Why Actors Rock: Designing a Distributed Database with `libcppa`

Matthias Vallentin
`matthias@bro.org`

University of California, Berkeley

C++Now
May 15, 2014

Outline

1. System Overview: VAST
2. Architecture: Ingestion, Indexing, and Query
   - Ingestion
   - Indexing
   - Query
3. Experience
4. Demo

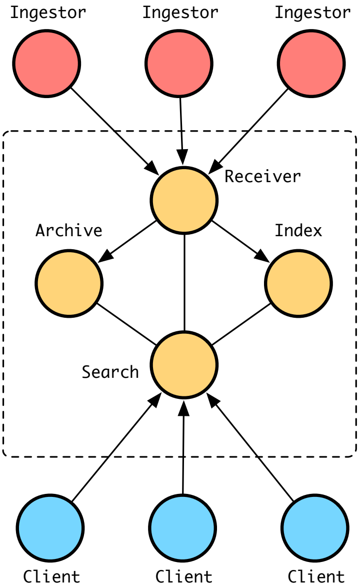
VAST: Visibility Across Space and Time

VAST

Distributed database built with `libcppa`

Goals

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



Example Use Case: Network Security Analysis

Network Forensics & Incident Response

- Scenario: security breach discovered
- Analysts tasked with determining scope and impact

Analyst questions

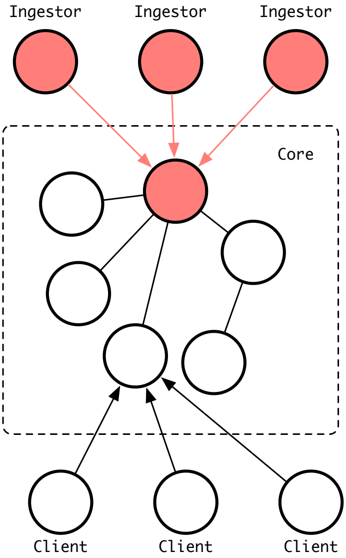
- How did the attacker(s) get in?
- How long did the they stay under the radar?
- What is the damage ($$$, reputation, data loss, etc.)?
- How to detect similar attacks in the future?

Outline

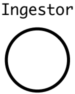
Outline

1. System Overview: VAST
2. Architecture: Ingestion, Indexing, and Query
   - Ingestion
   - Indexing
   - Query
3. Experience
4. Demo

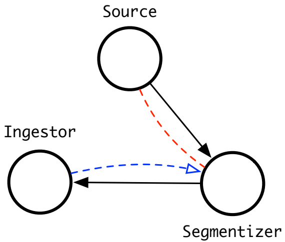
Ingestion



Ingestion



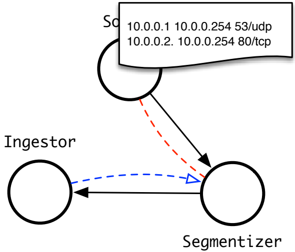
Ingestion



Ingestion

INGESTOR

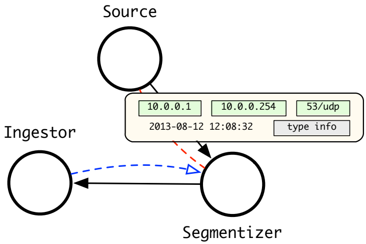
1. Parse input into events



Ingestion

INGESTOR

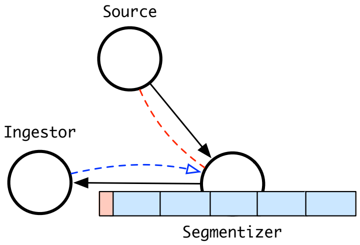
1. Parse input into events



Ingestion

INGESTOR

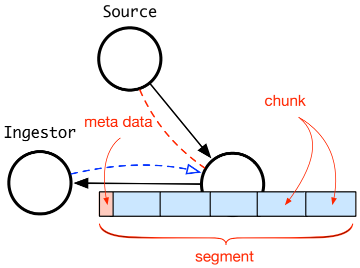
1. Parse input into events
2. Compress & chunk into segments



Ingestion

INGESTOR

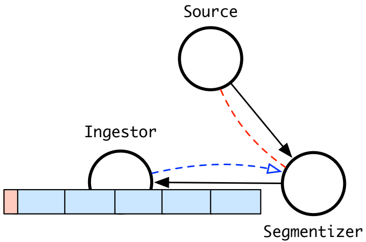
1. Parse input into events
2. Compress & chunk into segments



