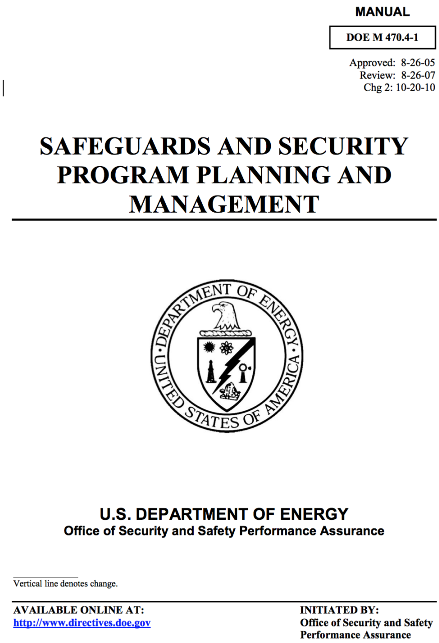
VAST: Visibility Across Space and Time
Architecture & Usage

Matthias Vallentin
matthias@bro.org

University of California, Berkeley

LBNL
July 22, 2014



MANUAL

DOE M 470.4-1

Approved: 8-26-05
Review: 8-26-07
Chg 2: 10-20-10

SAFEGUARDS AND SECURITY PROGRAM PLANNING AND MANAGEMENT

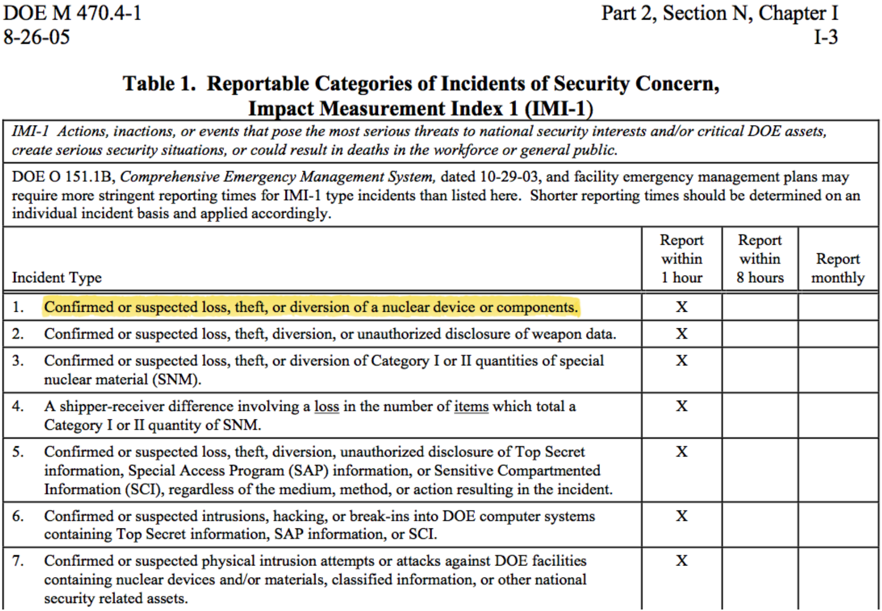
U.S. DEPARTMENT OF ENERGY

Office of Security and Safety Performance Assurance

Vertical line denotes change.

AVAILABLE ONLINE AT:
http://www.directives.doe.gov

INITIATED BY:
Office of Security and Safety Performance Assurance



DOE M 470.4-1
8-26-05

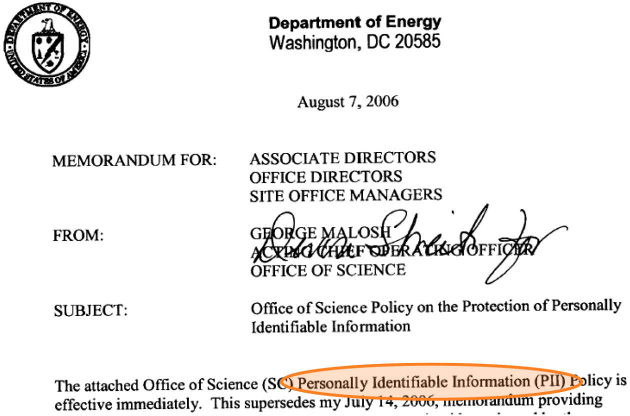
Part 2, Section N, Chapter I
I-3

Table 1. Reportable Categories of Incidents of Security Concern, Impact Measurement Index 1 (IMI-1)

IMI-1 Actions, inactions, or events that pose the most serious threats to national security interests and/or critical DOE assets, create serious security situations, or could result in deaths in the workforce or general public.

