Almost Good Enough to Scale:
A Lua Mail Handler and Spam Filter

Norman Ramsey

Tufts University

USA

Fidelis Assis

Embratel

Brazil

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

Lua makes part-time collaboration hard

Ground rules:

- **For both collaborators, a hobby project**
- **Interruptions frequent — often sustained**

Tough with 10,000 lines of code

Modular reasoning enables scaling

Crucial idea from 1970s

To understand, debug, or evolve code:

- **Understand package in isolation**
- **Know only interfaces (APIs) of packages it requires**

Modular reasoning enables scaling

Crucial idea from 1970s

To understand, debug, or evolve code:

- **Understand package in isolation**
- **Know only interfaces (APIs) of packages it requires**

Problem: Lua cannot express an API

Modular reasoning enables scaling

Crucial idea from 1970s

To understand, debug, or evolve code:

- **Understand package in isolation**
- **Know only interfaces (APIs) of packages it requires**

Problem: Lua cannot express an API

- **A Lua package is not a module**
- **Must examine source code of most packages**

Modular reasoning enables scaling

Crucial idea from 1970s

To understand, debug, or evolve code:

- **Understand package in isolation**
- **Know only interfaces (APIs) of packages it requires**

Problem: Lua cannot express an API

- **A Lua package is not a module**
- **Must examine source code of most packages**

Result: Work with code every day, or you lose

APIs need backing from Lua Team

Like packages, crucial for interoperability

- **The Lua Team must set a standard**

APIs need backing from Lua Team

Like packages, crucial for interoperability

- **The Lua Team must set a standard**

What to try?

APIs need backing from Lua Team

Like packages, crucial for interoperability

- **The Lua Team must set a standard**

What to try?

- **Not LuaDoc, Oxygen, and so on**

APIs need backing from Lua Team

Like packages, crucial for interoperability

- **The Lua Team must set a standard**

What to try?

- **Not LuaDoc, Oxygen, and so on**
- **Not a language extension**

APIs need backing from Lua Team

Like packages, crucial for interoperability

- **The Lua Team must set a standard**

What to try?

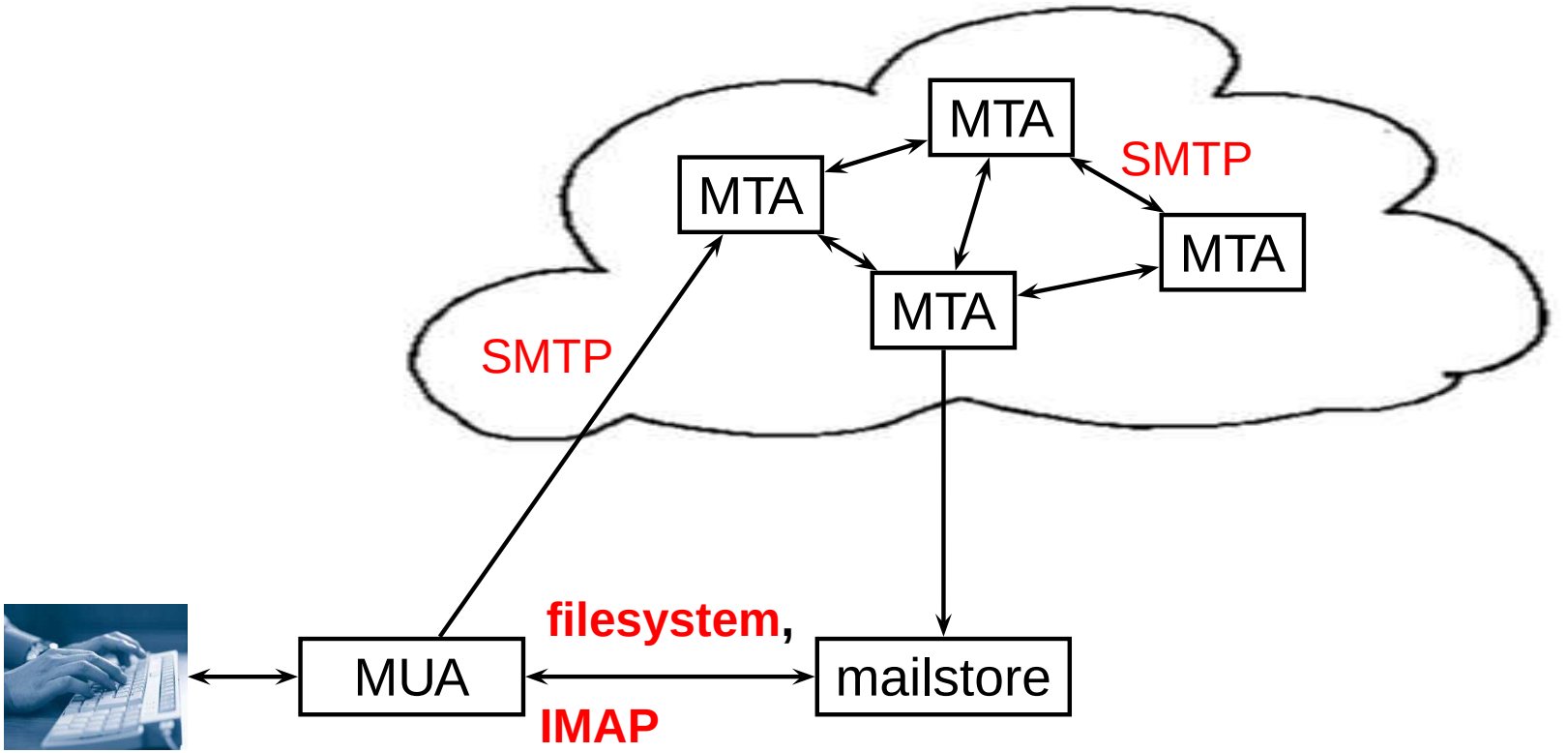
- **Not LuaDoc, Oxygen, and so on**
- **Not a language extension**
- **For now, a Lua data structure (baby steps)**

