Zenoss Core 4 Event Architecture

Jane Curry

jane.curry@skills-1st.co.uk

based on the paper at

http://community.zenoss.org/docs/DOC-13746 or

http://www.skills-1st.co.uk/papers/jane/zenoss4-events/

Agenda

- Quick look at Zenoss 3 architecture
- Zenoss 4 architecture
  - subsystems
  - daemons
  - databases
- 10 minute comfort break (that's 600 seconds!)
- Event life cycle
  - Event generation
  - Device context
  - Event class mapping / event context / event transforms
  - Database insertion & duplicates
  - Resolution & Ageing

In the beginning...

- Events received by various daemons
- Events **processed** by zenhub daemon
- Event life cycle
  - Event generation -- Device context
  - Event class mapping -- Event context
  - Event transforms -- Database insertion & deduplication
  - Resolution -- Ageing
- MySQL events database
  - **summary** table for active events -- **history** table for closed events
- Database fields = Event console fields
  = Event mapping / transform fields
- Table definitions in
  $ZENHOME/Products/ZenEvents/db/zenevents.sql

In the beginning...

- Events subsystem was a severe bottleneck
- zenhub had lots of other responsibilities
- Single pipeline for processing events
- Almost no indexing on MySQL database
- Event Console updates slow
- Event deletions very slow
- Responsiveness of emails / pages / automation commands slow
- Debugging hard (zenhub.log & event.log)
-
- Fairly easy to understand

Skills 1st

jane@bino:~ - Shell - Konsole <2>

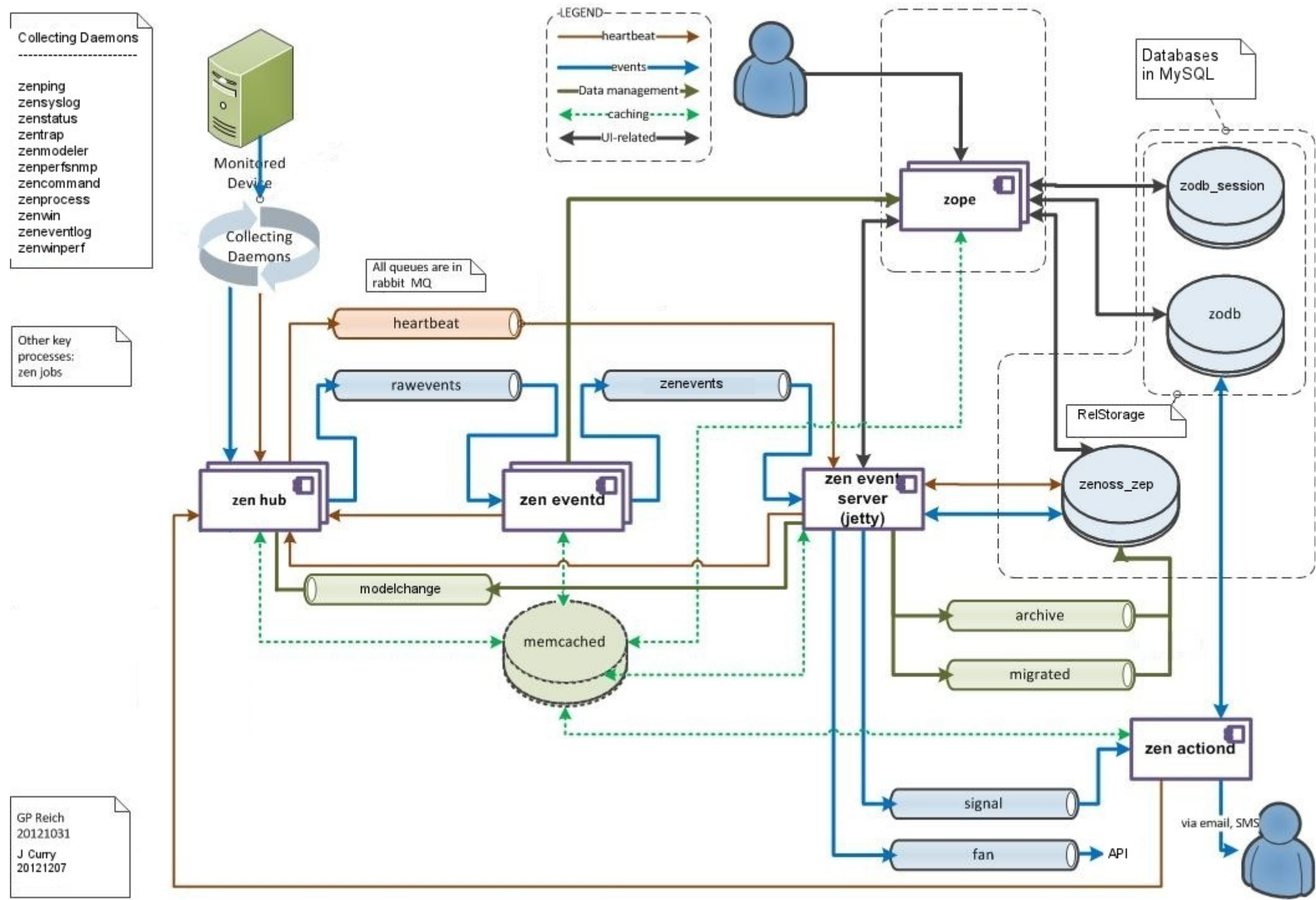
Session Edit View Bookmarks Settings Help

```
CREATE TABLE IF NOT EXISTS status
(
    dedupid          varchar(255) not null,
    evid             char(25) not null,
    device           varchar(128) not null,
    component        varchar(128) default "",
    eventClass       varchar(128) default "/Unknown",
    eventKey         varchar(128) default "",
    summary          varchar(128) not null,
    message          varchar(4096) default "",
    severity         smallint default -1,
    eventState       smallint default 0,
    eventClassKey    varchar(128) default "",
    eventGroup       varchar(64) default "",
    stateChange      timestamp,
    firstTime        double,
    lastTime         double,
    count            int default 1,
    prodState        smallint default 0,
    suppid           char(36) not null,
    manager          varchar(128) not null,
    agent            varchar(64) not null,
    DeviceClass      varchar(128) default "",
    Location         varchar(128) default "",
    Systems          varchar(255) default "",
    DeviceGroups     varchar(255) default "",
    ipAddress        char(15) default "",
    facility         varchar(8) default "unknown",
    priority         smallint default -1,
    ntevid           smallint unsigned default 0,
    ownerid          varchar(32) default "",
    clearid          char(25),
    DevicePriority   smallint(6) default 3,
    eventClassMapping varchar(128) default "",
    monitor          varchar(128) default "",
    PRIMARY KEY ( dedupid ),
    Index evididx (evid),
    Index clearidx (clearid),
    Index severityidx (severity),
    Index deviceidx (device)
) ENGINE=INNODB;
"zenevents.sql" [readonly] 163 lines --0%--                    1,1           Top
```

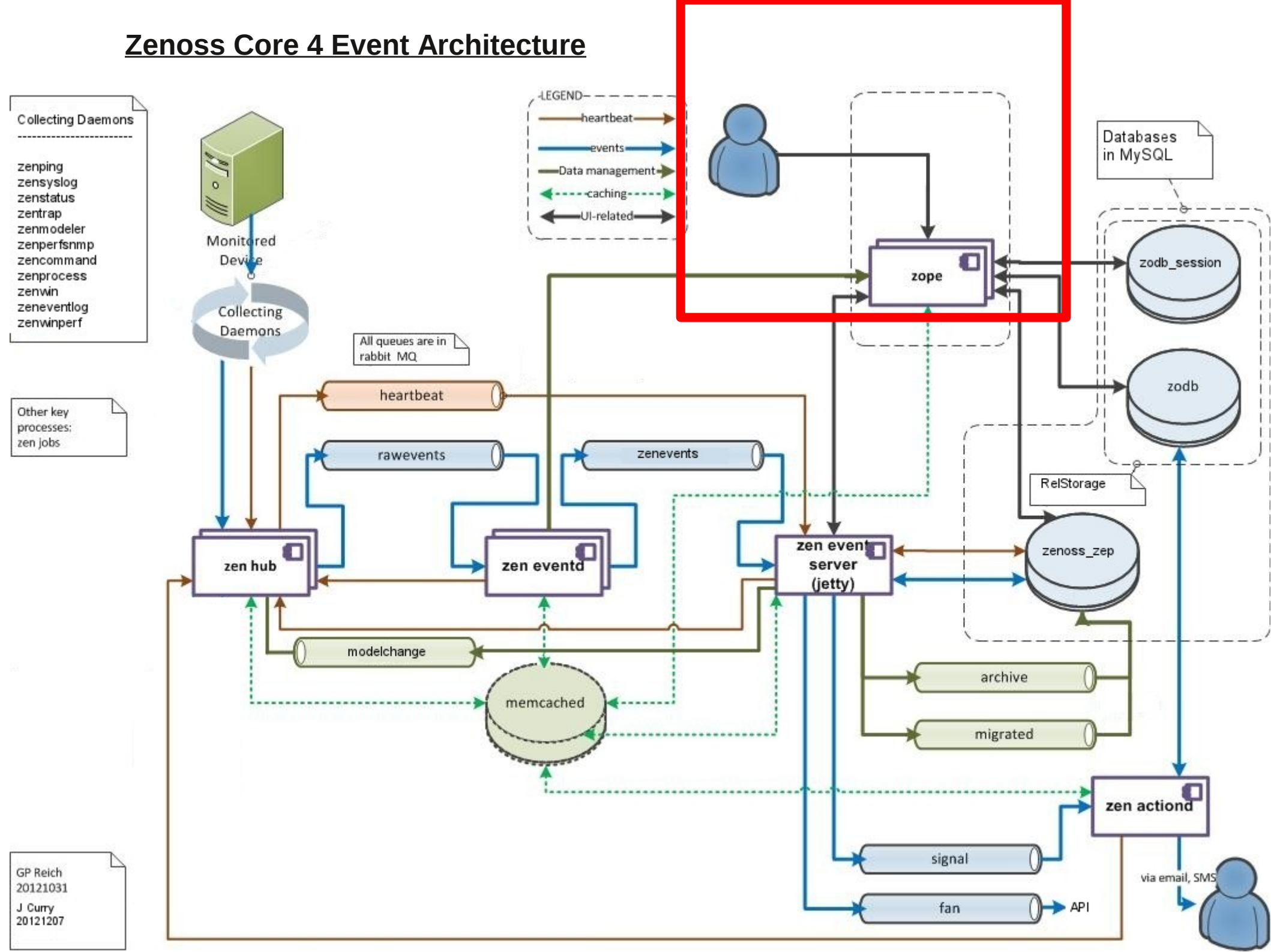
Zenoss 3 MySQL status table fields in events database