DOE O 151.1B, *Comprehensive Emergency Management System,* dated 10-29-03, and facility emergency management plans may require more stringent reporting times for IMI-1 type incidents than listed here. Shorter reporting times should be determined on an individual incident basis and applied accordingly.

Incident Type	Report within 1 hour	Report within 8 hours	Report monthly
1. Confirmed or suspected loss, theft, or diversion of a nuclear device or components.	X		
2. Confirmed or suspected loss, theft, diversion, or unauthorized disclosure of weapon data.	X		
3. Confirmed or suspected loss, theft, or diversion of Category I or II quantities of special nuclear material (SNM).	X		
4. A shipper-receiver difference involving a loss in the number of items which total a Category I or II quantity of SNM.	X		
5. Confirmed or suspected loss, theft, diversion, unauthorized disclosure of Top Secret information, Special Access Program (SAP) information, or Sensitive Compartmented Information (SCI), regardless of the medium, method, or action resulting in the incident.	X		
6. Confirmed or suspected intrusions, hacking, or break-ins into DOE computer systems containing Top Secret information, SAP information, or SCI.	X		
7. Confirmed or suspected physical intrusion attempts or attacks against DOE facilities containing nuclear devices and/or materials, classified information, or other national security related assets.	X		



Department of Energy
Washington, DC 20585

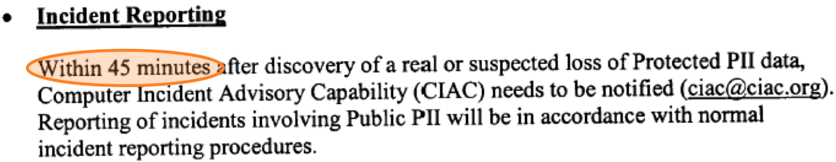
August 7, 2006

MEMORANDUM FOR: ASSOCIATE DIRECTORS
OFFICE DIRECTORS
SITE OFFICE MANAGERS

FROM: GEORGE MALOSH
ACTING CHIEF OPERATING OFFICER
OFFICE OF SCIENCE

SUBJECT: Office of Science Policy on the Protection of Personally Identifiable Information

The attached Office of Science (SC) Personally Identifiable Information (PII) Policy is effective immediately. This supersedes my July 14, 2006, memorandum providing



- **Incident Reporting**

Within 45 minutes after discovery of a real or suspected loss of Protected PII data, Computer Incident Advisory Capability (CIAC) needs to be notified (ciac@ciac.org). Reporting of incidents involving Public PII will be in accordance with normal incident reporting procedures.

Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

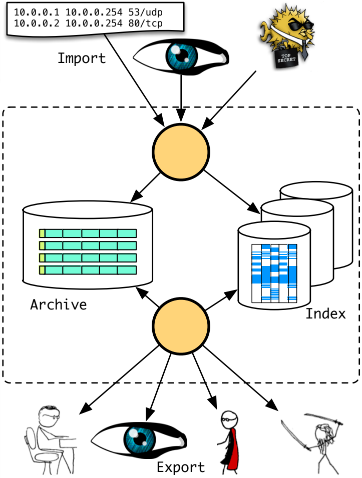
VAST: Visibility Across Space and Time

VAST

A unified platform for network forensics

Goals

- Interactivity
  - Sub-second response times
  - Iterative query refinement
- Scalability
  - Scale with data & number of nodes
  - Sustain high & continuous input rates
- Strong and rich typing
  - High-level types and operations
  - Type safety



```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

VAST & Bro

Bro

- Generates rich-typed logs representing high-level summary of activity

→ How to process these huge piles of logs?

- Fine-grained events exist at runtime only

→ Make ephemeral events persistent?

VAST

- Visibility across **Space**
  - Unified data model: same expressiveness as Bro
  - Combine host-based and network-based activity
- Visibility across **Time**
  - Historical queries: retrieve data from the past
  - Live queries: get notified when new data matches query

VAST & Big Data Analytics

MapReduce (Hadoop)

Batch-oriented processing: *full scan* of data

- + Expressive: no restriction on algorithms
- - Speed & Interactivity: full scan for each query

In-memory Cluster Computing (Spark)

Load full data set into memory and then run query

- + Speed & Interactivity: fast on arbitrary queries over working set
- - Thrashing when working set too large

Distributed Indexing (VAST)

Distributed building and querying of bitmap indexes

- + Fast: only access space-efficient indexes
- + Caching of index hits enables iterative analyses
- - Limited query language (e.g., no joins)

Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

High-Level Architecture of VAST

Import

- Unified data model
- Sources provide events

Archive

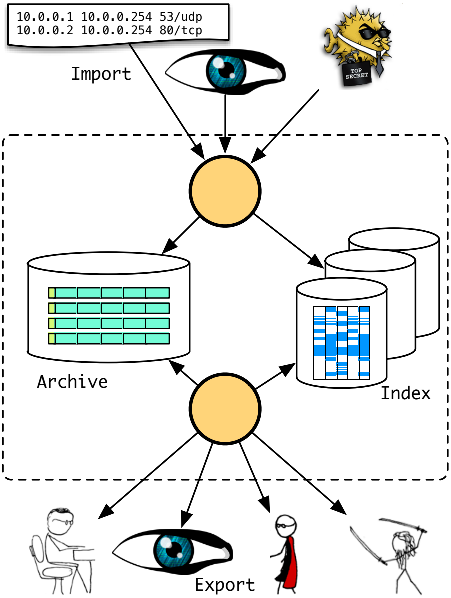
- Stores raw data as events
- Compressed chunks & segments

Index

- Secondary indexes into archive
- Horizontally partitioned

Export

- Interactive query console
- JSON/Bro output