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

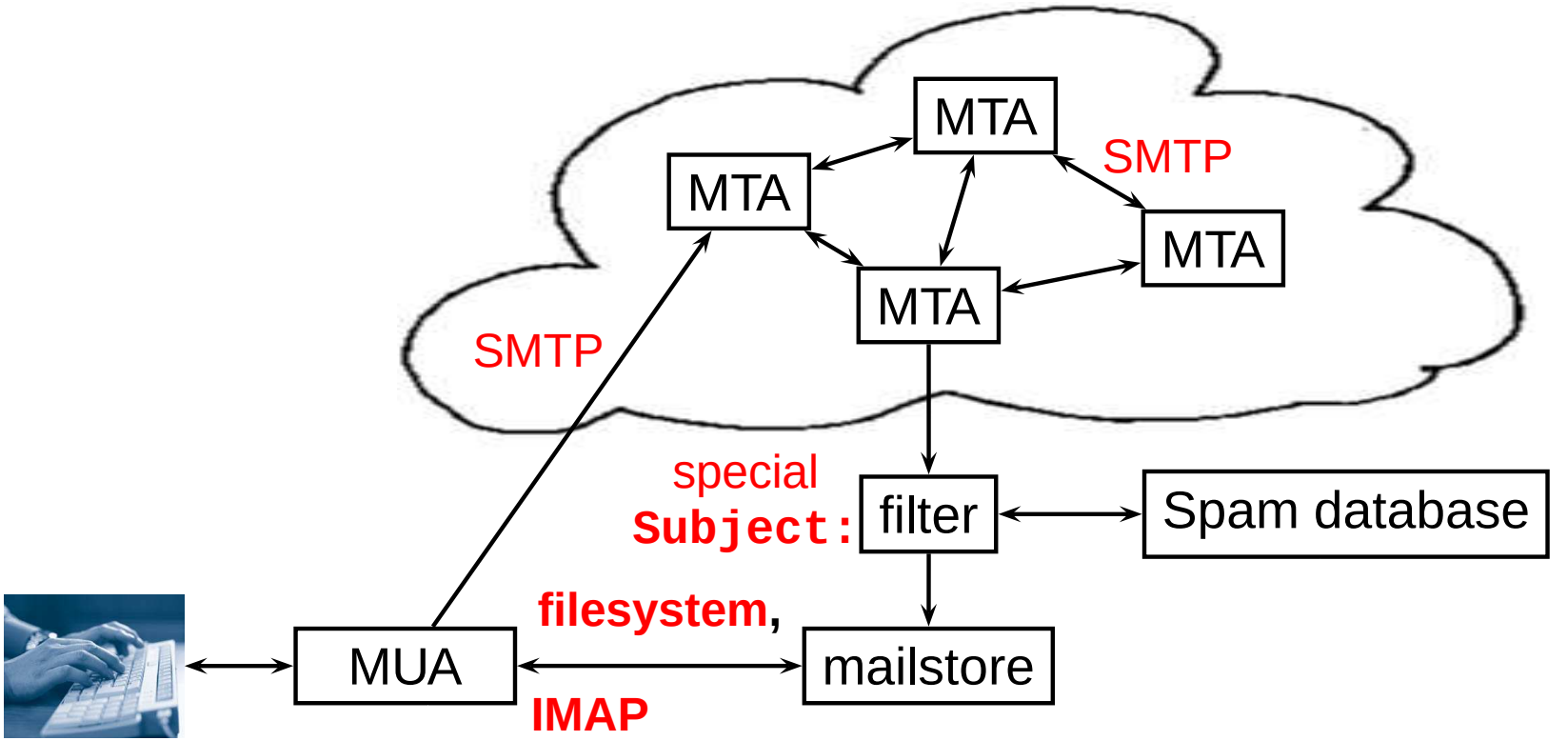
Internet mail has at least four players

Many, many interfaces:



