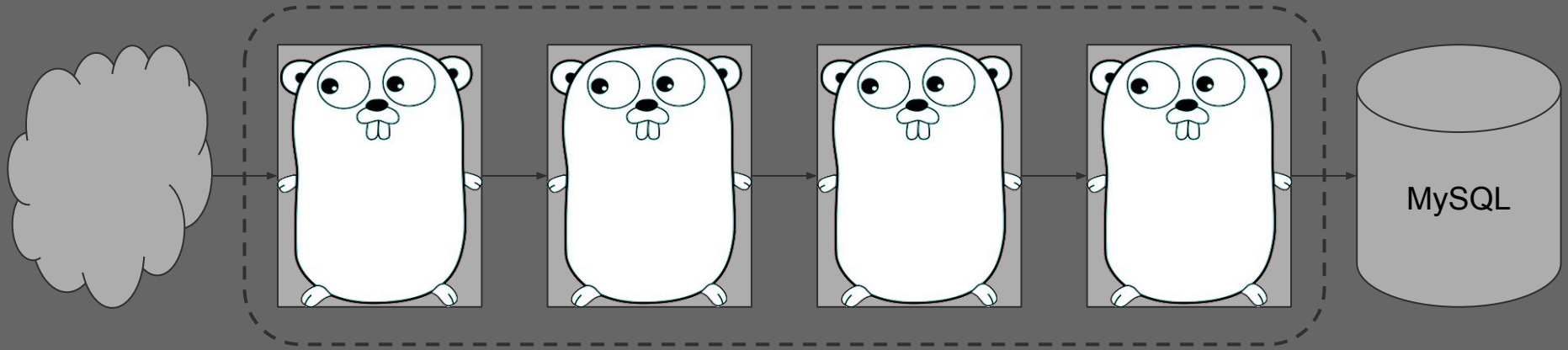
Profiling

- Misleading
- Helpful
- In Between

```
$ go tool pprof -cum -top cap pcap.prof | head
730ms of 1120ms total (65.18%)
Showing top 80 nodes out of 150 (cum >= 20ms)
      flat  flat%   sum%        cum   cum%
         0     0%     0%        810ms 72.32%  runtime.goexit
         0     0%     0%        360ms 32.14%  github.com/google/gopacket.(*PacketSource).NextPacket
         0     0%     0%        360ms 32.14%  github.com/google/gopacket.(*PacketSource).packetsToChannel
         0     0%     0%        300ms 26.79%  github.com/google/gopacket/pcap.(*Handle).ReadPacketData
         0     0%     0%        300ms 26.79%  github.com/google/gopacket/pcap.(*Handle).getNextBufPtrLocked
         0     0%     0%        300ms 26.79%  main.db_extract_output
         0     0%     0%        280ms 25.00%  github.com/google/gopacket/pcap._Cfunc_pcap_next_ex
```

Profiling

- What is the cause of bogus profiling data?
- Think about the order of events



Profiling

- What is the cause of bogus profiling data?
- Think about the order of events

time	Capture	Parse	Filter	Store
0	`read()`	`select()`	`select()`	`select()`
1	`read()`	`select()`	`select()`	`select()`
2	`read()`	`select()`	`select()`	`select()`
3	`read()`	`select()`	`select()`	`select()`
4	`read()`	`select()`	`select()`	`select()`

Working.