```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

Outline

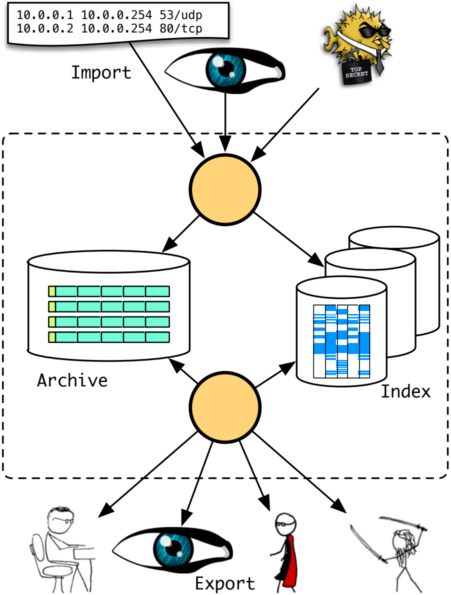
1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

Architecture: Archive & Index

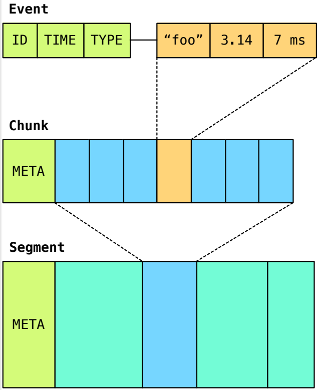


```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

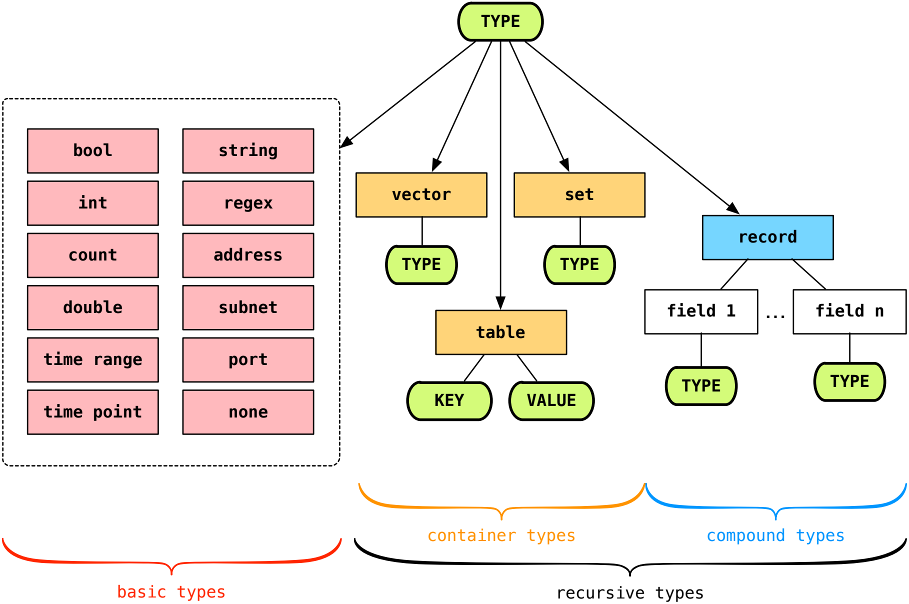
Type System

Terminology

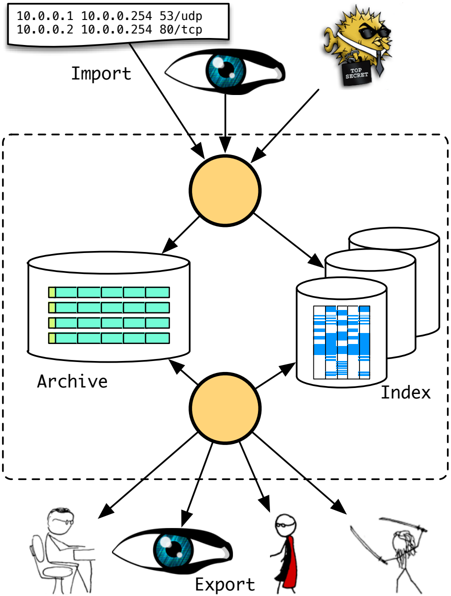
- **Type**: interpretation of data
- **Value**: instance of a type
- **Event**: value + *named* type + meta data
  - A timestamp
  - A unique ID i where $i \in [1, 2^{64} - 2]$
- **Schema**: collection of event types
- **Chunk**: serialized & compressed events
  - Meta data: schema + time range
  - Fixed number of events, variable size
- **Segment**: sequence of chunks
  - Meta data: union of chunk meta data
  - Bound on size, variable number of chunks



Types: Interpretation of Data



Architecture: Archive & Index