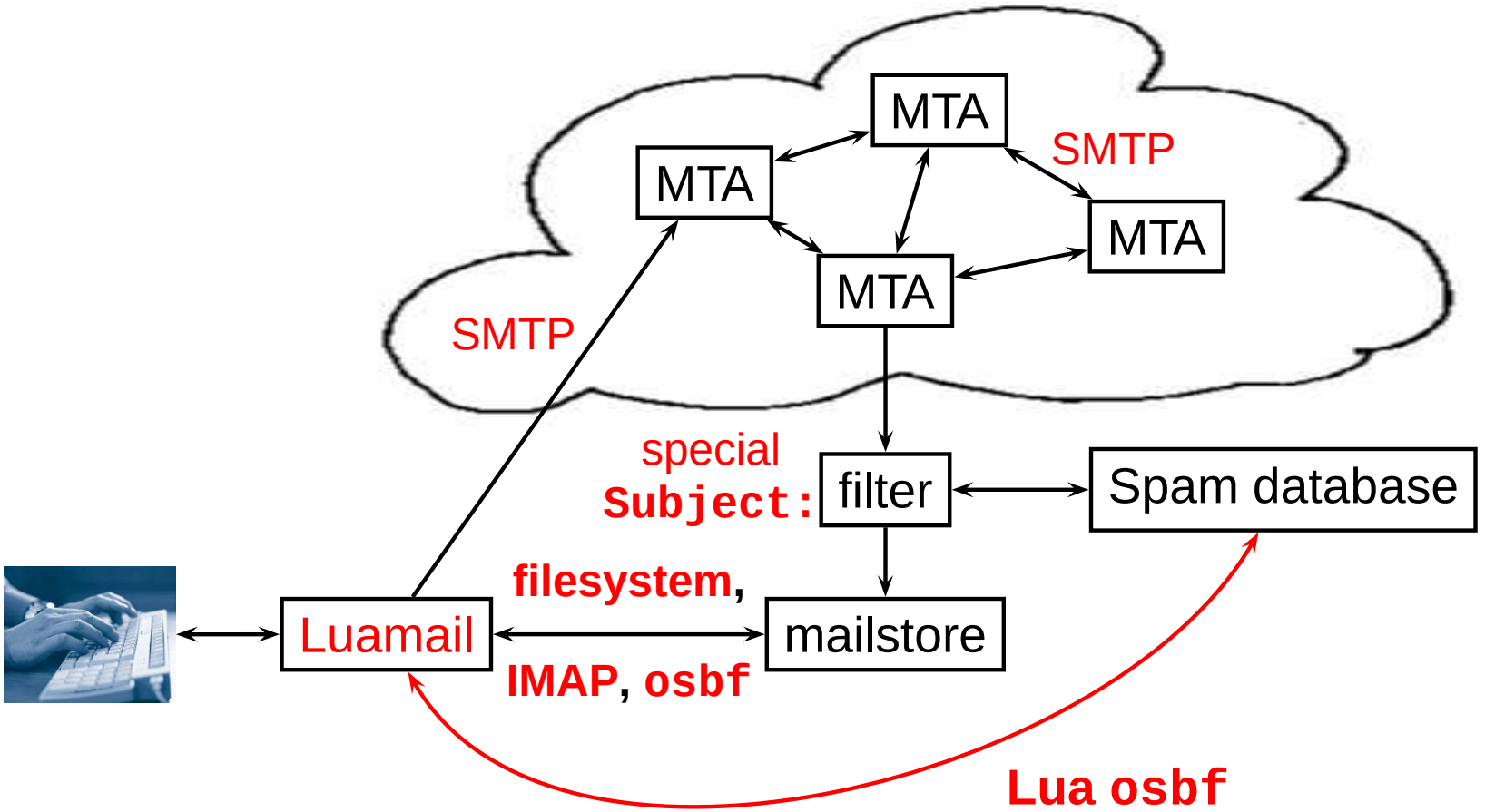
Internet mail has at least six players

Many, many interfaces:



Internet mail has at least six players

Many, many interfaces:



OSBF-Lua delivers a triple interface

For existing users:

- **The email interface**

For Luamail:

- **Unix command-line interface**

For embedding:

- **Lua API for classification, learning, filtering**

OSBF-Lua delivers a triple interface

For existing users:

- **The email interface**

For Luamail:

- **Unix command-line interface**

For embedding:

- **Lua API for classification, learning, filtering**

The hard part is the API

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

How we got here

2004: Assis's Orthogonal Sparse Bigram Filter

- **In CRM114: 3× accuracy, 6× speed, $\frac{1}{10}$ space**

2004: Ramsey starts building Luamail

2005: Assis creates OSBF-Lua

- **C code from CRM-114**
- **Standalone mail filter in Lua**

2006: OSBF-Lua wins spam contest at TREC

2007: Assis and Ramsey join forces

Luamail is a better MH

Luamail is a better MH

From MH:

- **No “Mail User Agent” as such**

Luamail is a better MH

From MH:

- **No “Mail User Agent” as such**
- **Mail commands are shell commands**

Luamail is a better MH

From MH:

- **No “Mail User Agent” as such**
- **Mail commands are shell commands**
- **Mailstore is filesystem (with bugs!)**

Luamail is a better MH

From MH:

- **No “Mail User Agent” as such**
- **Mail commands are shell commands**
- **Mailstore is filesystem (with bugs!)**
- **Uses MH parts (in C)**

Luamail is a better MH

From MH:

- **No “Mail User Agent” as such**
- **Mail commands are shell commands**
- **Mailstore is filesystem (with bugs!)**
- **Uses MH parts (in C)**

From Lua: customized user experience

In 2008, work in progress

Mostly completed, but still in flight:

1. **Refactor OSBF-Lua:**
   - **More Lua, less C**
   - **Break into reusable parts (packages)**
2. **From packages, external API for clients**
3. **Build command-line client over API**
4. **Migrate Luamail to use `osbf` APIs**

In 2008, work in progress

Mostly completed, but still in flight:

1. **Refactor OSBF-Lua:**
   - **More Lua, less C**
   - **Break into reusable parts (packages)**
2. **From packages, external API for clients**
3. **Build command-line client over API**
4. **Migrate Luamail to use `osbf` APIs**

Major barrier:

understanding and refactoring our own APIs

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

Our errors in brief

I take most of the blame:

- Streams
- Objects
- Feared bad performance using Lua
- Thought mail parsing was hard
- Tried to reuse existing parsers

Our errors in brief

I take most of the blame:

- **Streams**
- **Objects**
- **Feared bad performance using Lua**
- **Thought mail parsing was hard**
- **Tried to reuse existing parsers**

In our defense:

Our errors in brief

I take most of the blame:

- Streams
- Objects
- Feared bad performance using Lua
- Thought mail parsing was hard
- Tried to reuse existing parsers

In our defense:

- PUC Rio advocates streams (LuaSocket mail)

Our errors in brief

I take most of the blame:

- **Streams**
- **Objects**
- **Feared bad performance using Lua**
- **Thought mail parsing was hard**
- **Tried to reuse existing parsers**

In our defense:

- **PUC Rio advocates streams (LuaSocket mail)**
- **PIL objects the *only* worked example of abstraction**

Our errors in brief

I take most of the blame:

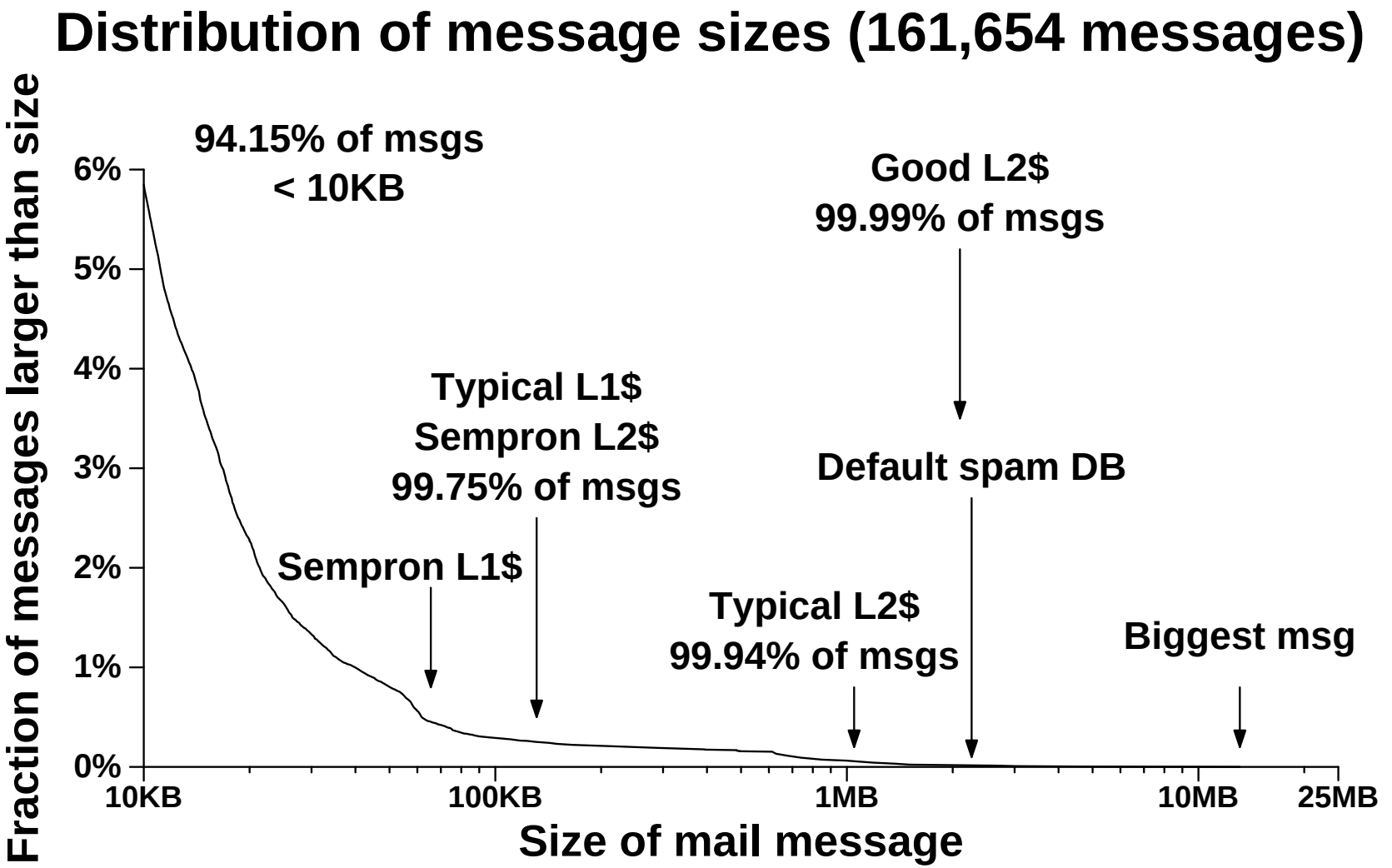
- **Streams**
- **Objects**
- **Feared bad performance using Lua**
- **Thought mail parsing was hard**
- **Tried to reuse existing parsers**

In our defense:

- **PUC Rio advocates streams (LuaSocket mail)**
- **PIL objects the *only* worked example of abstraction**
- **Existing parsers used streams and objects**

Streams and Objects Considered Harmful (in two slides)

Almost all mail fits on chip



Why streams are bad

Why streams are bad

State of your scanner not in the program counter

Why streams are bad

State of your scanner not in the program counter

- Requires explicit representation of state: objects
- 1,400 lines Lua with streams, ∼30 lines without

Why streams are bad

State of your scanner not in the program counter

- Requires explicit representation of state: objects
- 1,400 lines Lua with streams, ∼30 lines without

String matching doesn't work on streams!

Good stuff

We've done three good things

Since February 2007

- **More Lua, less C, better performance**
- **Refactored Lua into better design**
- **Useful representations of APIs**

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

Lua is (mostly) more than fast enough

To understand, a short explanation of How It Works

Spam database stores probability

Convert email to set of "sparse bigrams" $\{B\}$

For each bigram B, we store

- Probability that B appears in document drawn at random from spam corpus
- Probability that B appears in document drawn at random from ham corpus

Data structure is hash table (Bloom filter)

OSBF filter computes log odds

Compute p_{spam} and p_{ham}, then

$$\text{“Spamminess”} = \log \frac{p_{spam}}{1 - p_{spam}}$$

$$\text{“Hamminess”} = \log \frac{p_{ham}}{1 - p_{ham}}$$

Highest confidence (log odds) wins

OSBF filter computes log odds

Compute p_{spam} and p_{ham}, then

$$\text{“Spamminess”} = \log \frac{p_{spam}}{1 - p_{spam}}$$

$$\text{“Hamminess”} = \log \frac{p_{ham}}{1 - p_{ham}}$$

Highest confidence (log odds) wins

Log odds are good for reporting to users

- **Highest odds is always most likely**
- **Highest odds always nonnegative**
- **High odds zero is meaningful (no information)**