Shell

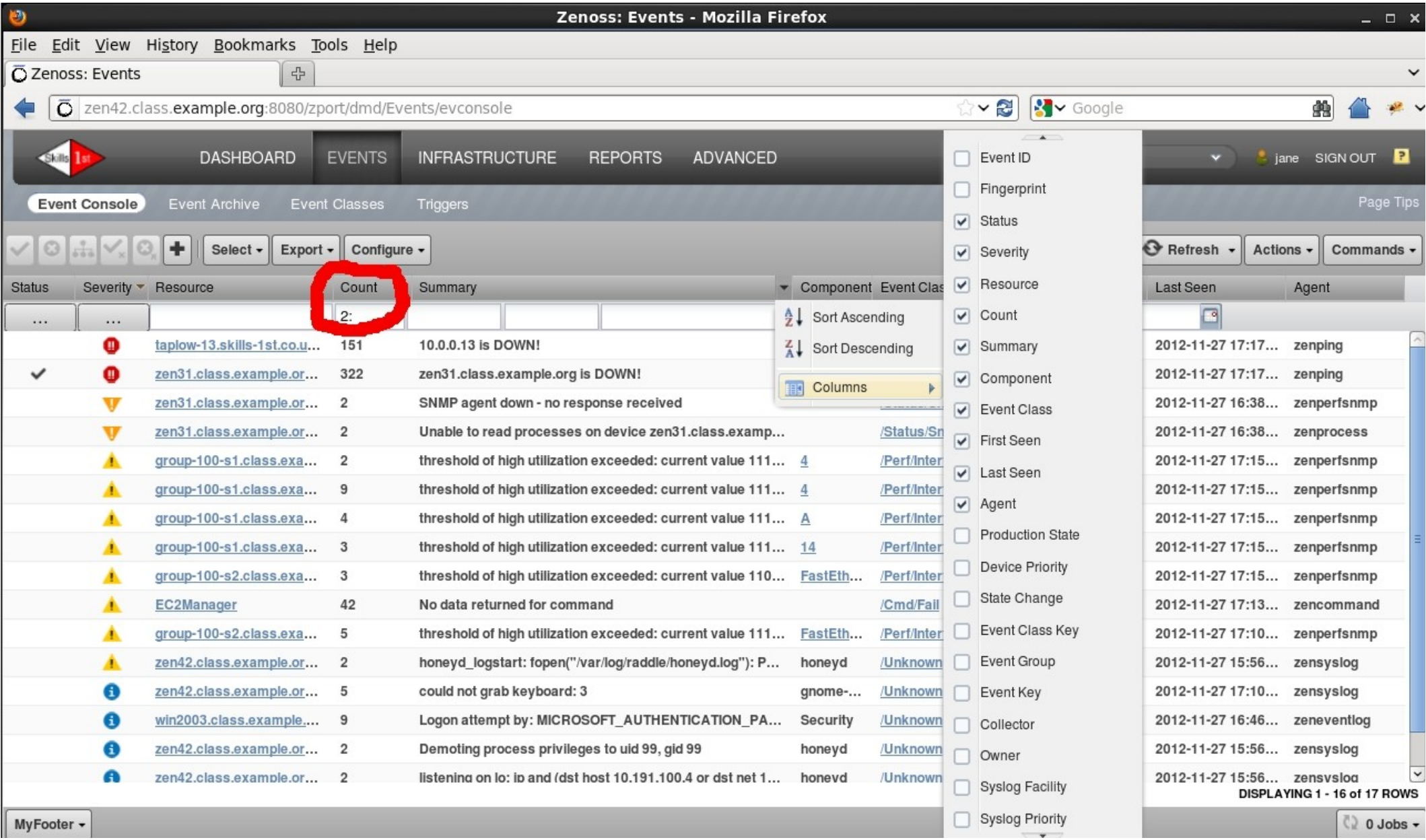
Zenoss Core 4 Event Architecture



Zenoss Core 4 Event Architecture



Event Console



Event Console

- **Severity**
  - Critical
  - Error
  - Warning
  - Info
  - Debug
  - Clear
- Resource / Component
- Event Class
- Summary

- **Status**
  - New (0)
  - Acknowledged (1)
  - Suppressed (2)
  - Closed (3)
  - Cleared (4) †
  - Dropped (5) †
  - Aged (6) †
- First Seen / Last Seen
- Agent

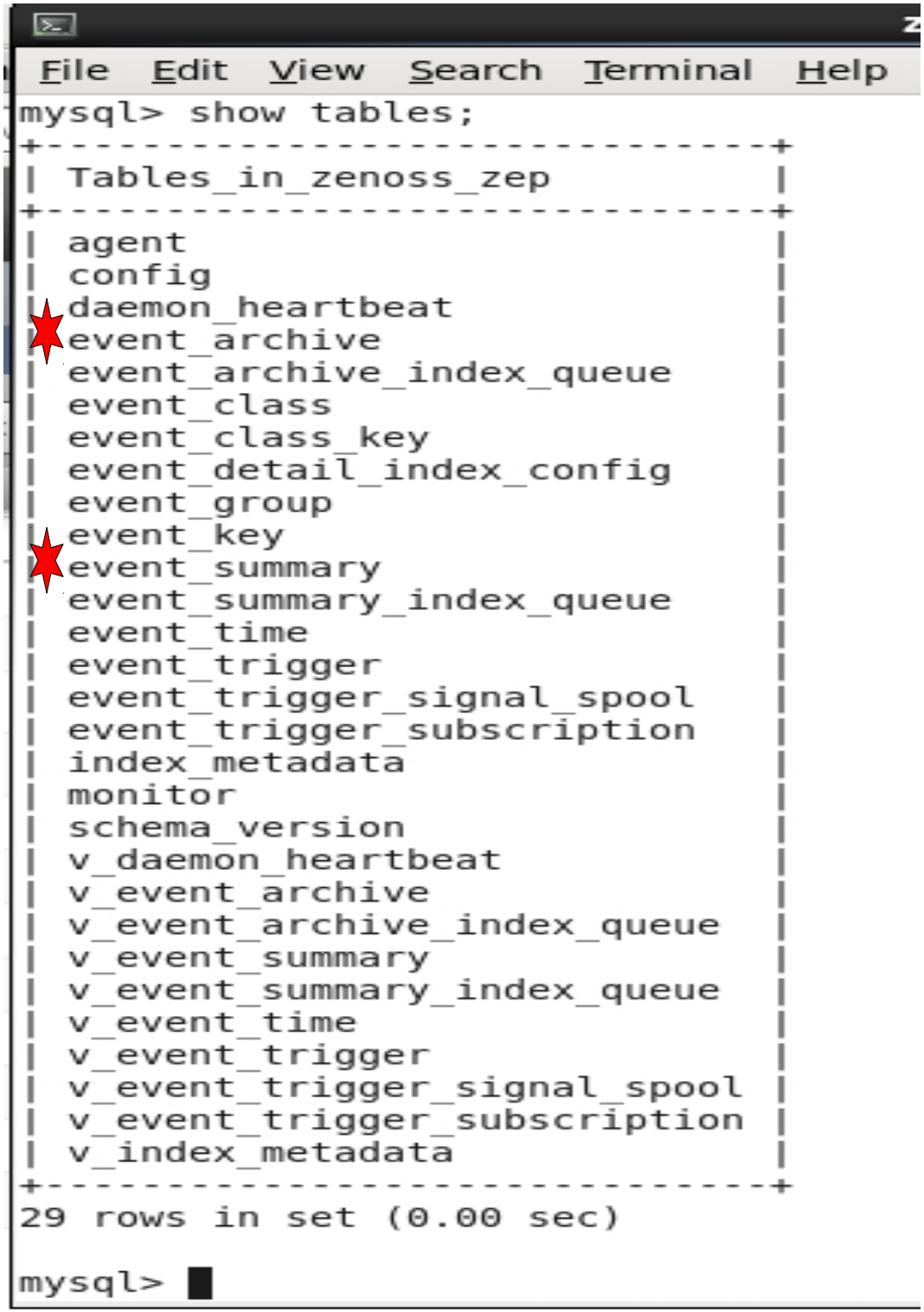
† new in Zenoss 4

Skills 1st

MySQL **zenoss_zep** database

- zenoss_zep new with Zenoss 4
- event_summary table
  - may contain Closed, Cleared and Aged events
- event_archive table
  - only has Closed, Cleared and Aged events
- attributes in tables do **not** match event console