```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

Index Hits: Sets of Events

Bitvector: sets of events

- Query result $\equiv$ set of event IDs from $[1, 2^{64} - 2]$

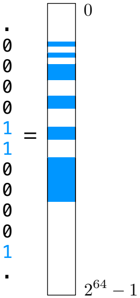
→ Model as **bit vector**: $[4, 7, 8] = 000100110 \cdots$

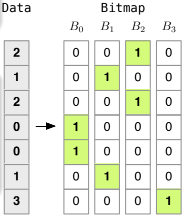
Bitstream: encoded append-only sequence of bits

- EWAH (no patents unlike WAH, PLWAH, COMPAX)
- Compact, space-efficient representation
- Bitwise operations do not require decoding

Bitmap: maps values to bitstreams

- `push_back(T` x`)`: append value x of type `T`
- `lookup(T` x`, Op` $\circ$`)`: get bitstream for x under $\circ$



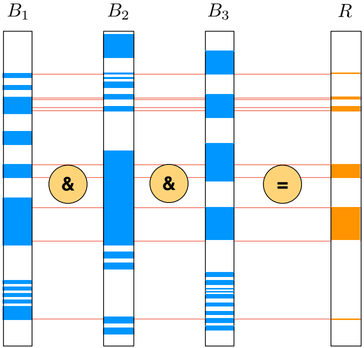


Data	B_0	B_1	B_2	B_3
2	0	0	1	0
1	0	1	0	0
2	0	0	1	0
0	1	0	0	0
0	1	0	0	0
1	0	1	0	0
3	0	0	0	1

Composing Results via Bitwise Operations

Combining Predicates

- Query $Q = X \wedge Y \wedge Z$
  - $x = \texttt{1.2.3.4} \;\wedge\; y < \texttt{42} \;\wedge\; z \in \texttt{"foo"}$
- Bitmap index lookup yields $X \rightarrow B_1$, $Y \rightarrow B_2$, and $Z \rightarrow B_3$
- Result $R = B_1 \,\&\, B_2 \,\&\, B_3$



Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

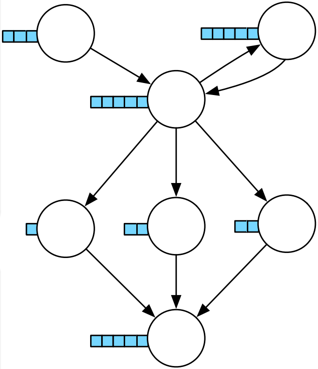
3. Using VAST

4. Demo

Actor Model

Actor: unit of sequential execution

- **Message**: typed tuple $\langle T_0, \ldots, T_n \rangle \ni \mathbb{T}^n$
- **Behavior**: partial function over $\mathbb{T}^n$
- **Mailbox**: FIFO with typed messages ▭▭▭
- Can send messages to other actors
- Can spawn new actors
- Can monitor each actors