Log odds extends to multiple classes

Currently filtering my mail as follows:

- **Spam**
- **Ham**
- **Calls for papers**
- **Talk announcements**
- **Funding opportunities**
- **Job postings**
- **Mailing-list info**
- **Cold calls from people seeking jobs**
- **My employer's financial system**

Base of logarithm can be chosen freely

Higher log odds mean greater confidence

Users want to quantify confidence

Base of logarithm can be chosen freely

Higher log odds mean greater confidence

Users want to quantify confidence

After experiments, log base 49:

- **Log odds $<$ 10 = low confidence**
- **Log odds $>$ 20 = high confidence**
- **Otherwise, depends on training**

Base of logarithm can be chosen freely

Higher log odds mean greater confidence

Users want to quantify confidence

After experiments, log base 49:

- Log odds $<$ 10 = low confidence
- Log odds $>$ 20 = high confidence
- Otherwise, depends on training

User must train “on or near error”

- Messages that are incorrectly classfied
- Messages with low confidence

Lua can do almost everything

Clean and easy in Lua:

- **Confidence**
- **Classes and training thresholds**
- **Extract text to classify, train; get p_{class} back**
- **Message headers**
- **Filtering, logging**
- **Cache of recent messages (for training)**
- **Whitelist, blacklist**
- **Much of multiclassifier**

Performance comes from profiling

Not from indiscriminate use of C!

Performance comes from profiling

Not from indiscriminate use of C!

- **Avoid using `mmap` with writable hash tables**
- **Inline and cache hash-table index computations**
- **Parse message headers using state machine, not Lua string matching (touch each character in header once!)**

Performance comes from profiling

Not from indiscriminate use of C!

- **Avoid using `mmap` with writable hash tables**
- **Inline and cache hash-table index computations**
- **Parse message headers using state machine, not Lua string matching (touch each character in header once!)**

TREC 2006, 2.2MB database, 37,822 messages:

- **Released OSBF 2.0.4: 69 classifications/s**
- **Alpha OSBF 3: 274 classifications/s**

Performance comes from profiling

Not from indiscriminate use of C!

- **Avoid using `mmap` with writable hash tables**
- **Inline and cache hash-table index computations**
- **Parse message headers using state machine, not Lua string matching (touch each character in header once!)**

TREC 2006, 2.2MB database, 37,822 messages:

- **Released OSBF 2.0.4: 69 classifications/s**
- **Alpha OSBF 3: 274 classifications/s**

Next goal: 9 classes in L2$! (18MB today)

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

API documentation is Lua value

Every package has table documenting its contents

Access using `internals` command:

```
 % osbf3 internals
Documented modules:
  boot
  cache
  cfg
  command_line
  commands
  core
  filter
  lists
  log
  mime
  msg
  options
  output
  roc
  sfid
  util

Undocumented functions:
  options.env_default
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Abbreviated doco for package overview

Use `-short` to get one line per function (some functions omitted here)

```
 % osbf3 internals -short msg
msg: T = The representation of a message
msg.add_header = function(T, tag, contents)
msg.del_header = function(T, tag, ...)
msg.headers_tagged = fun(msg, tag, ...) returns itera
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
msg.to_orig_string = function(T) returns string
msg.synopsis = function(T, w) returns string
```

Full documentation for a package value

```
 % osbf3 internals msg.synopsis
msg.synopsis = function(T, w) returns string
    Returns a string of width w (default 60)
    which is a synopsis of the message T.
    The synopsis is formed from the Subject:
    line and the first few words of the body.

 % scan last:6   # luamail: synopsis on right
3751   Fri  To:fidelis@pobox.  your affiliat
3752  02:01 Mark P Jones       Re: ICFP 2007
3753  03:07 "David Tranah"     Re: your infa
3754  05:00 <info@newegg.com>  Your Newegg.c
3755  05:51 "David Tranah"     Re: ICFP 2007
3756  00:22 Cron Daemon        Cron <nr@drdo
```

Synopsis extra valuable for training

Low-confidence messages sorted by log odds:

```
 % newmail
+in/train has 9 messages (1-9).
1 [spam? (11)] <oanacristina.p  America the Beautiful<<American Indepe
2 [spam? (14)] dennis arthur    URGENT FROM MR ARTHUR DENNIS<<M: Dear,
3 [spam? (15)] <elipower69@bel  Proud to be an American<<The best of 4
4 [spam? (16)] "Demetrius Kirk  get out of debt loans<<Debt Freedom in
5 [spam? (17)] Stephan Dougher  Customer Notice: Pharmacy Reminder<<M:
6 [spam? (17)] Victoria Smith   CONTRACT/INHERITANCE FUND<< Attn:Sir,
7 [spam? (20)] Alice Lacy       Fwd.YourPharmaacyOrder no.032028<<M: U
8 [spam? (20)] Bacso            Oil prices drop to new low<<M: Get sma
9 [talks? (23] kkk5z@cs.virgin  [cs-announce] come to tea time!<<MH:
```

Training works on mailstore and spam DB

```
 % class spam            # choose those messages classified as spam
1 [spam? (11)] <oanacristina.p  America the Beautiful<<American Indepe
2 [spam? (14)] dennis arthur    URGENT FROM MR ARTHUR DENNIS<<M: Dear,
3 [spam? (15)] <elipower69@bel  Proud to be an American<<The best of 4
4 [spam? (16)] "Demetrius Kirk  get out of debt loans<<Debt Freedom in
5 [spam? (17)] Stephan Dougher  Customer Notice: Pharmacy Reminder<<M:
6 [spam? (17)] Victoria Smith   CONTRACT/INHERITANCE FUND<< Attn:Sir,
7 [spam? (20)] Alice Lacy       Fwd.YourPharmaacyOrder no.032028<<M: U
8 [spam? (20)] Bacso            Oil prices drop to new low<<M: Get sma
 % confirm it && rmm it # confirm and remove them
1 is spam, confidence 12.5 [train] (at arrival: spam, confidence 11.5)
Adding 1 to spam corpus
1: Trained as spam: confidence 12.47 -> 39.81
    ...
```

Training works on mailstore and spam DB