Console	**Database**
device	element_identifier
component	element_sub_identifier
eventState	status_id
firstSeen	first_seen



Tables in zenoss_zep

* Lots more tables than Zenoss 3

- agent
- event_class
- event_class_key
- event_group
- evet_key
- event_trigger
- monitor

zenoss@zen42:/opt/zenoss/local/json_api_python/4.2

File Edit View Search Terminal Help

```
mysql> describe event_summary;
+--------------------------+--------------+------+-----+---------+-------+
| Field                    | Type         | Null | Key | Default | Extra |
+--------------------------+--------------+------+-----+---------+-------+
| uuid                     | binary(16)   | NO   | PRI | NULL    |       |
| fingerprint_hash         | binary(20)   | NO   | UNI | NULL    |       |
| fingerprint              | varchar(255) | NO   |     | NULL    |       |
| status_id                | tinyint(4)   | NO   | MUL | NULL    |       |
| event_group_id           | int(11)      | YES  |     | NULL    |       |
| event_class_id           | int(11)      | NO   |     | NULL    |       |
| event_class_key_id       | int(11)      | YES  |     | NULL    |       |
| event_class_mapping_uuid | binary(16)   | YES  |     | NULL    |       |
| event_key_id             | int(11)      | YES  |     | NULL    |       |
| severity_id              | tinyint(4)   | NO   | MUL | NULL    |       |
| element_uuid             | binary(16)   | YES  | MUL | NULL    |       |
| element_type_id          | tinyint(4)   | YES  |     | NULL    |       |
| element_identifier       | varchar(255) | NO   |     | NULL    |       |
| element_title            | varchar(255) | YES  |     | NULL    |       |
| element_sub_uuid         | binary(16)   | YES  | MUL | NULL    |       |
| element_sub_type_id      | tinyint(4)   | YES  |     | NULL    |       |
| element_sub_identifier   | varchar(255) | YES  |     | NULL    |       |
| element_sub_title        | varchar(255) | YES  |     | NULL    |       |
| update_time              | bigint(20)   | NO   |     | NULL    |       |
| first_seen               | bigint(20)   | NO   |     | NULL    |       |
| status_change            | bigint(20)   | NO   |     | NULL    |       |
| last_seen                | bigint(20)   | NO   | MUL | NULL    |       |
| event_count              | int(11)      | NO   |     | NULL    |       |
| monitor_id               | int(11)      | YES  |     | NULL    |       |
| agent_id                 | int(11)      | YES  |     | NULL    |       |
| syslog_facility          | int(11)      | YES  |     | NULL    |       |
| syslog_priority          | tinyint(4)   | YES  |     | NULL    |       |
| nt_event_code            | int(11)      | YES  |     | NULL    |       |
| current_user_uuid        | binary(16)   | YES  |     | NULL    |       |
| current_user_name        | varchar(32)  | YES  |     | NULL    |       |
| clear_fingerprint_hash   | binary(20)   | YES  | MUL | NULL    |       |
| cleared_by_event_uuid    | binary(16)   | YES  |     | NULL    |       |
| summary                  | varchar(255) | NO   |     |         |       |
| message                  | varchar(4096)| NO   |     |         |       |
| details_json             | mediumtext   | YES  |     | NULL    |       |
| tags_json                | mediumtext   | YES  |     | NULL    |       |
| notes_json               | mediumtext   | YES  |     | NULL    |       |
| audit_json               | mediumtext   | YES  |     | NULL    |       |
+--------------------------+--------------+------+-----+---------+-------+
38 rows in set (0.01 sec)
```

Fields in event_summary

* Most of the field names changed

  - many refer to other tables eg
    agent_id, event_class_id, ...

  - evid -> uuid
  - dedupid -> fingerprint
  - eventState -> status_id
  - firstSeen -> first_seen
  - count -> event_count
  - agent -> agent_id
  - facility -> syslog_facility

  - device -> element_
  - component -> element_sub_

  - _json fields
    details
    notes

* **However**...
  - summary is still summary
  - message is still message

MySQL database.....

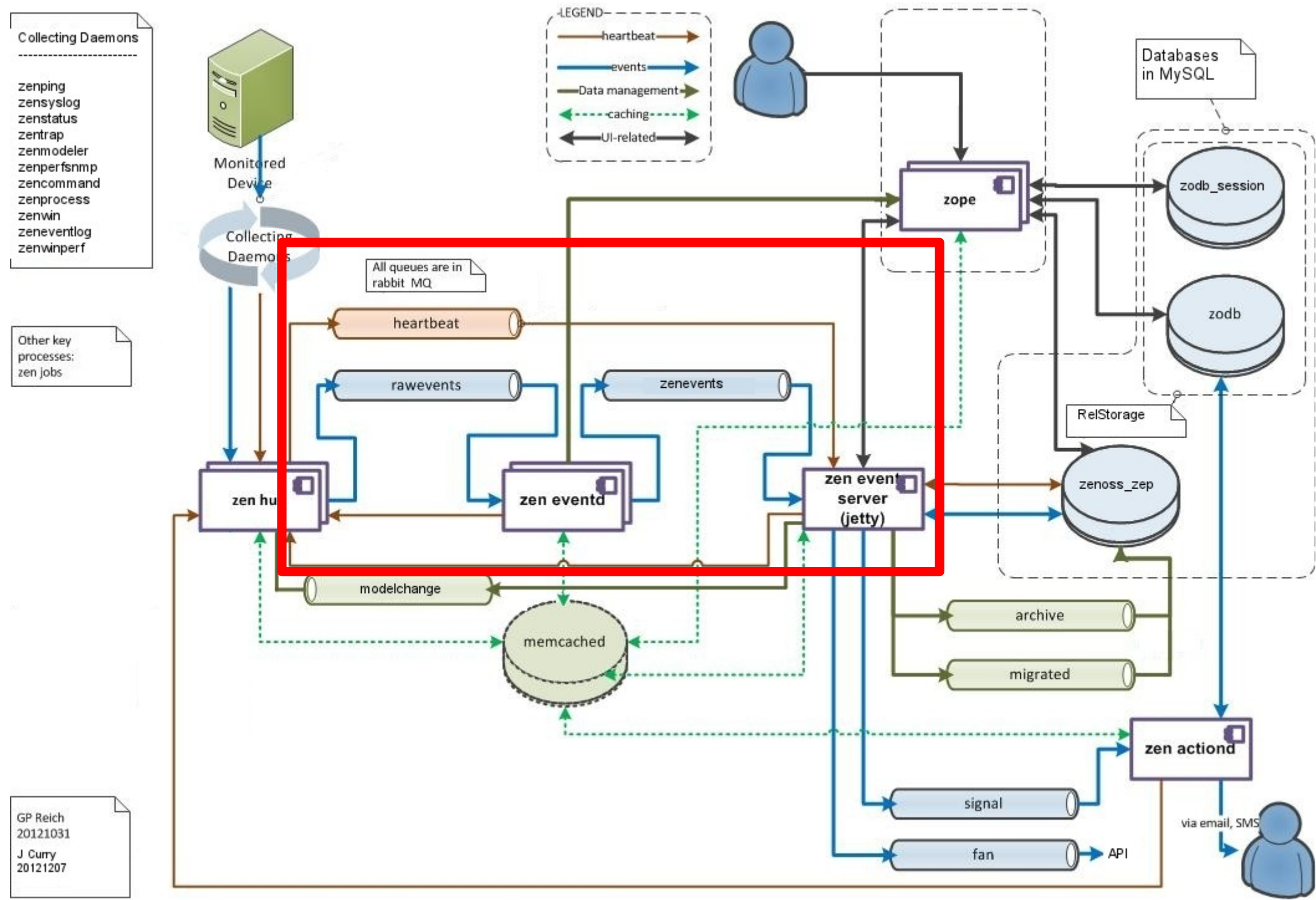
- Don't go there!

Here be dragons....

Event Proxy

- Writing rules and transforms for events operate on an **Event Proxy** data structure
  - manipulate **same** field names as in Zenoss 3.x
  - translation between database field names, message queue field names and Event Console field names
- See files under $ZENHOME/Products/ZenEvents/events2
- User-created event fields, including SNMP TRAP varbinds, are handled rather differently
  - event transforms may need changing

Zenoss Core 4 Event Architecture



New daemons

- zeneventserver
  - written in Java
- zeneventd
  - written in Python
- rabbitmq
  - open source Advanced Message Queueing Protocol (AMQP) package

zeneventserver daemon

- Also known as **zep**
- Consumes data from zenevents queue
- Stores processed events in MySQL database
- Presents data to GUI via Zope web application
  - uses JSON to present data to users
- Produces data for the **signal** queue for notification actions
- Manages ageing of data from event_summary to event_archive table in database
- May produce data for other queues