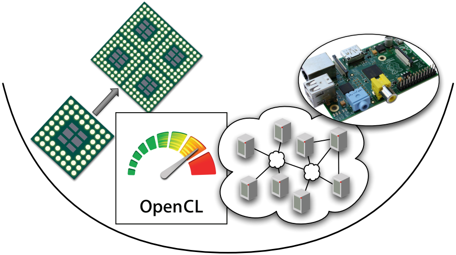
Benefits

- Modular, high-level components
- Robust SW design: no locks, no data races
- Network-transparent deployment
- Powerful concurrency model

CAF: C++ Actor Framework

libcaf

- **Native** implementation of the actor model
- **Strongly typed** actors available → protocol checked at compile-time
- **Pattern matching** to extract messages
- **Transparently** supports heterogeneous components
  - Intra-machine: efficient message passing with copy-on-write semantics
  - Inter-machine: TCP, UDP *(soon)*, multicast *(soon)*
  - Special hardware components: GPUs via OpenCL



https://github.com/actor-framework

Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

Query Language

Boolean Expressions

- Conjunctions `&&`
- Disjunctions `||`
- Negations `!`
- Predicates
  - `LHS op RHS`
  - `(expr)`

Examples

- `A && B || !(C && D)`
- `orig_h == 10.0.0.1 && &time < now - 2h`
- `&type == "conn" || :string +] "foo"`
- `duration > 60s && service == "tcp"`

LHS: Extractors

- `&type`
- `&time`
- `x.y.z.arg`
- `:type`

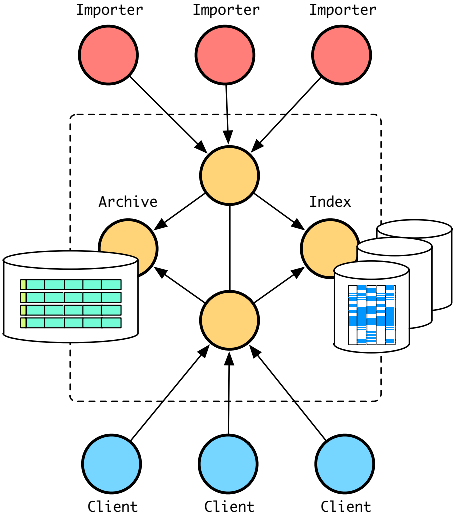
Relational Operators

- `<`, `<=`, `==`, `>=`, `>`
- `in`, `ni`, `[+`, `+]`
- `!in`, `!ni`, `[-`, `-]`
- `~`, `!~`

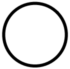
RHS: Value

- `T`, `F`
- `+42`, `1337`, `3.14`
- `"foo"`
- `10.0.0.0/8`
- `80/tcp`, `53/?`
- `{1, 2, 3}`