Ingestion

INGESTOR

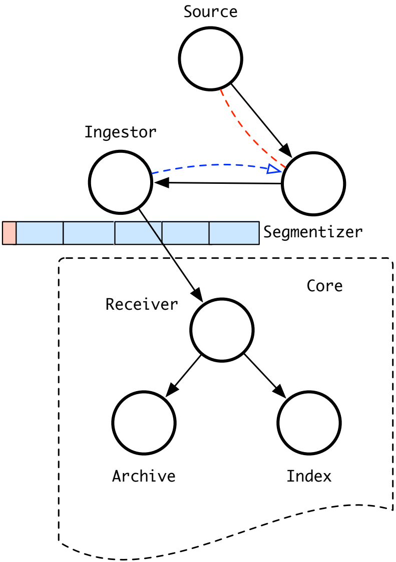
1. Parse input into events
2. Compress & chunk into segments



Ingestion

INGESTOR

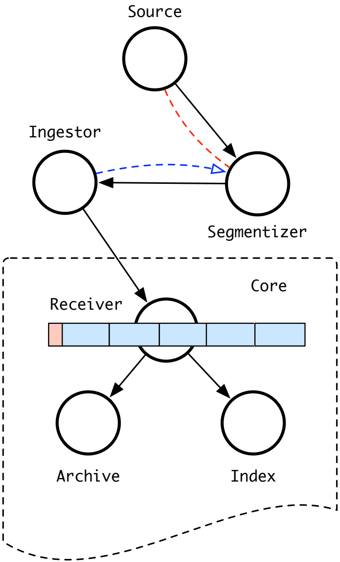
1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER



Ingestion

INGESTOR

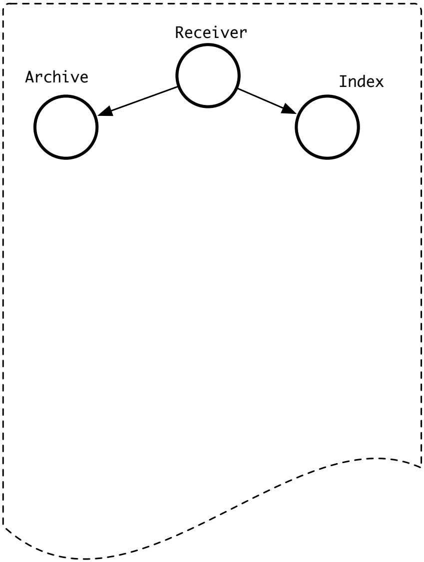
1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER



Ingestion

INGESTOR

1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER



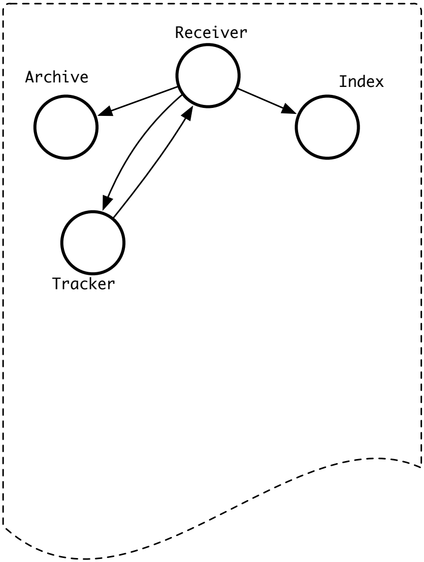
Ingestion

INGESTOR

1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER

RECEIVER

1. Accept and ACK segment
2. Assign segment an ID range from space 2^{64}



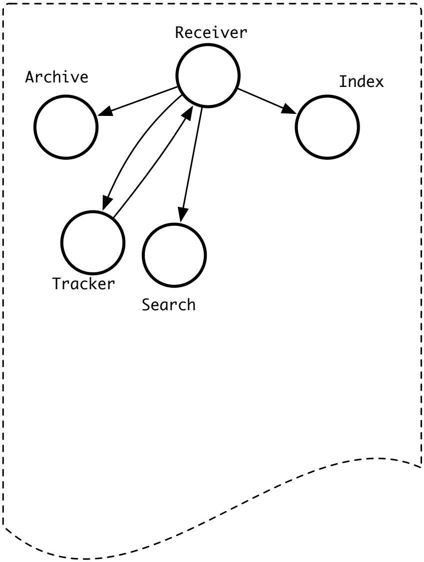
Ingestion

INGESTOR

1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER

RECEIVER

1. Accept and ACK segment
2. Assign segment an ID range from space 2^{64}
3. Record segment schema



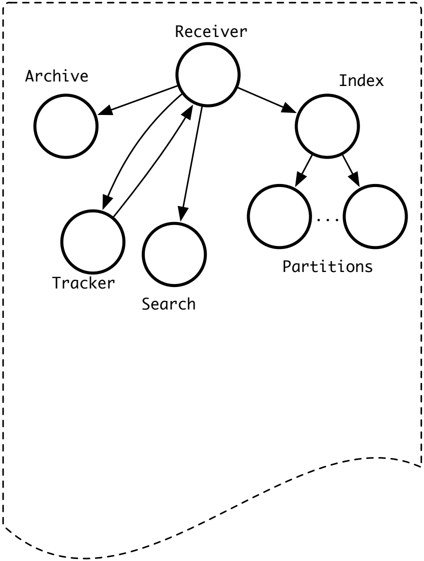
Ingestion

INGESTOR

1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER

RECEIVER

1. Accept and ACK segment
2. Assign segment an ID range from space 2^{64}
3. Record segment schema



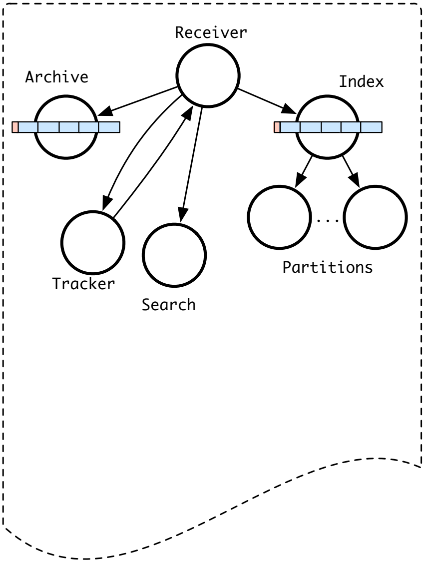
Ingestion

INGESTOR

1. Parse input into events
2. Compress & chunk into segments
3. Send segments to RECEIVER

RECEIVER

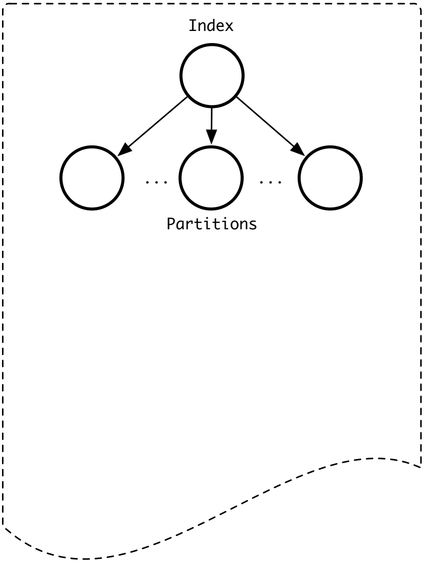
1. Accept and ACK segment
2. Assign segment an ID range from space 2^{64}
3. Record segment schema
4. Forward segment to ARCHIVE and INDEX



Outline

1. System Overview: VAST
2. Architecture: Ingestion, Indexing, and Query
   - Ingestion
   - Indexing
   - Query
3. Experience
4. Demo

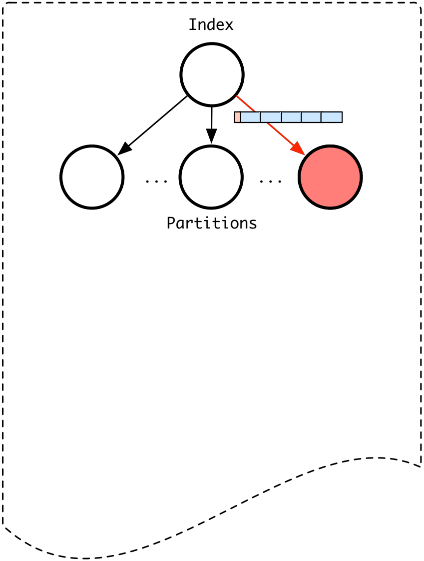
Indexing



Indexing

INDEX

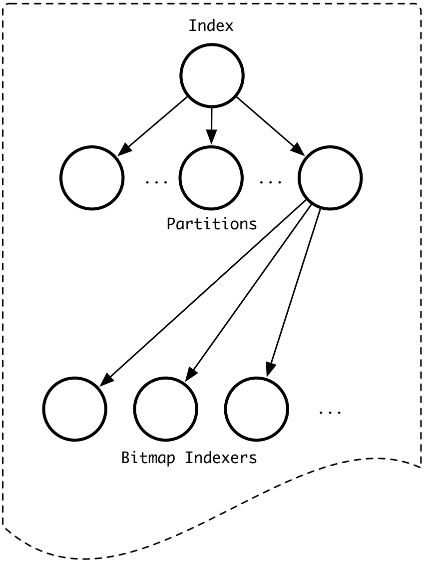
1. Forward segment to relevant partition