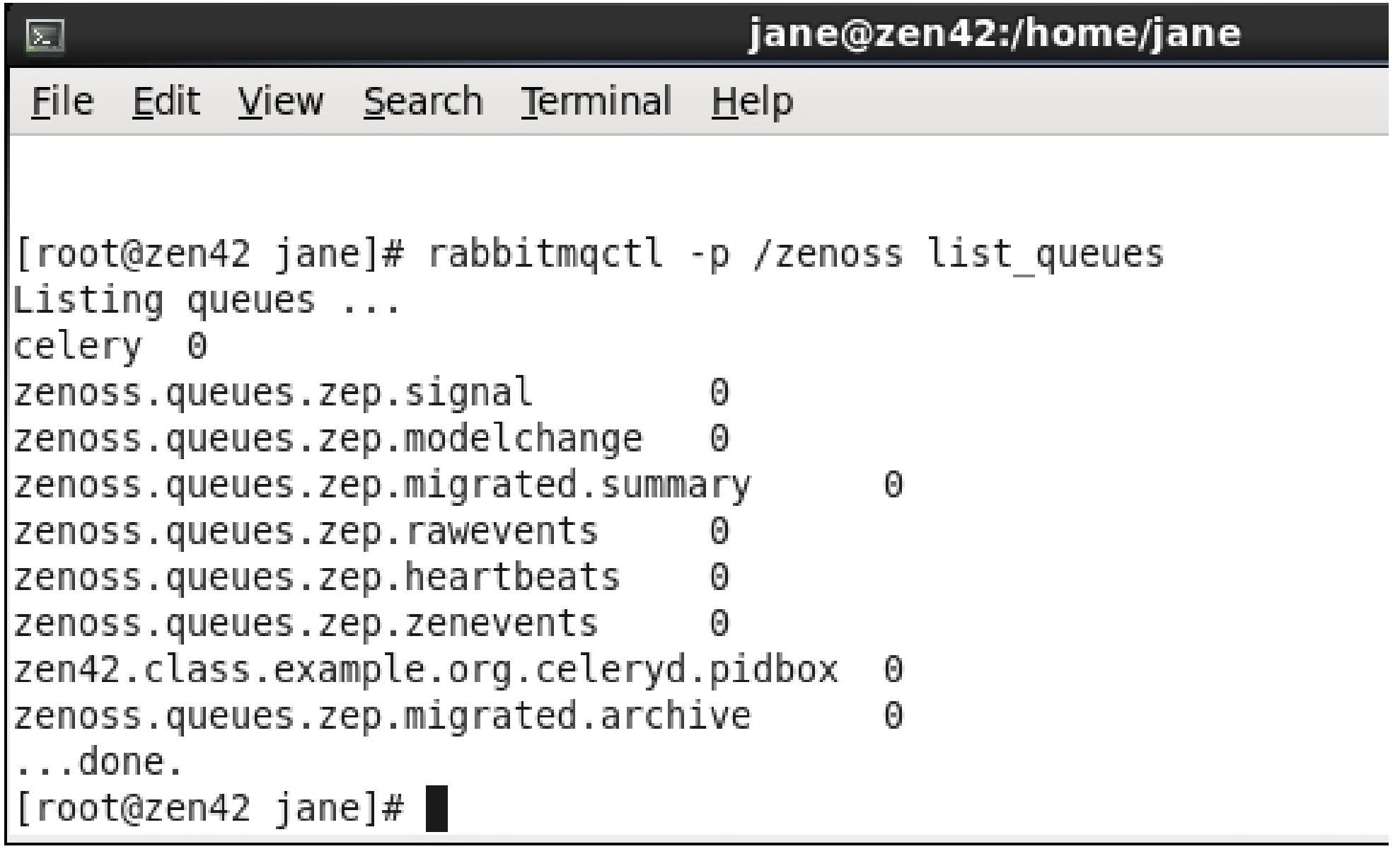
zeneventd daemon

- Responsible for most of the event processing
  - event classification
    - eventClassKey
    - Rule
    - Regex
  - device context
  - event context
  - transforms
  - deduplication fingerprint
- Consumes the **rawevents** queue
- Produces data for the **zenevents** queue

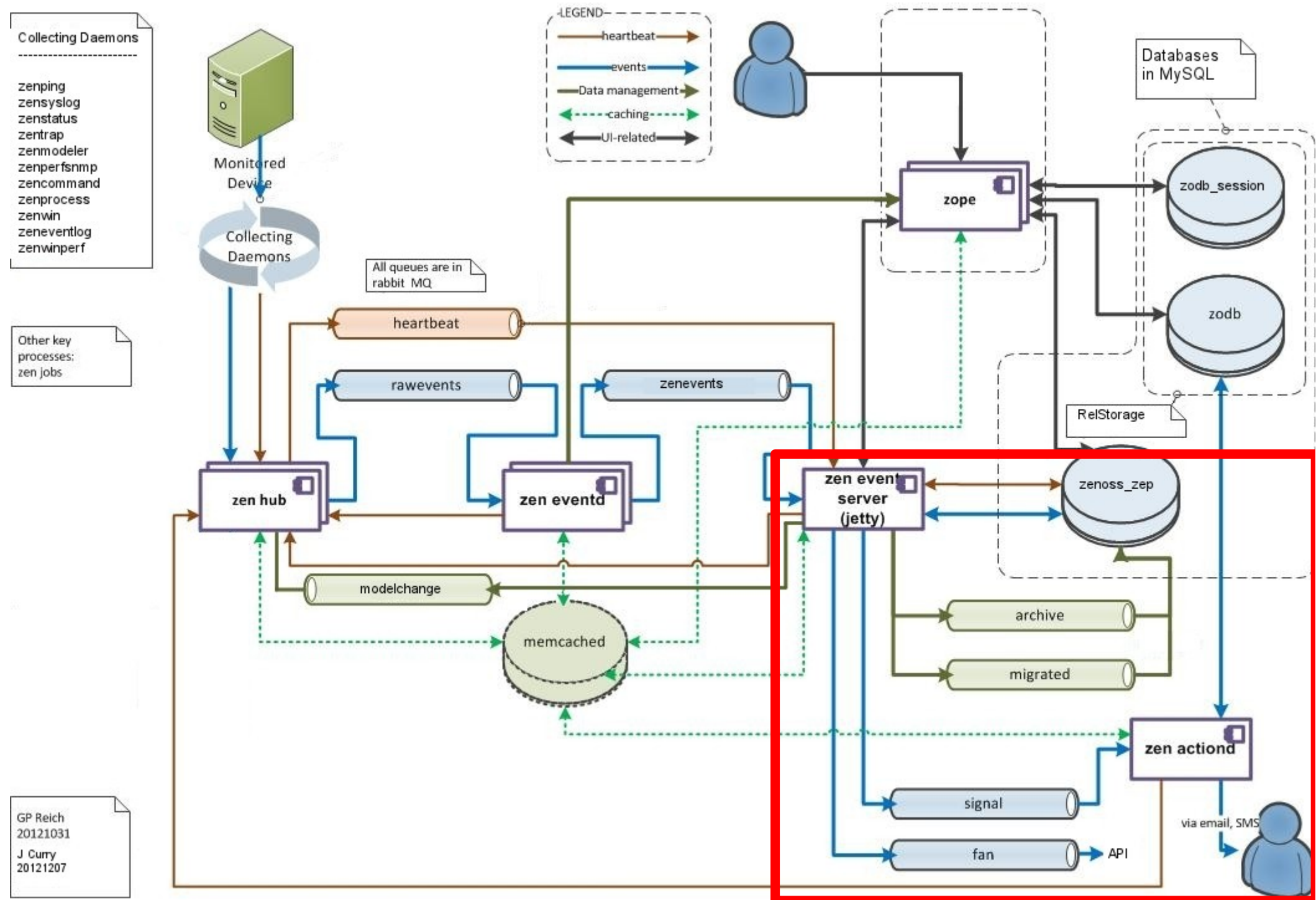
rabbitmq

- Crucial to Zenoss working - many daemons depend on it
- Configured when Zenoss installed
  - VHOST = “/zenoss”
  - USER = “zenoss”
  - PASSWORD = check $ZENHOME/etc/global.conf
- Started with *service zenoss start*
- If Zenoss server has name changed then rabbitmq needs reconfiguring
- rabbitmq commands need root privilege
- *rabbitmqctl report* - good to dump lots of data