```
 % class spam            # choose those messages classified as spam
1 [spam? (11)] <oanacristina.p  America the Beautiful<<American Indepe
2 [spam? (14)] dennis arthur    URGENT FROM MR ARTHUR DENNIS<<M: Dear,
3 [spam? (15)] <elipower69@bel  Proud to be an American<<The best of 4
4 [spam? (16)] "Demetrius Kirk  get out of debt loans<<Debt Freedom in
5 [spam? (17)] Stephan Dougher  Customer Notice: Pharmacy Reminder<<M:
6 [spam? (17)] Victoria Smith   CONTRACT/INHERITANCE FUND<< Attn:Sir,
7 [spam? (20)] Alice Lacy       Fwd.YourPharmaacyOrder no.032028<<M: U
8 [spam? (20)] Bacso            Oil prices drop to new low<<M: Get sma
 % confirm it && rmm it # confirm and remove them
1 is spam, confidence 12.5 [train] (at arrival: spam, confidence 11.5)
Adding 1 to spam corpus
1: Trained as spam: confidence 12.47 -> 39.81
    ...
```

Training can be complicated

Experimental multiclassifier not so trustworthy

```
+in/train has 9 messages (1-9).
   1  [jobs? (1) ] Marcelo Fiore      [TYPES/announce] Three papers.<<
   2  [cfp? (1)  ] "Llacer, Gregory   PRISE Distinguished Speaker Seri
   3  [jobs? (4) ] Benjamin Pierce    [TYPES/announce] Types Considere
   4  [spam? (14)] atlas@towncountry  Getway Specials from the Town an
   5  [spam? (19)] "ALLIANCE FINANCE  LOAN OFFER!!!<<Alliance Finance.
   6  [cold-calls] "Mega Millions"    TOUTES NOS FELICITATIONS !!!!!!!
   7  [work? (35)] "Whitney Smith"    Gentle Giant Moving Company <<M:
   8  [talks? (36] Krasimira Kapitan  [cs-announce] GESC Bar Night - Ju
   9  [work? (46)] "Whitney Smith"    Gentle Giant Moving Company <<M:
```

Outline

- Conclusions: Lua and modularity
- Introduction to Internet mail
- Lua email handling and spam filtering
- Our mistakes using Lua
- Success #1: How filtering works (fast!)
- Success #3: Moving toward modularity (with screen shots of mail)
- Future directions for modularity

Next step for API: dynamic checking

How can we check these functions?

```
msg.synopsis  = function(T, w) returns string
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
```

Next step for API: dynamic checking

How can we check these functions?

```
msg.synopsis  = function(T, w) returns string
msg.of_string = function(s, sure) returns T or nil
msg.to_string = function(T) returns string
```

Interpret arguments, results as predicates on v:

```
check.T ≡ type(v) == 'table' and getmetatable(v) == mm
check.w ≡ type(v) == 'number' and v > 0
check.string ≡ type(v) == 'string'
check.sure   ≡ v == nil or type(v) == 'boolean'
```

Build predicates using “validator”

$\tau \Rightarrow$ `boolean` | `number` | `string` | `userdata`

- | `table(`*t*`, `*exact*`)` type for each field in t
- | `list(`τ`)` keys 1 to #x validate against τ
- | `optional(`τ`)` validates against τ or is nil
- | `eq(v)` equal to value v
- | `having_metatable(meta)`
- | `union(`τ`, `τ`)` either of two types
- | `inter(`τ`, `τ`)` both of two types (e.g., table and list)
- | `by_function(f, expected, fname)`

Function predicates require wrapping

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Dynamically wrap functions: contracts