Indexing

INDEX

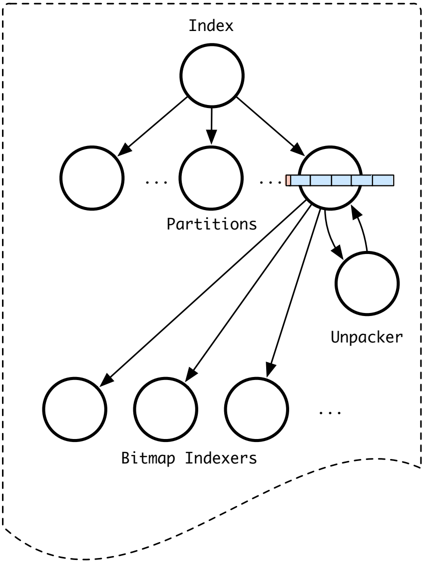
1. Forward segment to relevant partition
2. Spawn INDEXER for event values



Indexing

INDEX

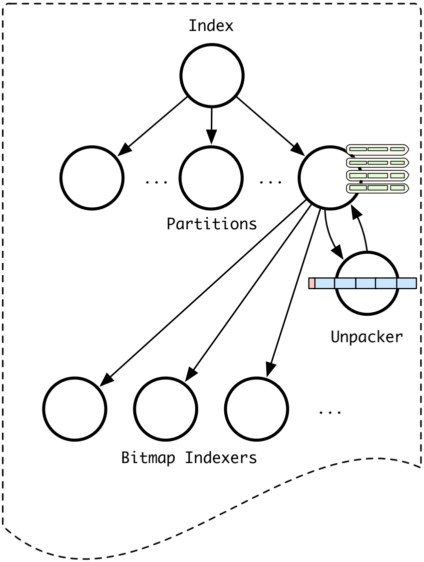
1. Forward segment to relevant partition
2. Spawn INDEXER for event values