Profiling

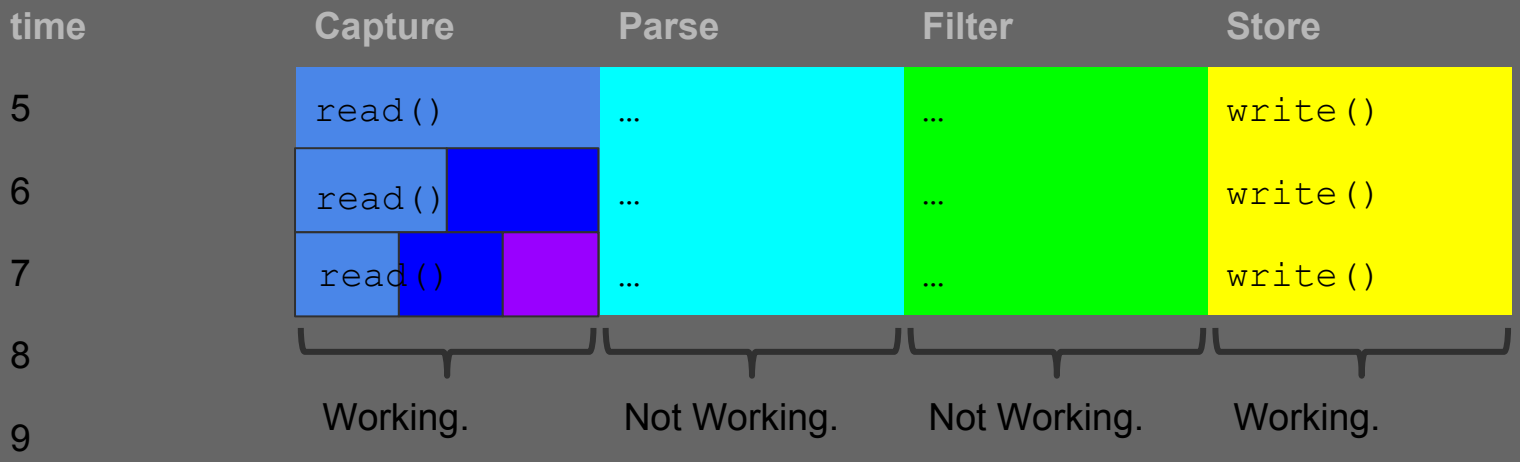
- What is the cause of bogus profiling data?
- Think about the order of events

Data starts →

time	Capture	Parse	Filter	Store
0	`read()`	`select()`	`select()`	`select()`
1	`read()`	`select()`	`select()`	`select()`
2	`read()`	`receive()`	`select()`	`select()`
3	`read()`	`receive()`	`receive()`	`select()`
4	`read()`	`receive()`	`receive()`	`receive()`

Profiling

- What is the cause of bogus profiling data?
- Think about the order of events



Profiling

- What is the cause of bogus profiling data?
- Think about the order of events

Data starts →

time	Capture	Parse	Filter	Store
0	`read()`	`select()`	`select()`	`select()`
1	`read()`	`select()`	`select()`	`select()`
2	`read()`	`receive()`	`select()`	`select()`
3	`read()`	`receive()`	`receive()`	`select()`
4	`read()`	`receive()`	`receive()`	`receive()`

Manual Profiling

- The equivalent of `printf()` debugging
- Remove components until performance improves

Manual Profiling

- The equivalent of `printf()` debugging
- Remove components until performance improves

Manual Profiling

- The equivalent of `printf()` debugging
- Remove components until performance improves