rabbitmq queues



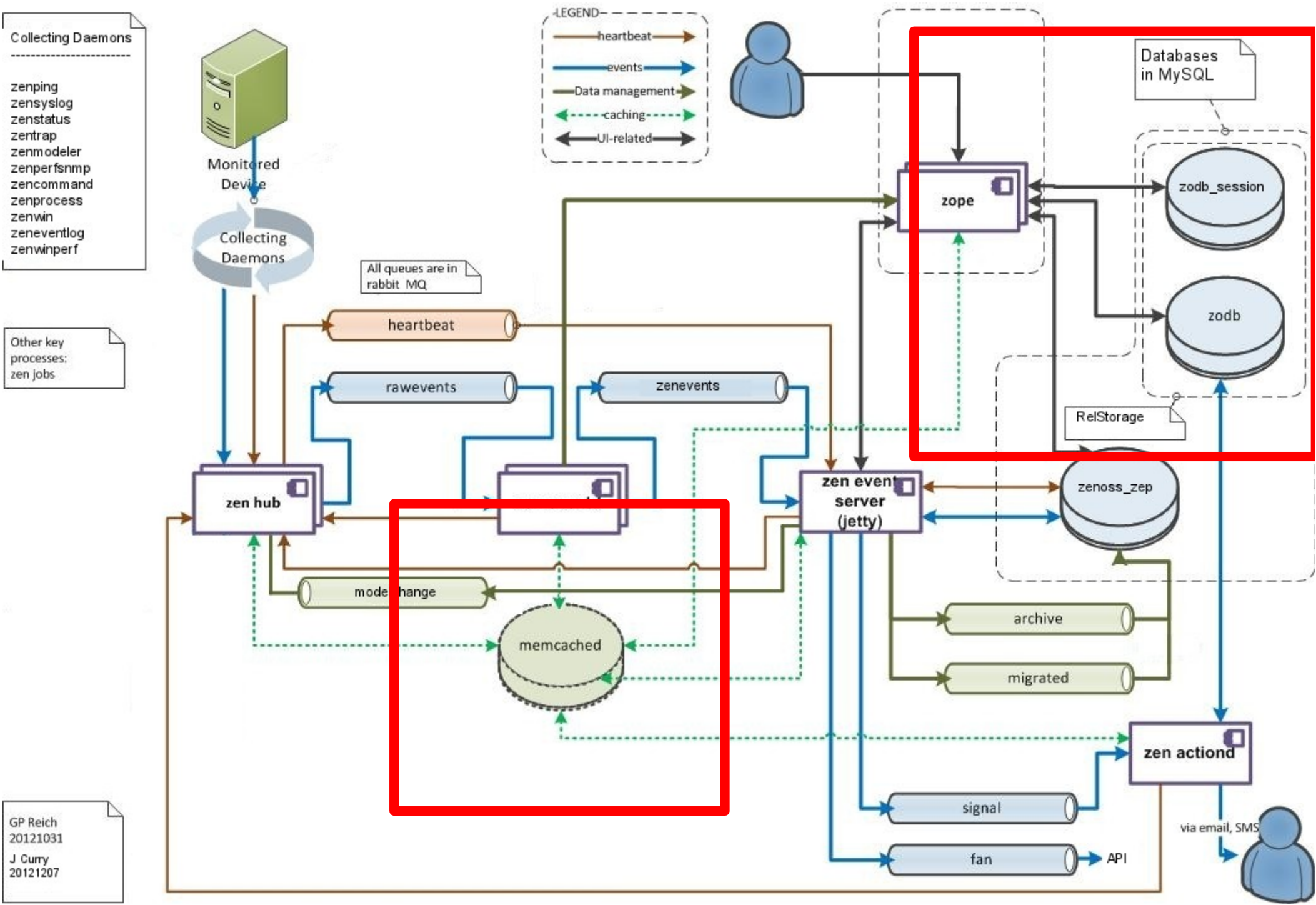
Zenoss Core 4 Event Architecture



zenactiond

- Responds to **Triggers** evaluated by zeneventserver
- Provides **Notifications**
  - email / page to users
  - command scripts / SNMP TRAPs for background automation
- Existed prior to Zenoss 4 with 60 sec cycle
- Completely rewritten for Zenoss 4
- Consumer of the signal queue
  - much more responsive

Zenoss Core 4 Event Architecture

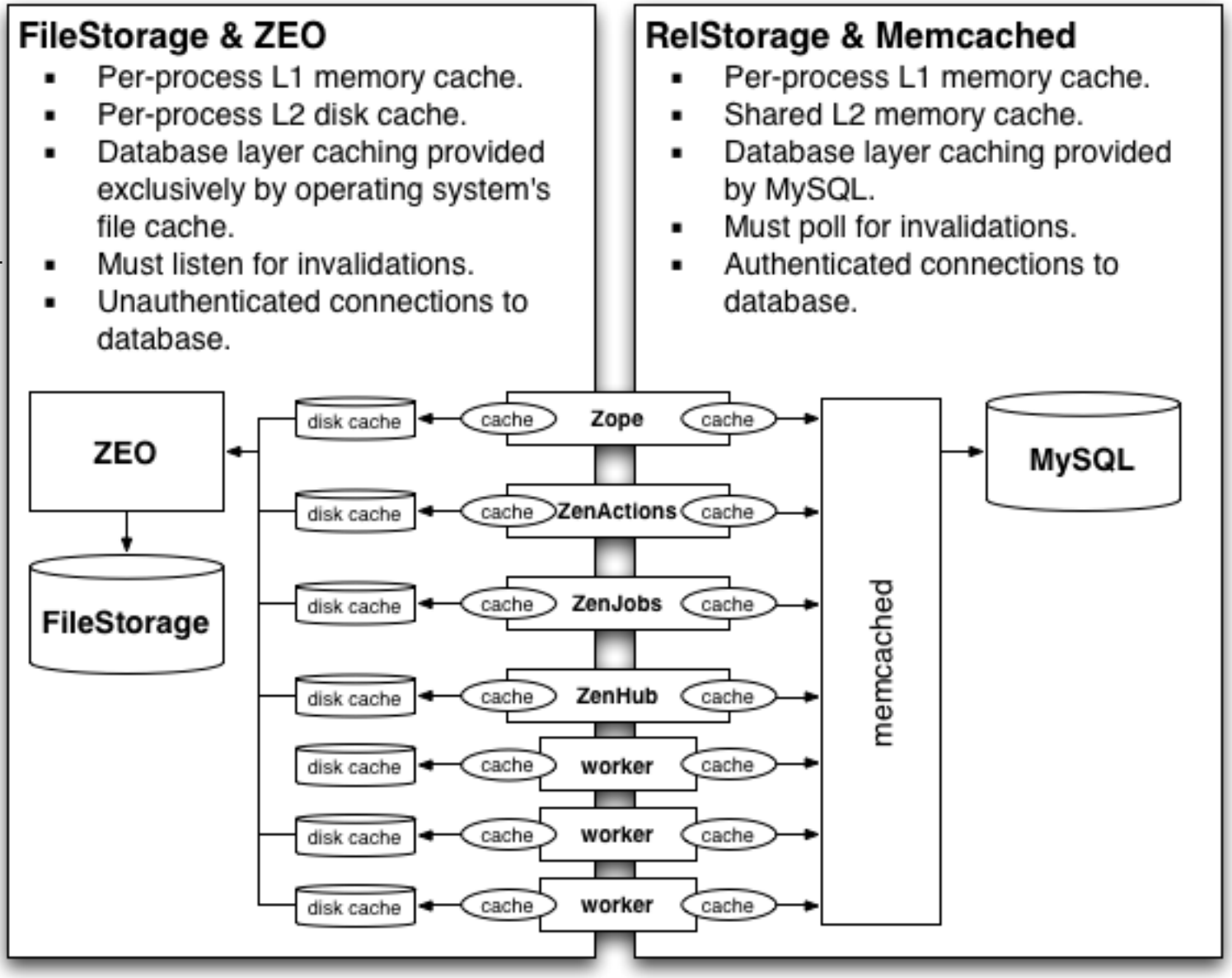


Other Zenoss 4 changes

- memcached subsystem
  - shared L2 memory cache for daemons
  - configured in */etc/sysconfig/memcached* and *$ZENHOME/etc/zope.conf* but **not** preallocated
- Zope database (zodb) now in MySQL database rather than *$ZENHOME/var/Data.fs*
- zodb_session database holds transient data
- RelStorage subsystem provides high-performance backend to ZODB into database
- Lucene subsystem for indexing of zodb and zenoss_zep
- Open source packages

Performance enhancements

Zenoss 2.x / Zenoss 3.x

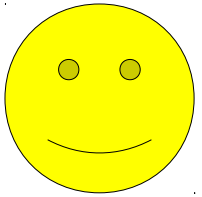


Zenoss 4.x

Intermission....

Back in 10 minutes.....