Indexing

INDEX

1. Forward segment to relevant partition
2. Spawn INDEXER for event values
3. Unpack segment back into events



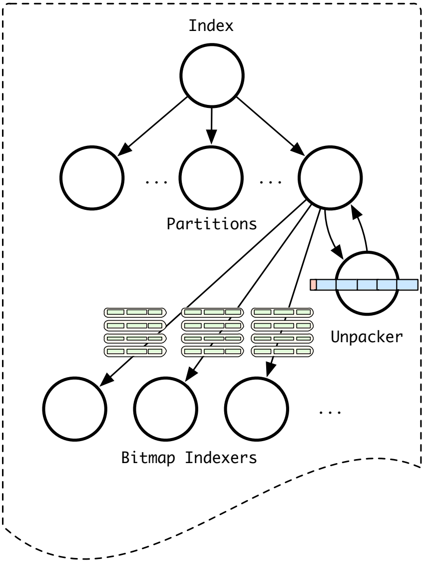
Indexing

INDEX

1. Forward segment to relevant partition
2. Spawn INDEXER for event values
3. Unpack segment back into events

INDEXER

1. Receive event



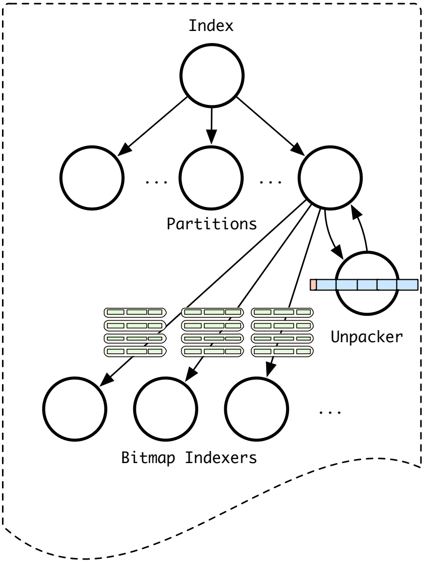
Indexing

INDEX

1. Forward segment to relevant partition
2. Spawn INDEXER for event values
3. Unpack segment back into events

INDEXER

1. Receive event
2. Select value to index



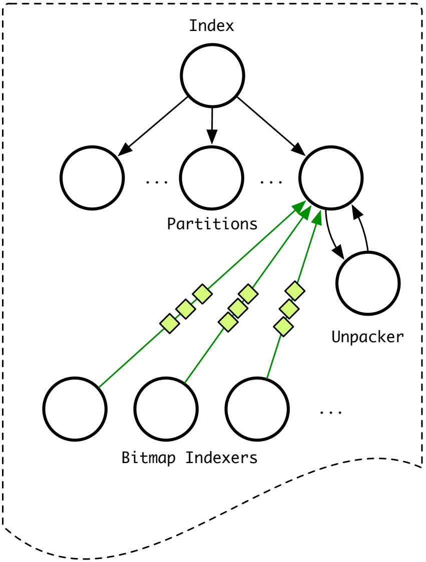
Indexing

INDEX

1. Forward segment to relevant partition
2. Spawn INDEXER for event values
3. Unpack segment back into events

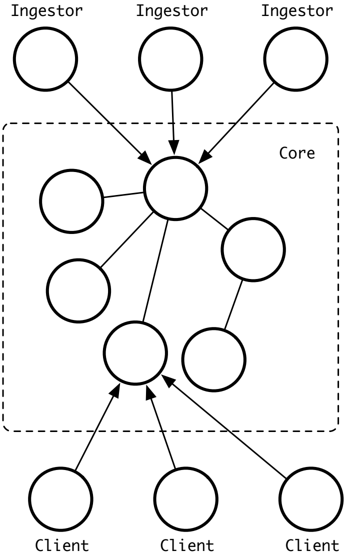
INDEXER

1. Receive event
2. Select value to index
3. Report statistics back to partition

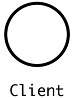