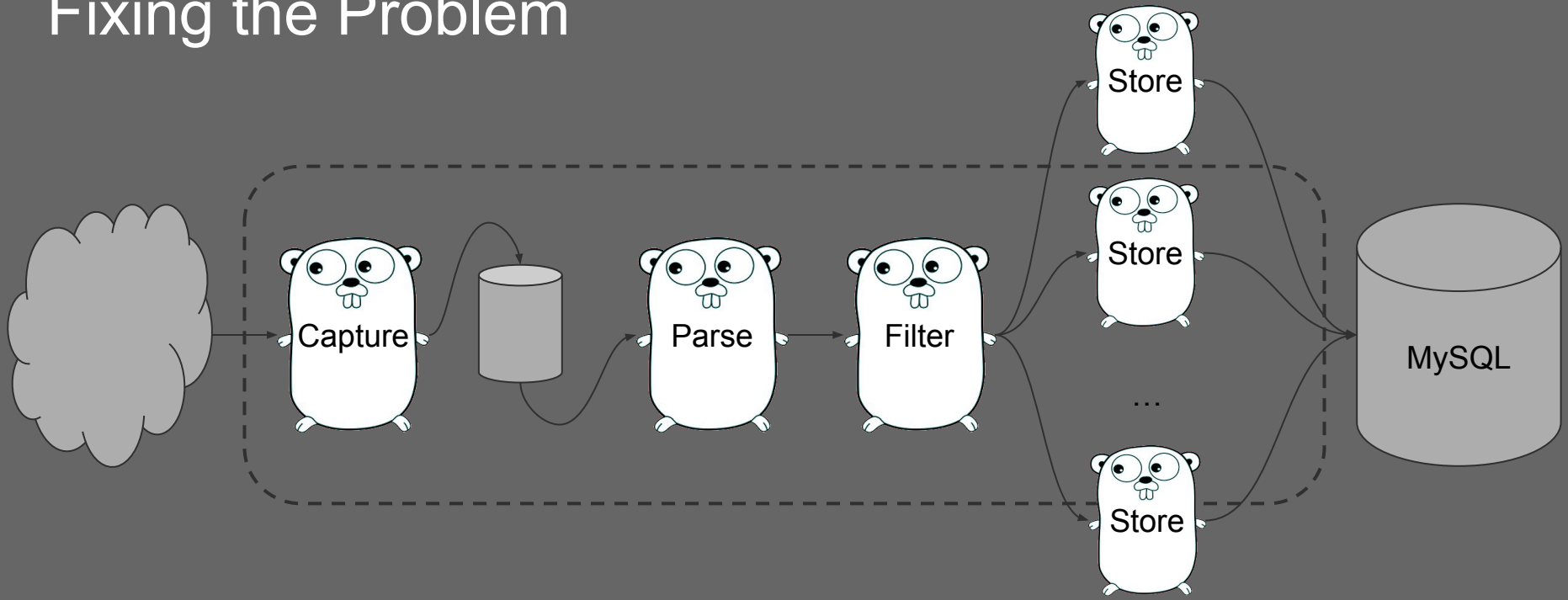
```
func log_cgc_packet(packet cgc_utils.IdsPacket, statement sql.Stmt) {
     _, err := statement.Exec(packet.Csid,
                              packet.ConnectionID,
                              fmt.Sprintf("%v", packet.ConnectionSide),
                              packet.MessageID,
                              packet.Message)
     if err != nil {
          fmt.Printf("Exec() error: %v\n", err);
     }
}
```

Manual Profiling

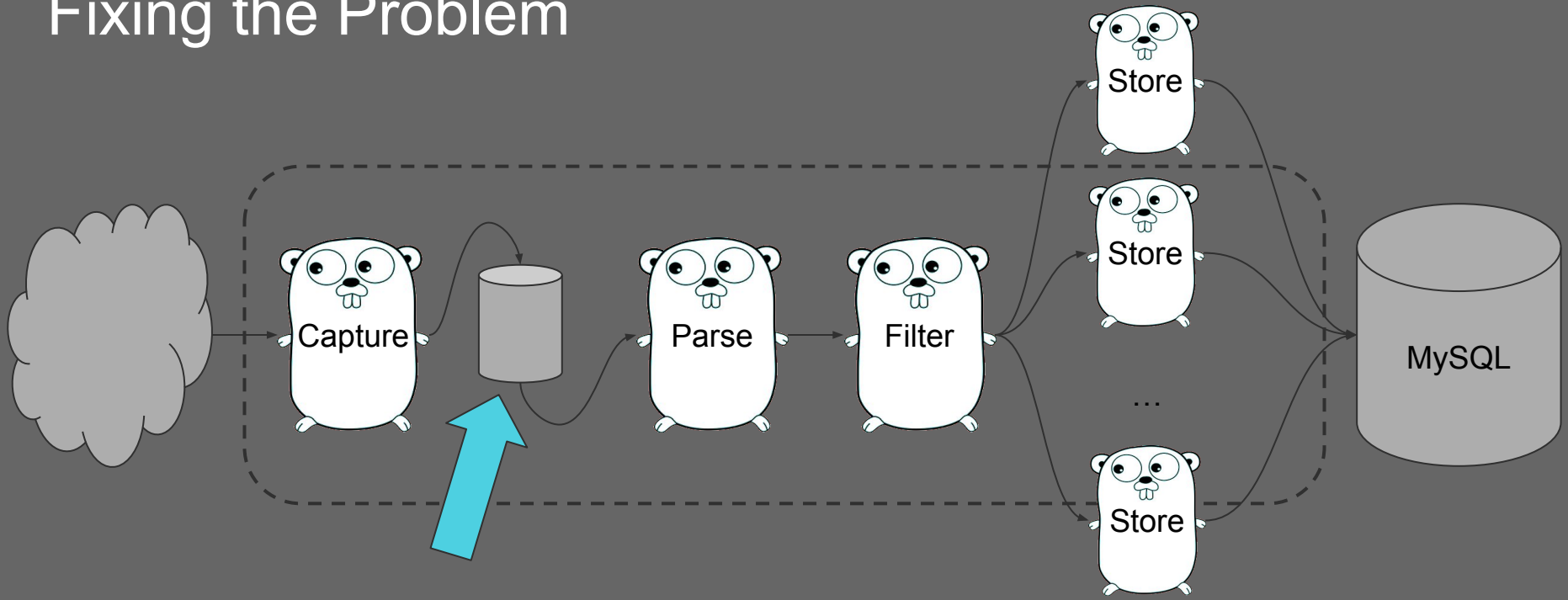
- The equivalent of `printf()` debugging
- Remove components until performance improves

```
func log_cgc_packet(packet cgc_utils.IdsPacket, statement sql.Stmt) {
    return
    if _, err := statement.Exec(packet.Csid,
                                packet.ConnectionID,
                                fmt.Sprintf("%v", packet.ConnectionSide),
                                packet.MessageID,
                                packet.Message); err != nil {
        fmt.Printf("Exec() error: %v\n", err)
    }
}
```