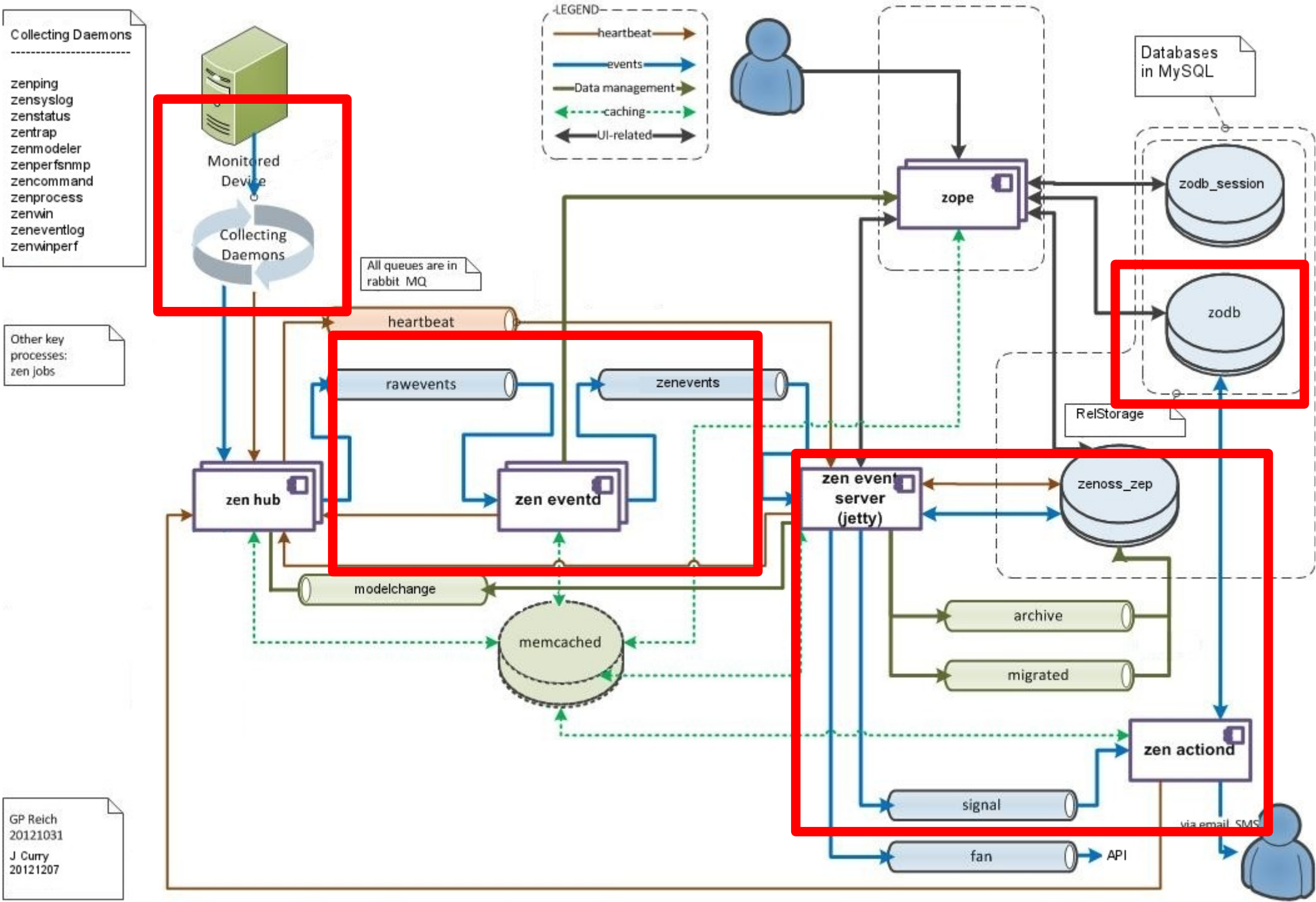
.... that's still 600 seconds



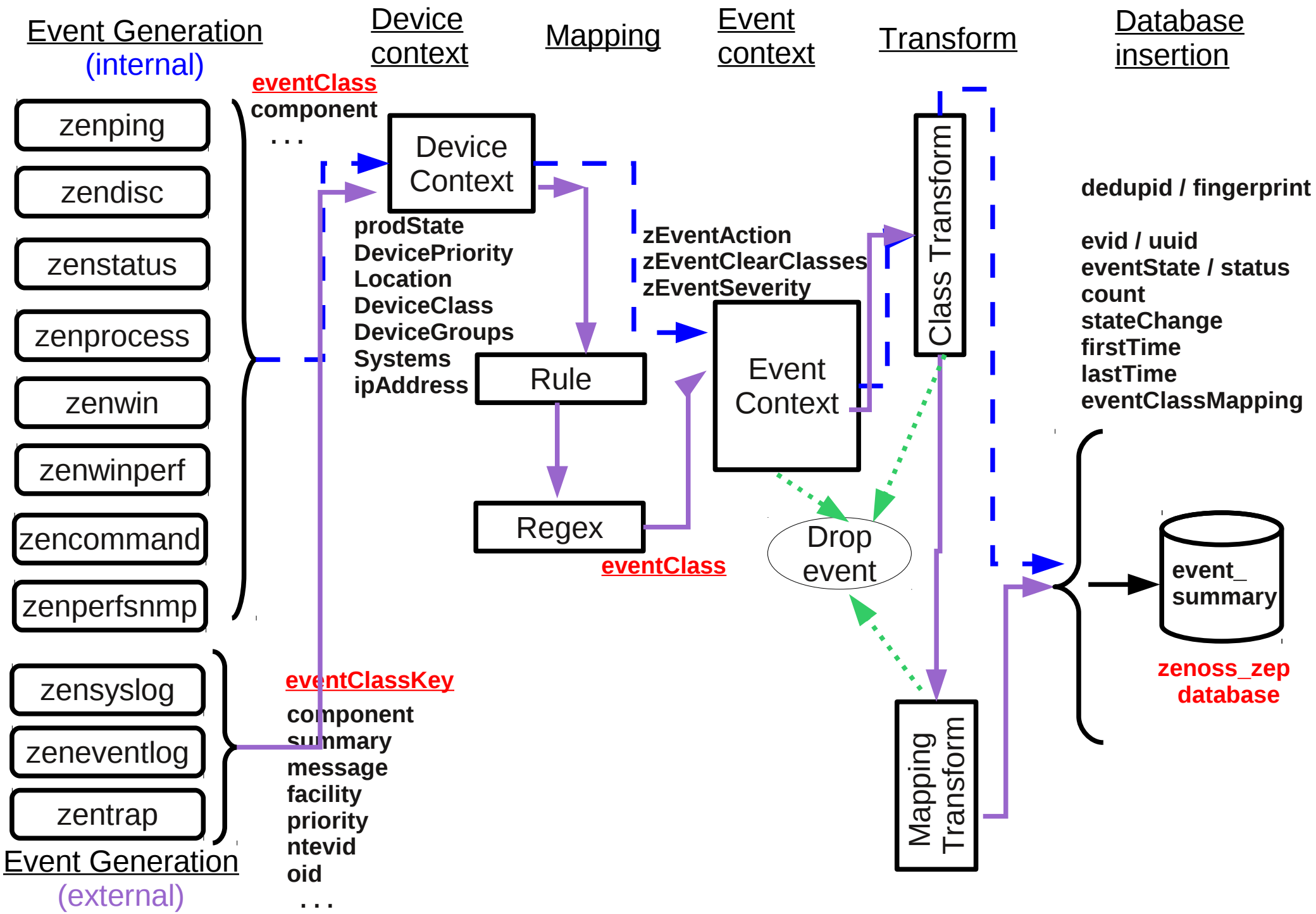
Event life cycle

- Event generation
- Device context
- Event class mapping
- Event context
- Event transform
- Database insertion and deduplication
- Resolution
- Ageing