Outline

Query



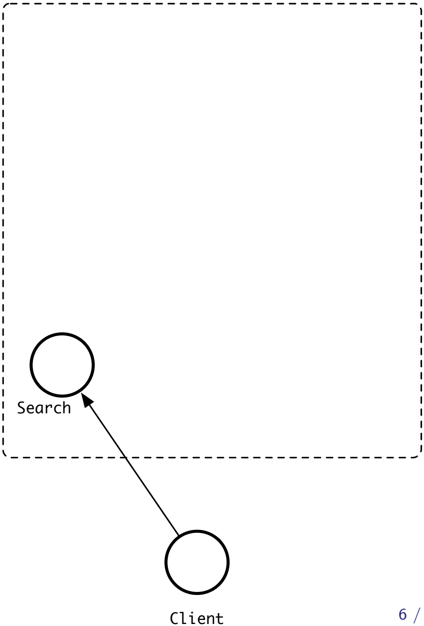
Query



Query

CLIENT

1. Send query string to SEARCH



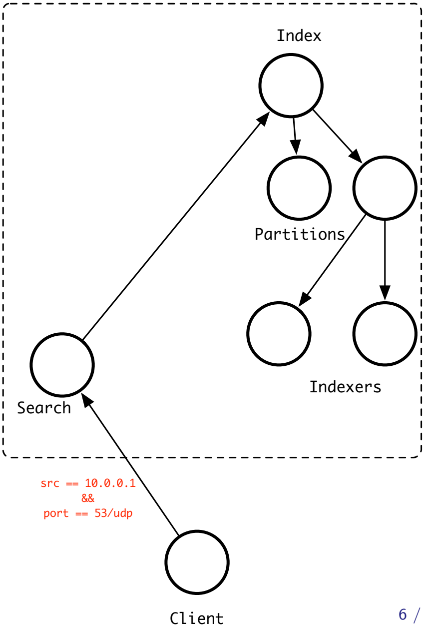
Query

CLIENT

1. Send query string to SEARCH

SEARCH

1. Parse and validate query string



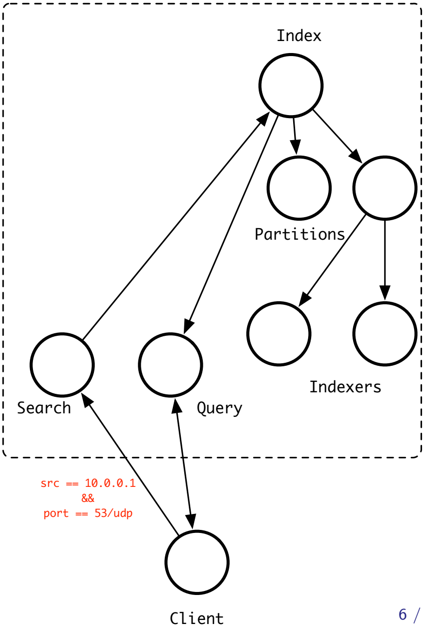
Query

CLIENT

1. Send query string to SEARCH

SEARCH

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



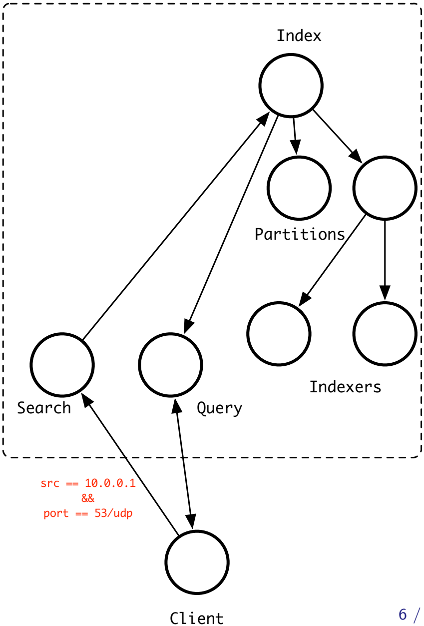
Query

CLIENT

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

SEARCH

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



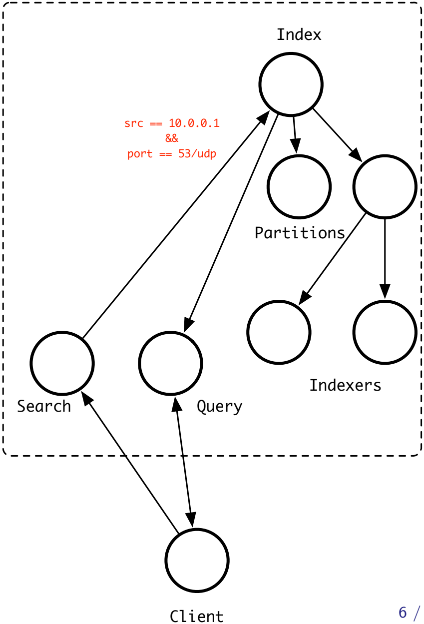
Query

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



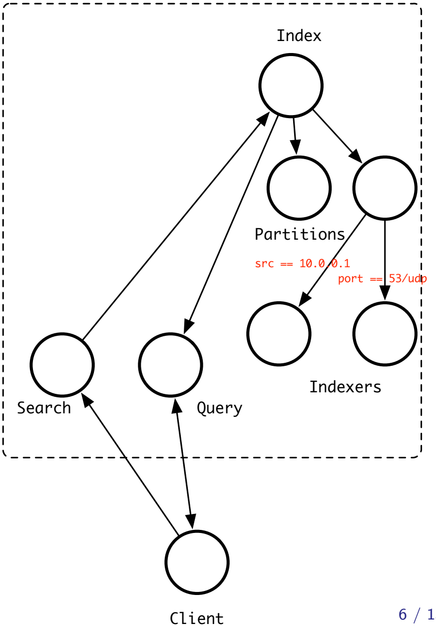
Query

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

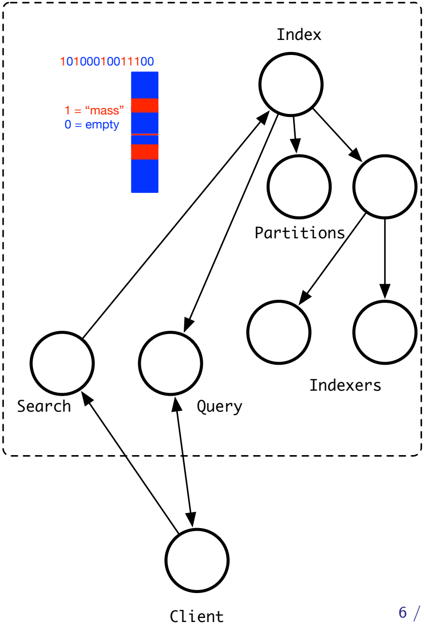
CLIENT

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



Query