Query


Importer
Importer
Importer
Archive
Index
Client
Client
Client

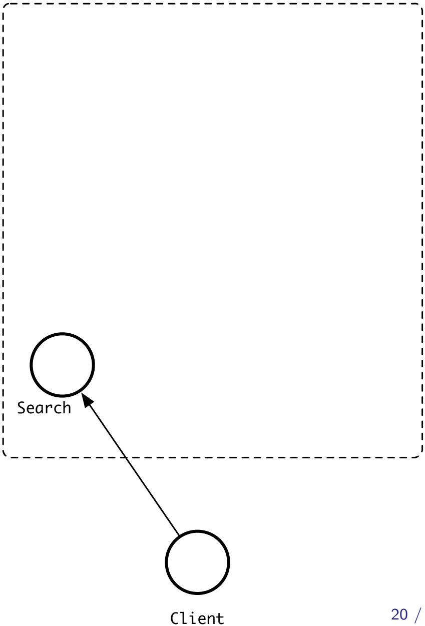
Query



Query

CLIENT

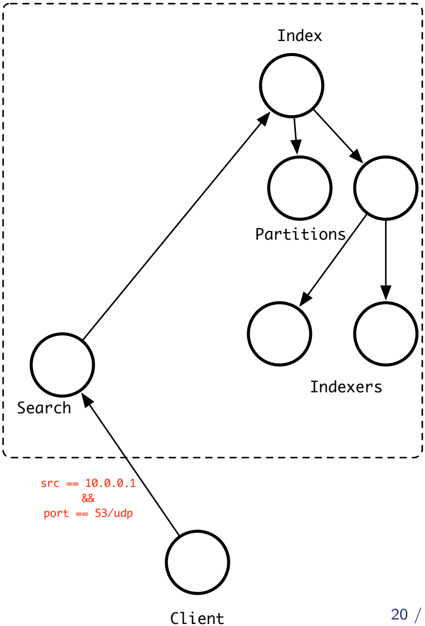
1. Send query string to SEARCH



Query

CLIENT

1. Send query string to SEARCH



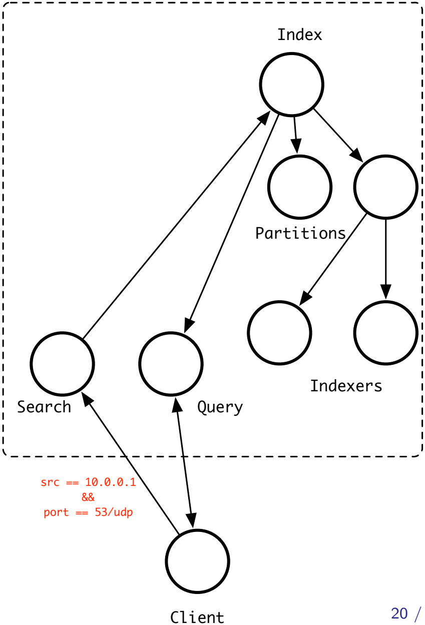
Query

CLIENT

1. Send query string to SEARCH

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY



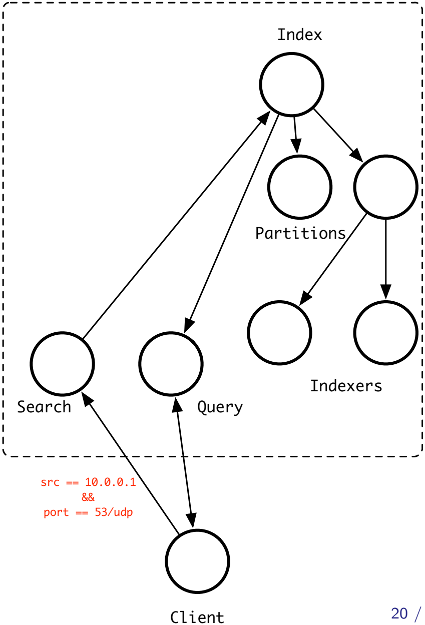
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY



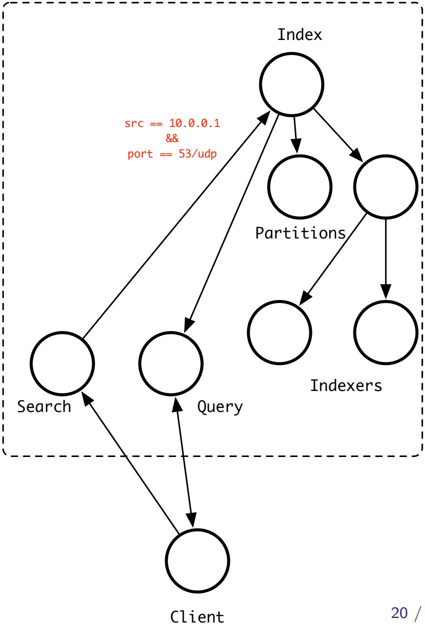
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX



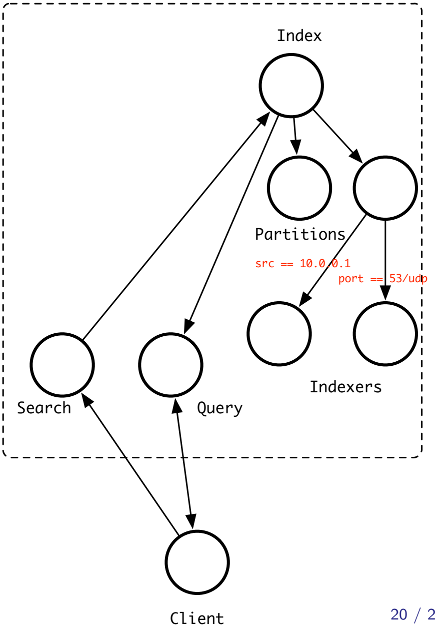
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX



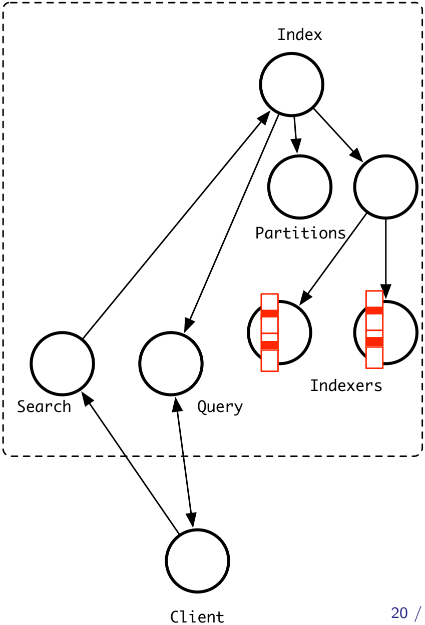
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX



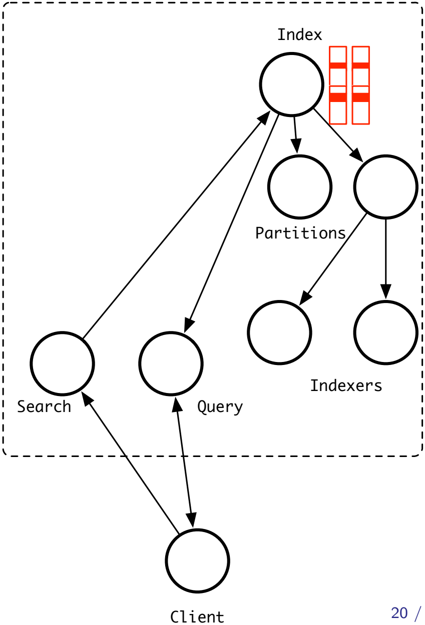
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX



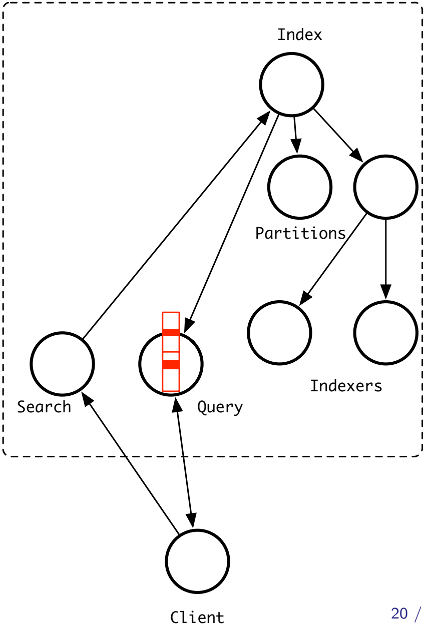
Query

CLIENT

1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX



Query

CLIENT

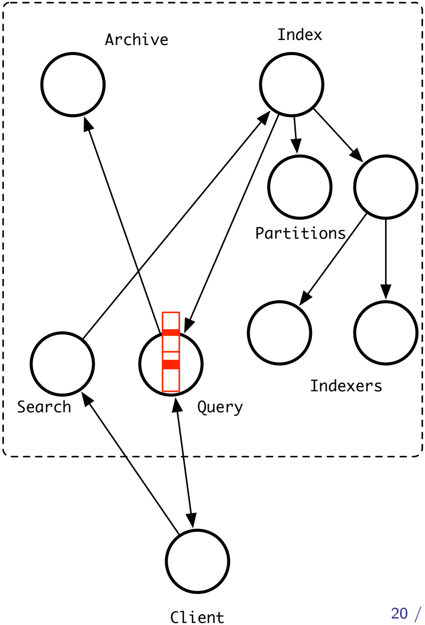
1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX

QUERY

1. Receive hits from INDEX



Query

CLIENT

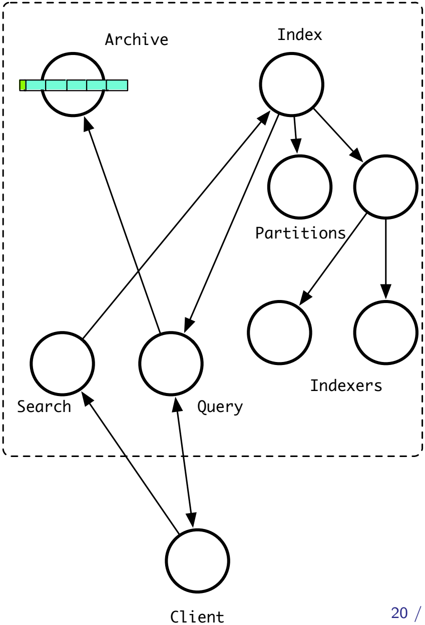
1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX

QUERY

1. Receive hits from INDEX
2. Ask ARCHIVE for segments



Query

CLIENT

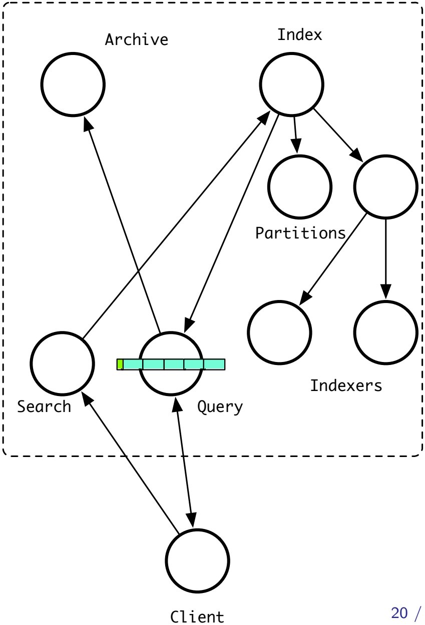
1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX

QUERY

1. Receive hits from INDEX
2. Ask ARCHIVE for segments
3. Extract events, check candidates



Query

CLIENT

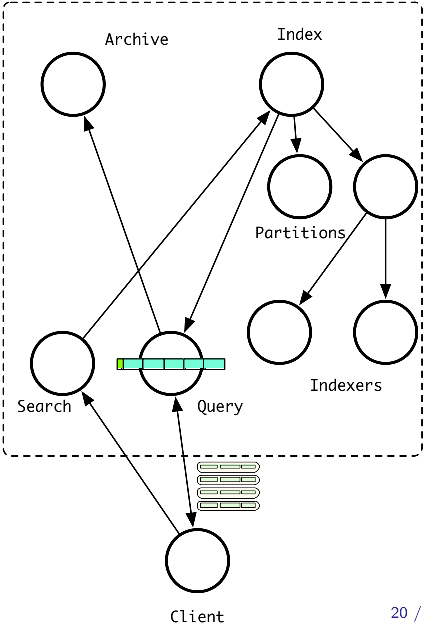
1. Send query string to SEARCH
2. Receive QUERY actor
3. Extract results from QUERY

SEARCH

1. Parse and validate query string
2. Spawn dedicated QUERY
3. Forward query to INDEX

QUERY

1. Receive hits from INDEX
2. Ask ARCHIVE for segments
3. Extract events, check candidates
4. Send results to CLIENT



Outline

1. Introduction: VAST

2. Architecture
    - Overview
    - Data Model
    - Implementation
    - Query

3. Using VAST

4. Demo

Getting Up and Running

Requirements

- C++14 compiler
  - Clang 3.4 (easiest bootstrapped with Robin's `install-clang`)
  - GCC 4.9 (not yet fully supported)
- CMake
- Boost Libraries (headers only)
- C++ Actor Framework (`unstable` branch currently)

Installation

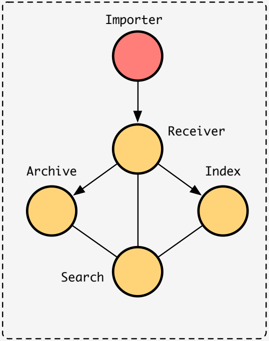
- `git clone git@github.com:mavam/vast.git && cd vast`
- `./configure && make && make test && make install`
- `vast -h # brief help`
- `vast -z # complete options`

Deployment

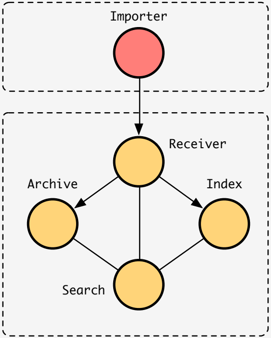
Network Transparency

- Actors can live in the same address space
  → Efficiently pass messages as pointer
- Actors can live on different machines
  → Transparent serialization of messages

One-Shot Import



Import with 2 Processes



Importing Logs

One-Shot Import

```
vast -C -I -r conn.log
zcat *.log.gz | vast -C -I
vast -C -I -p partition-2014-01 < conn.log
```

Import with 2 Processes

```
vast -C             # core
vast -I < conn.log  # importer
```

Synopsis: One-Shot Queries

JSON Query

```
vast -C                   # core
vast -E -o json -l 5 -q ':addr in 10.0.0.0/8'
```

Bro Query

```
vast -C                   # core
vast -E -o bro -l 5 -q ':addr in 10.0.0.0/8'
```

Outline

1. Introduction: VAST

2. Architecture
   - Overview
   - Data Model
   - Implementation
   - Query

3. Using VAST

4. Demo

Thank You... Questions?

```
 _   _____   __________
| | / / _ | / __/_  __/
| |/ / __ |_\ \  / /
|___/_/ |_/___/ /_/
```

https://github.com/mavam/vast

IRC at Freenode: #vast