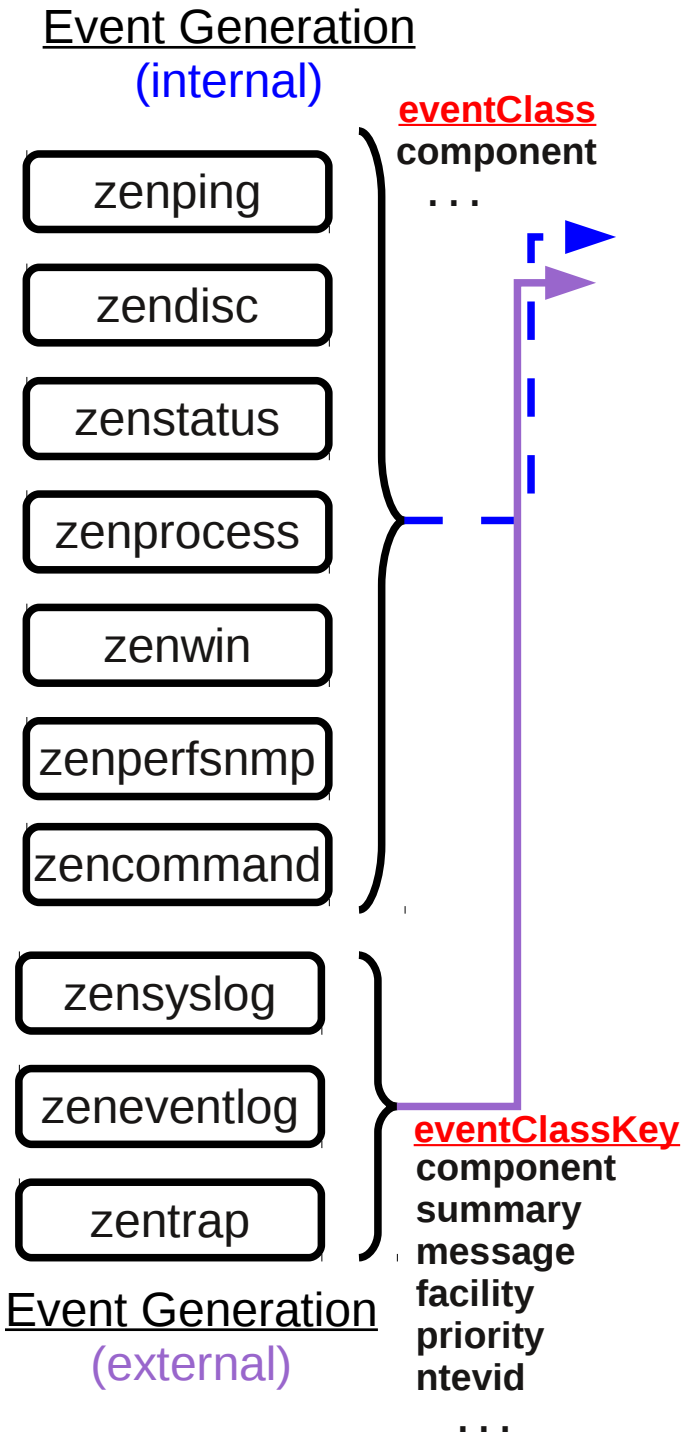
Zenoss Core 4 Event Architecture



Event Life Cycle – generation to initial database insertion



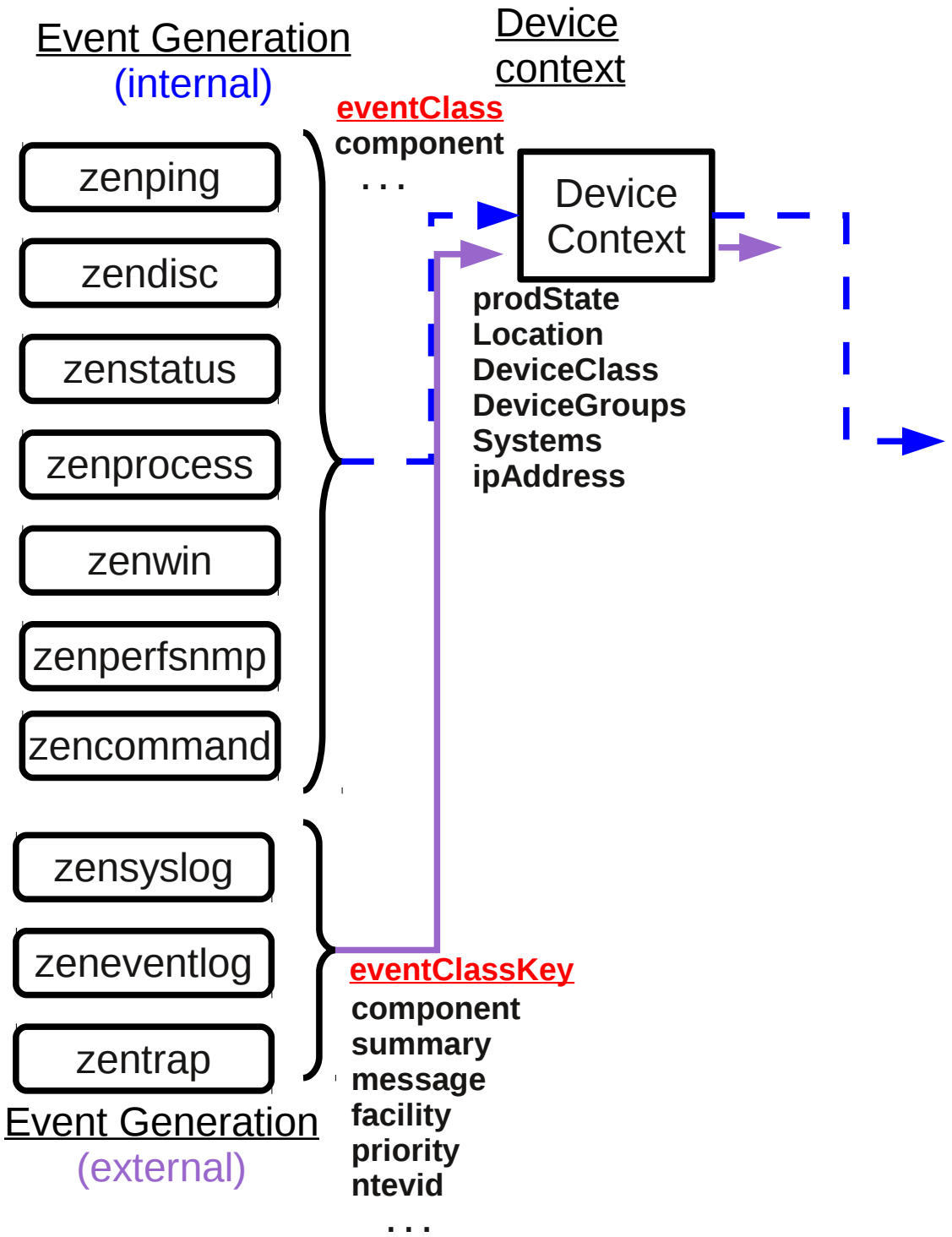
Event Life Cycle – generation



Event generation

- **Internal** event daemons
  - zenping
  - zendisc
  - zenstatus
  - zenprocess
  - zenwin
  - zenwinperf
  - zencommand
  - zenperfsnmp
  - other daemons introduced by ZenPacks
- **External** event daemons
  - zensyslog
  - zeneventlog
  - zentrap

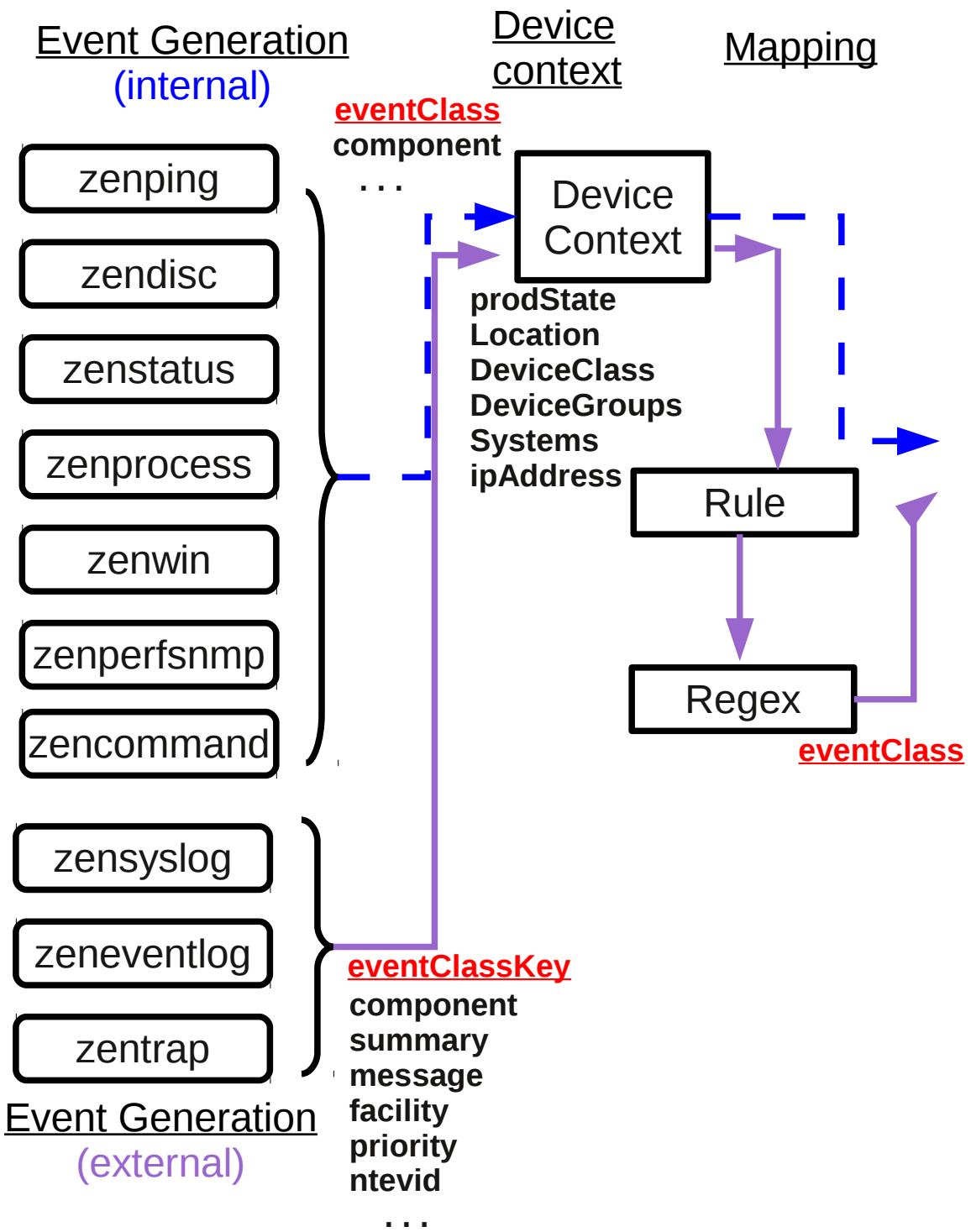
Event Life Cycle – generation to device context



Device context

- Event fields pertaining to the device that generated the event
  - prodState
  - Location
  - DeviceClass
  - DeviceGroups
  - Systems
  - ipAddress
- This configuration data is retrieved from the Zope Object Database (ZODB)

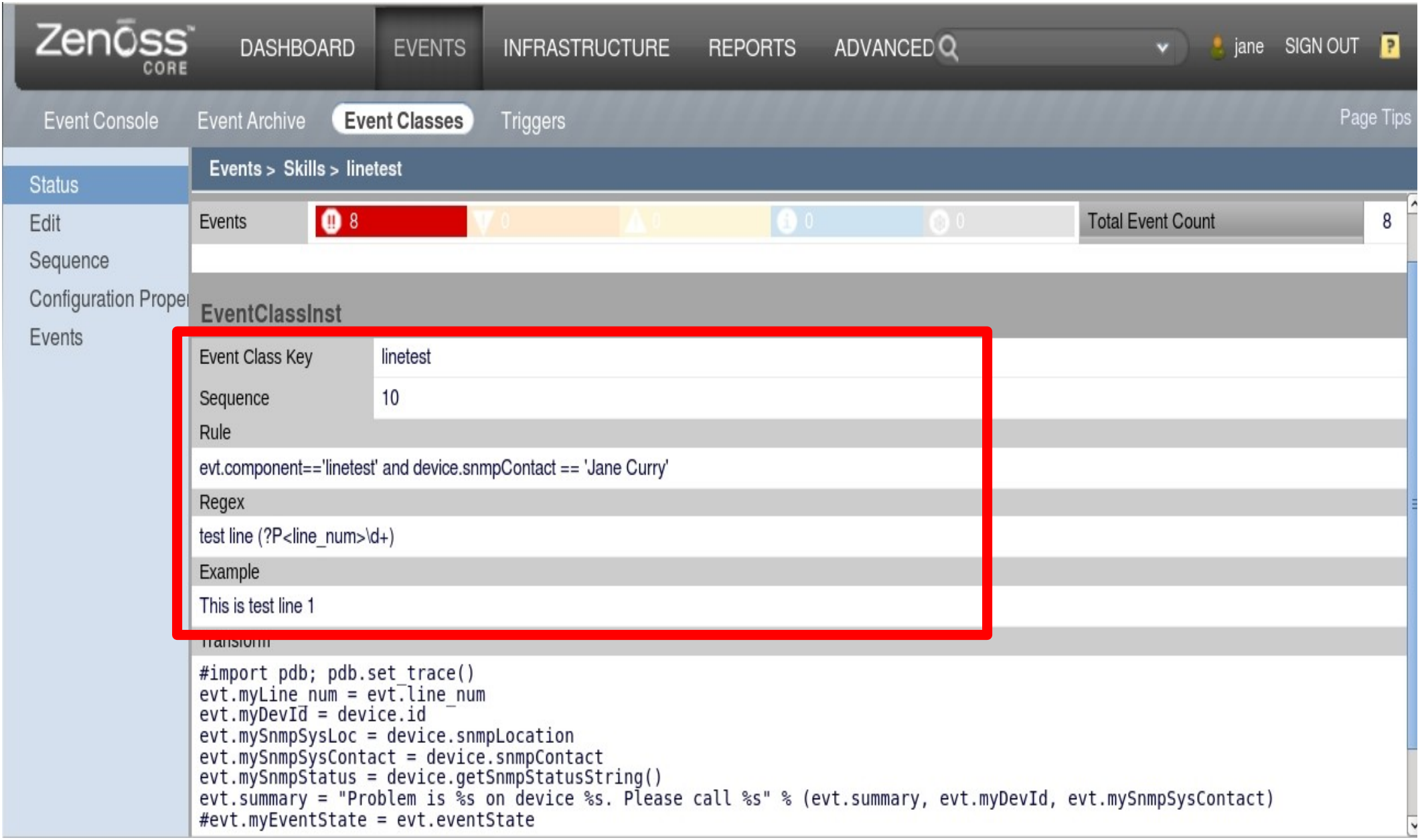
Event Life Cycle – generation to event class mapping