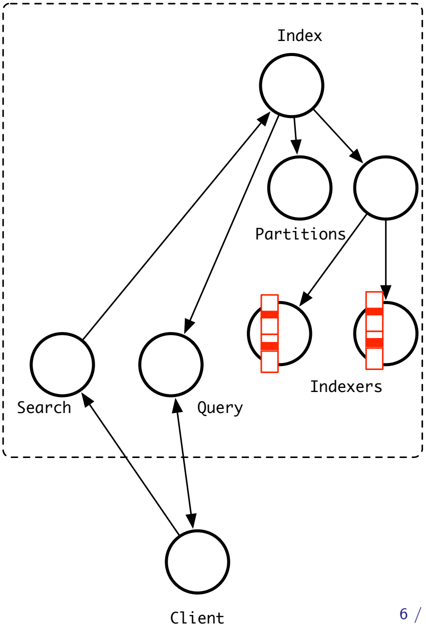
CLIENT

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



Query

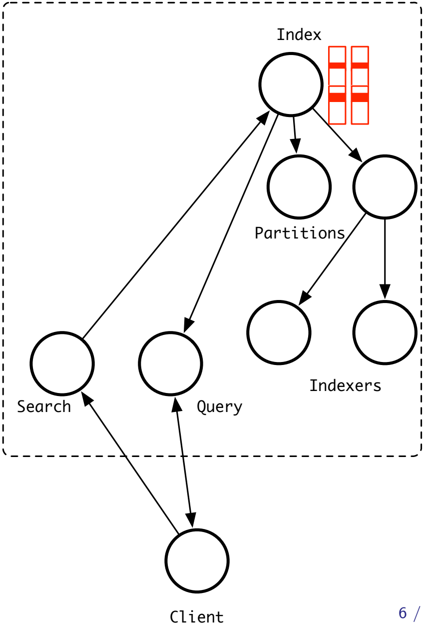
CLIENT

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



Query

CLIENT

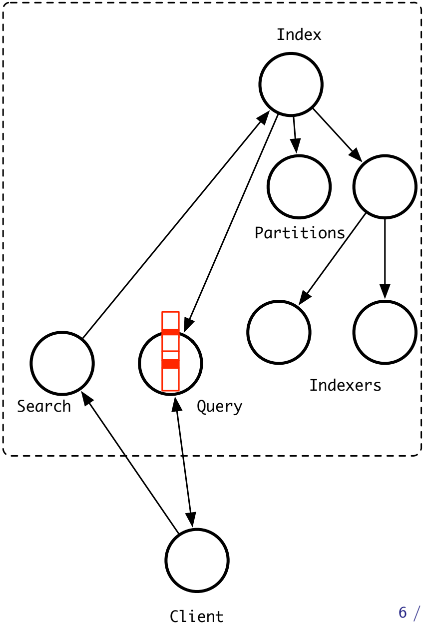
1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

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

QUERY

1. Receive hits from INDEX



Query

CLIENT

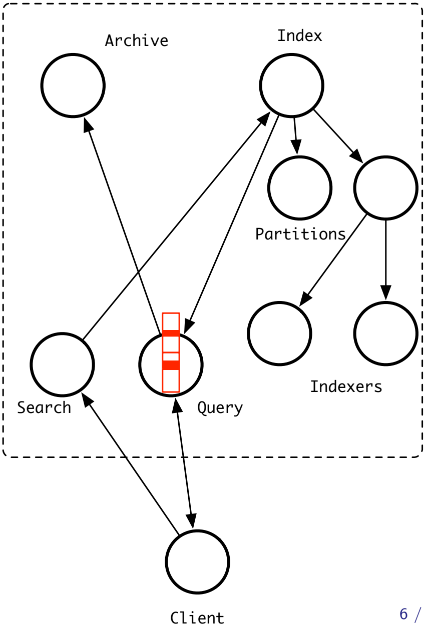
1. Send query string to SEARCH
2. Receive QUERY actor

SEARCH

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

QUERY

1. Receive hits from INDEX



Query

CLIENT

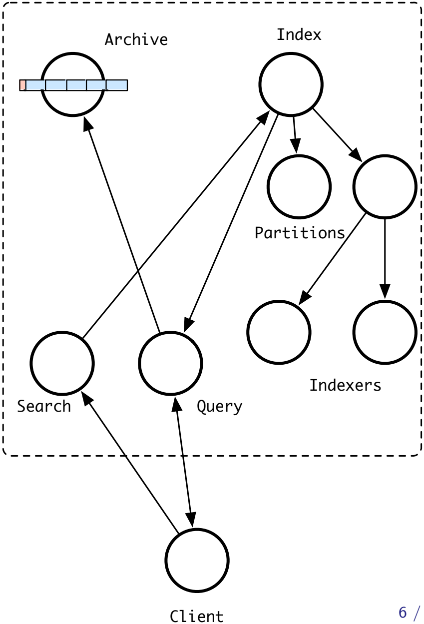
1. Send query string to SEARCH
2. Receive QUERY actor

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

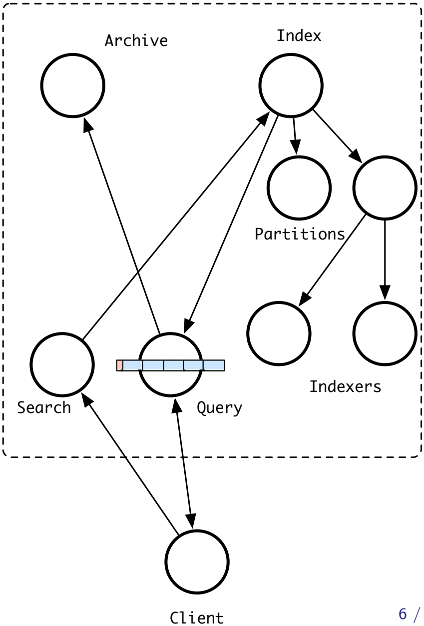
1. Send query string to SEARCH
2. Receive QUERY actor