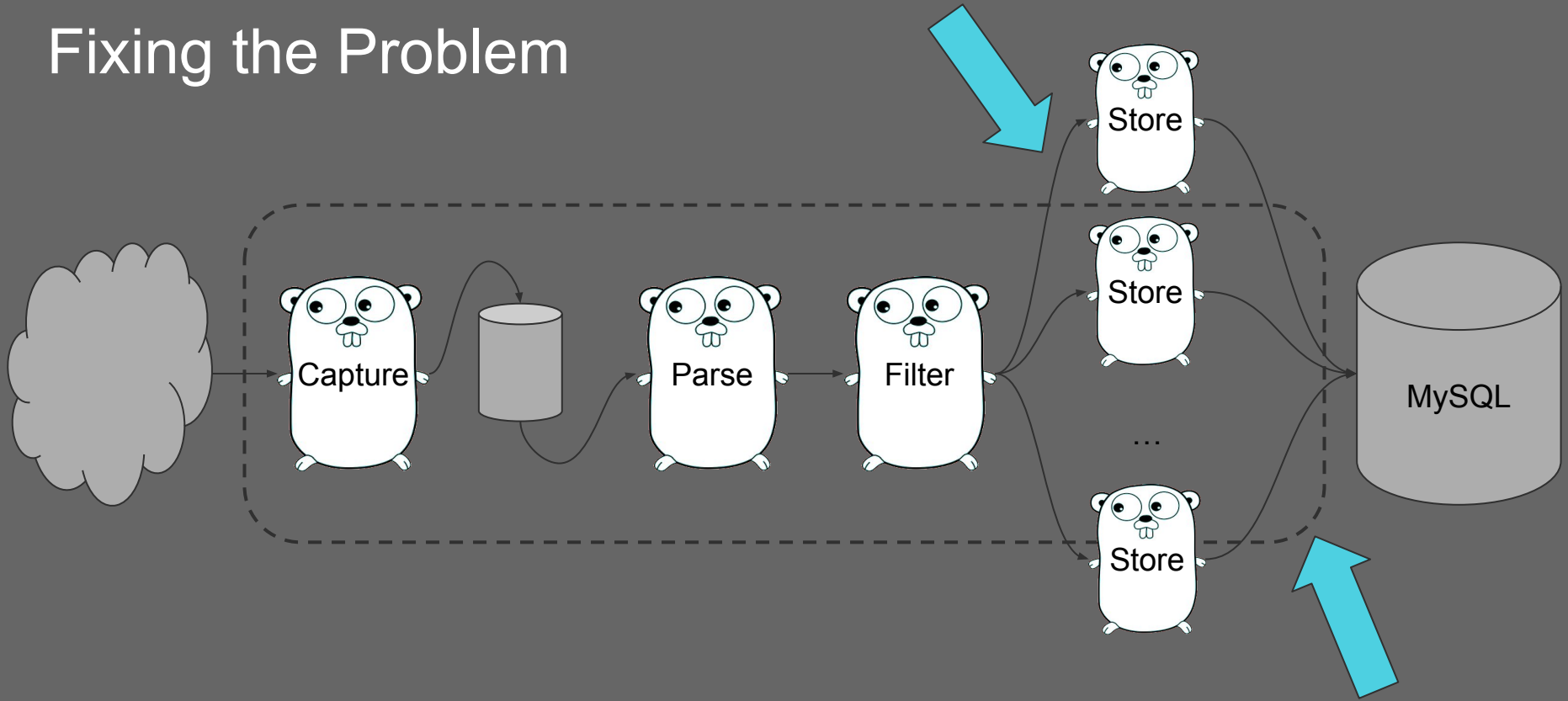
Fixing the Problem



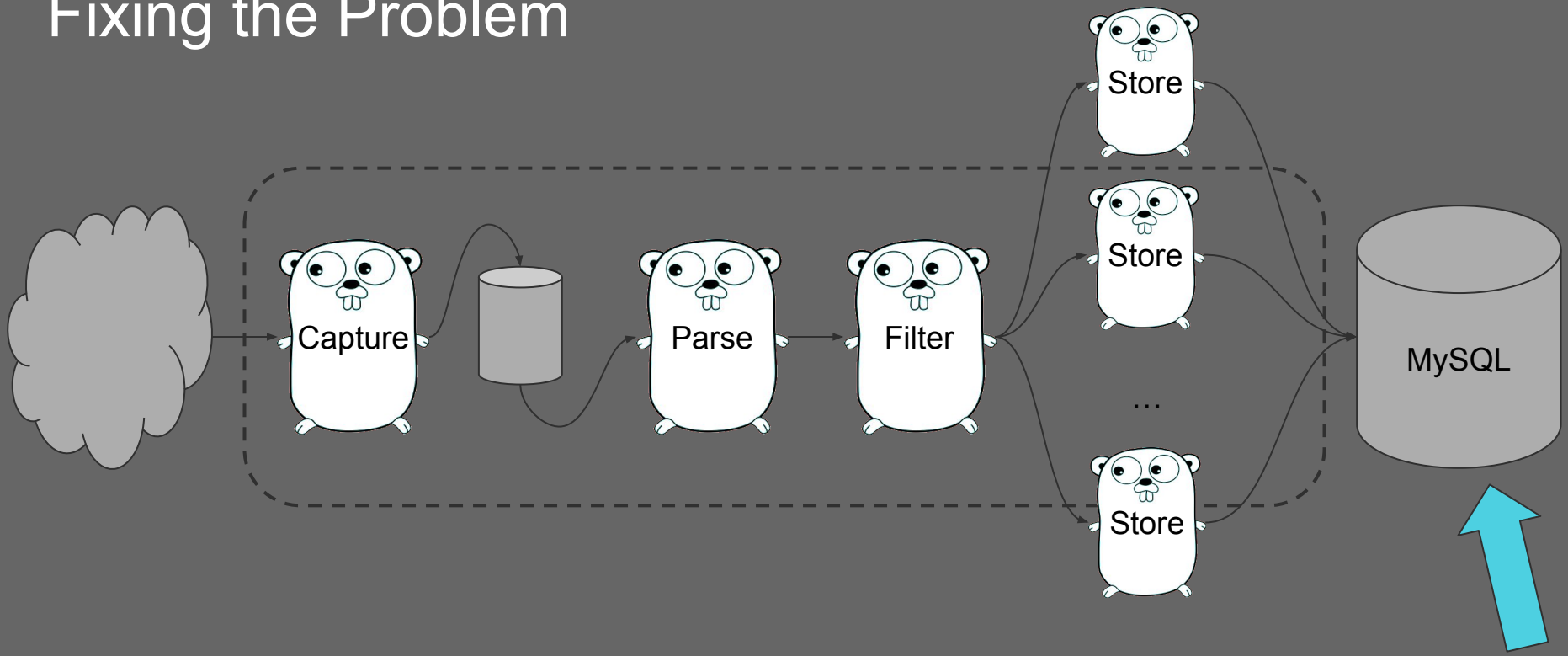
Fixing the Problem



Fixing the Problem



Fixing the Problem



#winning


CYBER
GRAND_CHALLENGE
XANDRA
AUGUST 2016 | LAS VEGAS, NV
DARPA

Questions