```
to_string =
  function(x, ...)
    if select('#', ...) > 0 then
      violation('Too many arguments to to_string')
    elseif not check.T(x) then
      violation('First argument to to_string is not of type T')
    else
      local function check_result(ok, s, ...)
        if not ok then
          violation('to_string called error()')
        elseif select('#', ...) > 0 then
          violation('Too many results from to_string')
        elseif not check.string(s) then
          violation('Result of to_string is not a string')
        else
          return s
        end
      end
      return check_result(pcall(orig_to_string, x))
    end
```

Goal: Generate this function from __doc string!

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

`osbf3 internals` is a lifesaver, but we want more:

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

`osbf3 internals` is a lifesaver, but we want more:

- Full agreement on representation of APIs

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

`osbf3 internals` is a lifesaver, but we want more:

- Full agreement on representation of APIs
- APIs to be checked

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

`osbf3 internals` is a lifesaver, but we want more:

- Full agreement on representation of APIs
- APIs to be checked
- Eventually, static checking

Modular reasoning can really help

Lua is great fun, but it can be difficult to use at scale

`osbf3 internals` is a lifesaver, but we want more:

- Full agreement on representation of APIs
- APIs to be checked
- Eventually, static checking

We want the Lua Team to lead us

Modularity road map

Have: Exportable APIs, not machine-checked

Wanted: Machine-checkable APIs including types

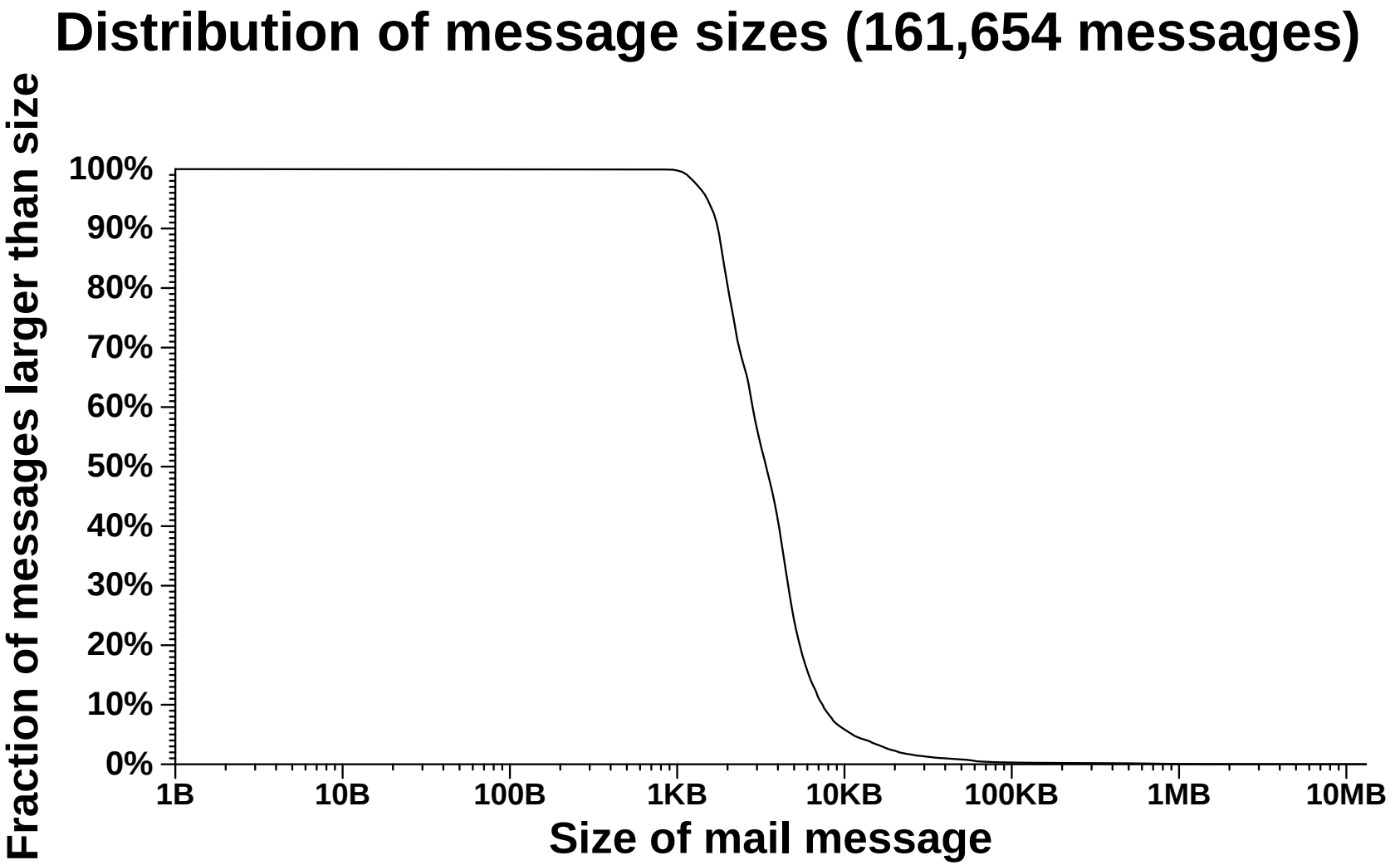
Wanted: Dynamic checks for APIs (contracts)

Wanted: Whole-program static analysis to find bugs

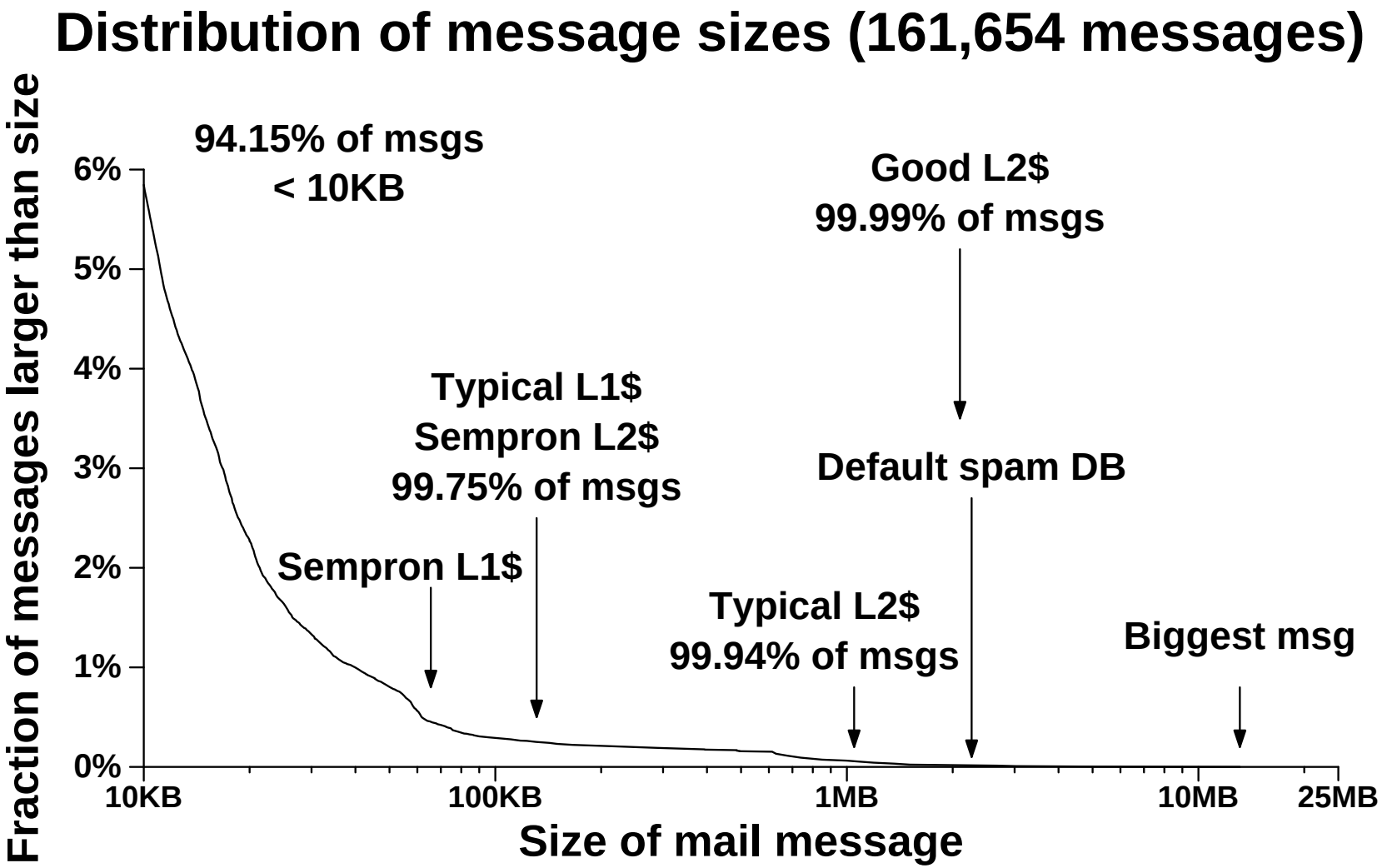
Wanted: Modular static analysis to find bugs

Wanted: Static type inference

Why streams are unnecessary

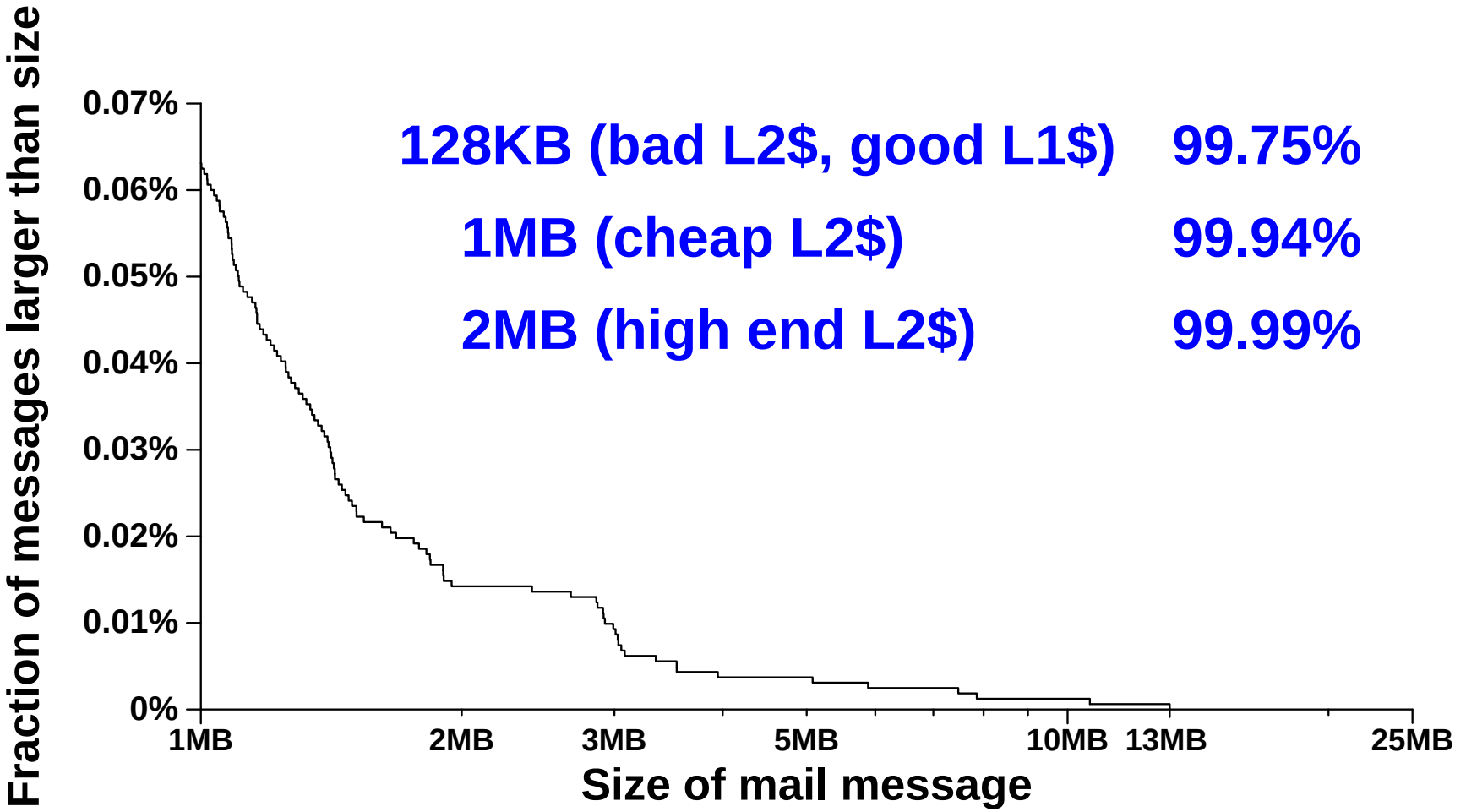


Almost all mail fits on chip



The tail is nearly irrelevant

Even biggest mails fit memory (25MB hard stop)



128KB (bad L2$, good L1$)	99.75%
1MB (cheap L2$)	99.94%
2MB (high end L2$)	99.99%