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

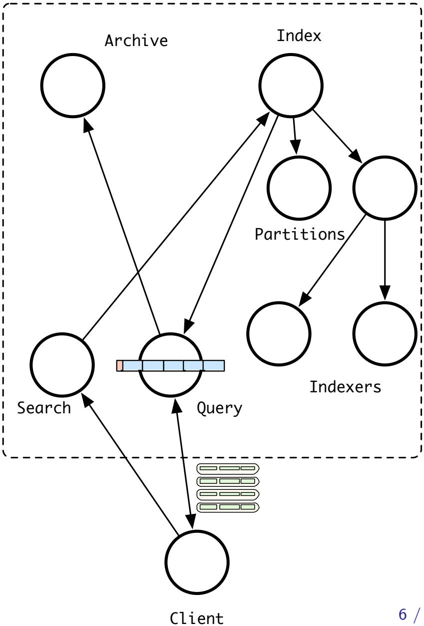
1. Send query string to SEARCH
2. Receive QUERY actor
3. Extract results from QUERY

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

Issue #1: Bufferbloat

Bufferbloat

Large buffers cause **high latency** and **jitter**

Aside: Go

goroutines execute concurrently and exchange messages via *channels*

- Sender blocks when channel is full
- Receiver blocks when channel is empty

→ Explicit notion of buffer

- `libcppa` : no blocking to signal overload

Bufferbloat in VAST

- Large segments (128MB)
- Data flow rates
  - Ingestion: 80k–100k events/sec
  - Indexing: 20k–200k events/sec

→ Sender overloads receiver: system runs out-of-memory

Solution #1: Flow Control

Flow Control

Feedback on capacity from overloaded resource up to sender

Revised indexing process

1. PARTITION spawns indexers and dispatches events
   - Queue length: number of events sent to INDEXER
2. Indexers report back how many events they have indexed
   - Decreases queue length by events processed
3. Receiver polls index every 100ms for maximum queue length
   - If watermark reached, tell INGESTORS to stop
   - If watermark cleared, tell INGESTORS to go

Problem #2: Data Structure Inflation

Initial indexing process

1. Unpack segment
2. Create one `vector<event>` for meta indexes (across events)
3. Create one `vector<event>` for data indexes (per event)
4. Forward to events to the corresponding indexers

Issues

1. Memory overhead from maintaining multiple different data slices
2. Effect exacerbated by buffer bloat

Solution #2: Data Sharing

Intra-Process Performance

Share data intelligently instead of partitioning it beforehand

Revised indexing process

- Do not “inflate” data just to partition it for workers
- GPGPU-style: make data available “globally” in workers
  - Disburdens CPU: no time needed to transform data
  - Reduces memory footprint: data exists exactly once

Problem #3: Messaging Complexity

Complex Query processing

A QUERY actor receives messages from ARCHIVE, INDEX, and CLIENT

- QUERY acts as "iterator" over the archive for index hit
- Maintains lots of state for incremental extraction of matches
- Difficult to implement correctly when messages arrive in any order
- Many `if-then-else` constructs clutter main logic

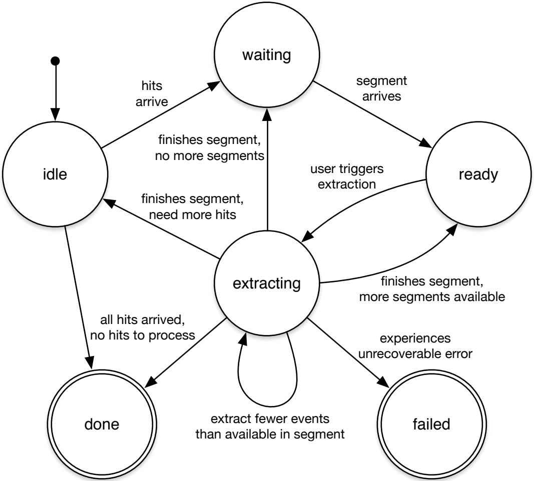
Solution #3: State Machine

Finite State Machine

Implement stateful logic with a finite state machine

Revised query process

- Each state defines a set of valid messages
- Explicit transitions make readable and clear code
- `libcppa` primitive: `become/unbecome`



Summary & Lessons Learned

Lesson #1

Programming distributed systems feels like “networking”

- Flow control prevents imbalanced sender/receiver speeds
- Bufferbloat increases latency and causes processing spikes
- Explicit state machines keep asynchronous messaging manageable

Lesson #2

GPGPU programming style fits well for intra-process concurrency

- Make *full* data available to *all* workers
- Each worker is responsible for extracting its relevant data

Outline

1. System Overview: VAST
2. Architecture: Ingestion, Indexing, and Query
   - Ingestion
   - Indexing
   - Query
3. Experience
4. Demo

Thank You... Questions?

FIN

`https://github.com/mavam/vast`
`https://github.com/Neverlord/libcppa`

IRC at Freenode: `#vast`, `#libcppa`