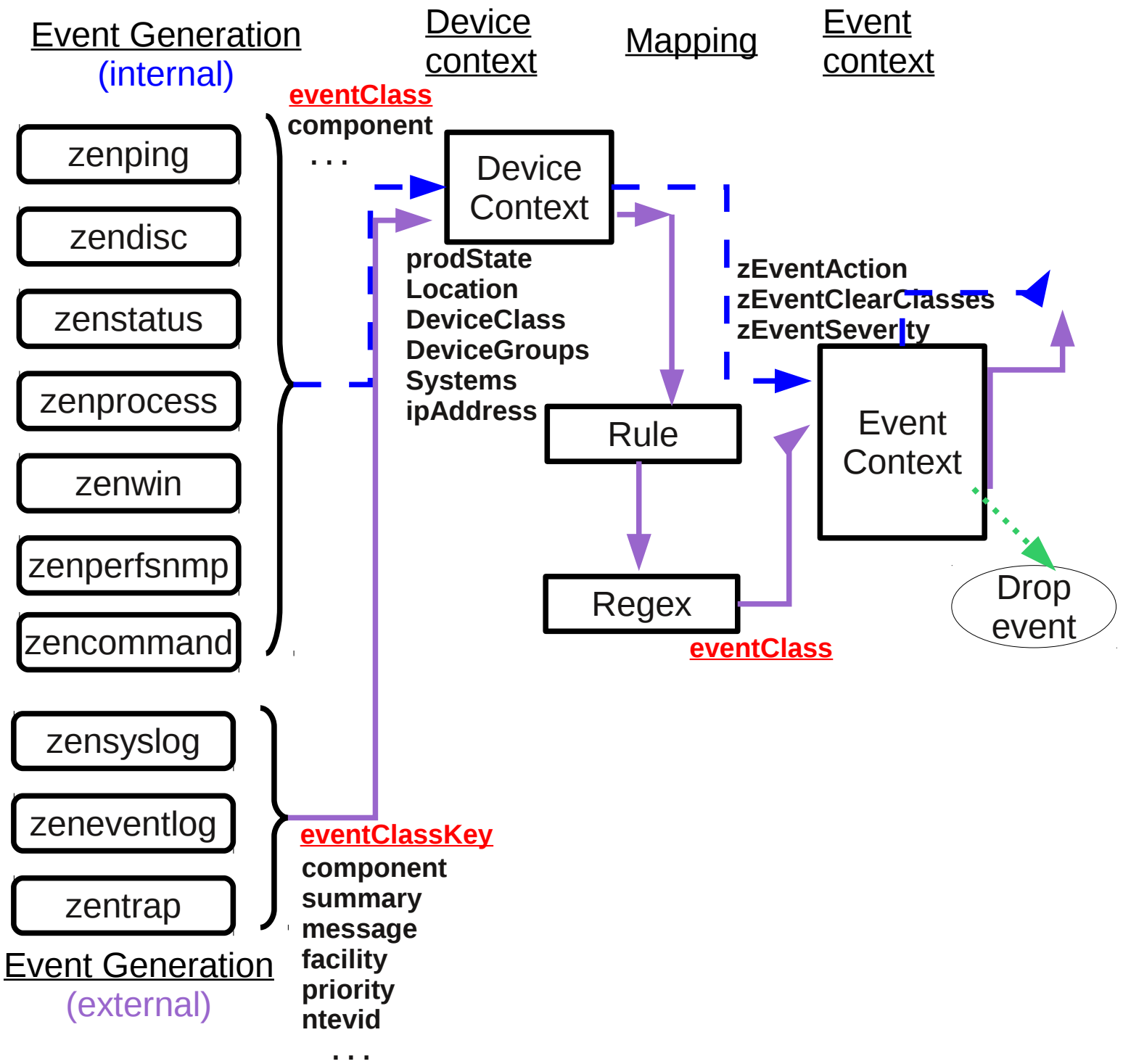
Event class mapping

- Used to map externally generated events to Zenoss event format
- Incoming event typically has **eventClassKey** field
  - Several source events may have same eventClassKey
- Target is to determine an **eventClass** field
- Python **Rule** tests any available field of event
  - If Rule exists, it must be satisfied
- Python **Regex** parses summary field of event
  - Regex must be satisfied if Rule doesn't exist
  - User-defined fields can be created if Regex matches
  - Regex need not be satisfied if Rule does exist
- Event mappings have sequence numbers

Event class mapping



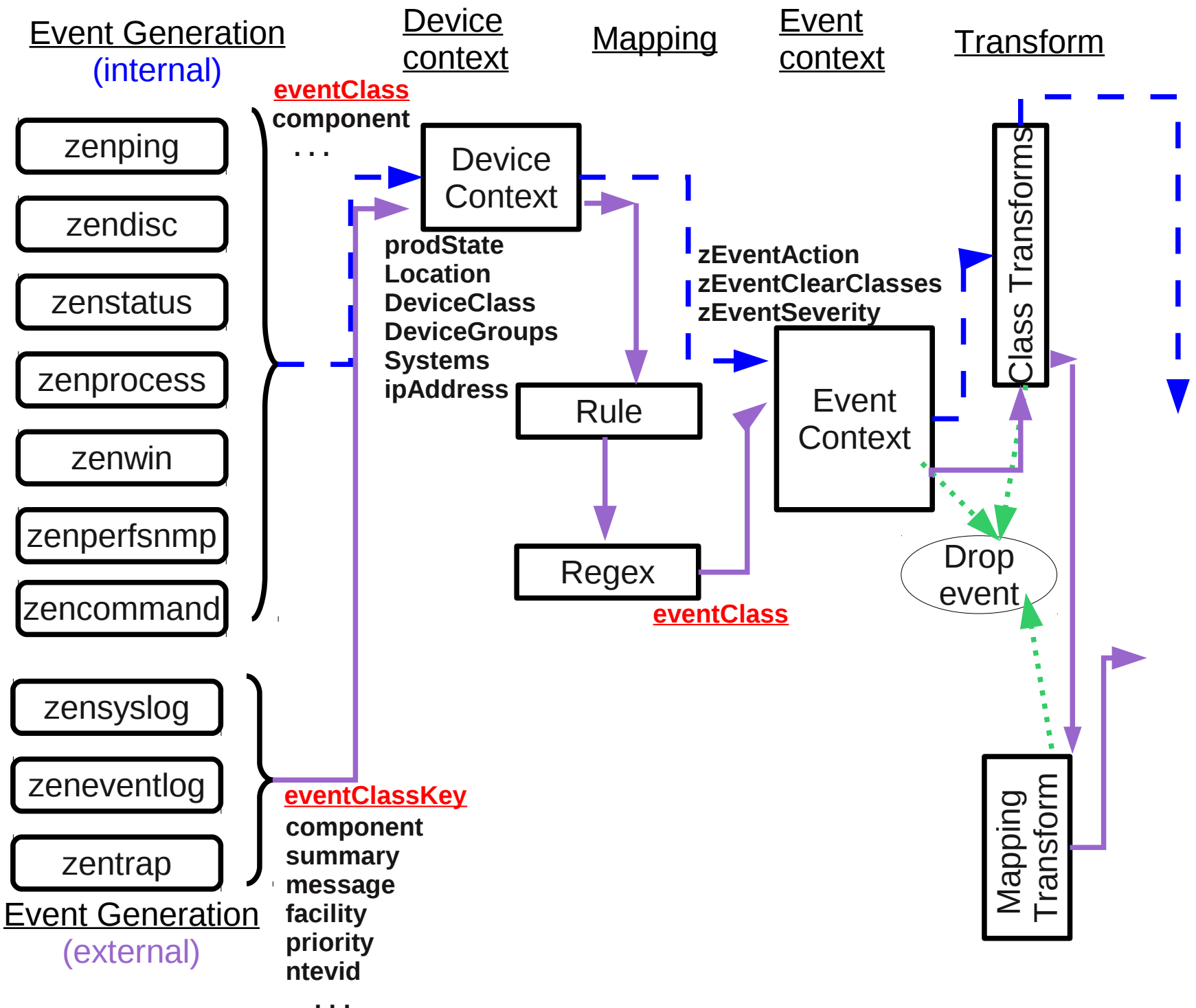
Event Life Cycle – generation to event context



Event context

- Defined by 3 event zProperties:
  - zEventAction status | history | drop
  - zEventClearClasses
  - zEventSeverity
- Event context is applied **after** event mapping but **before** event transforms
- Thus, zEventAction event context may specify *history* but an event transform could override by setting evt._action to the value *status*
    - these values reflect old database schema
        - status maps to eventState = New
        - history maps to eventState = Closed
        - both are stored in the event_summary table

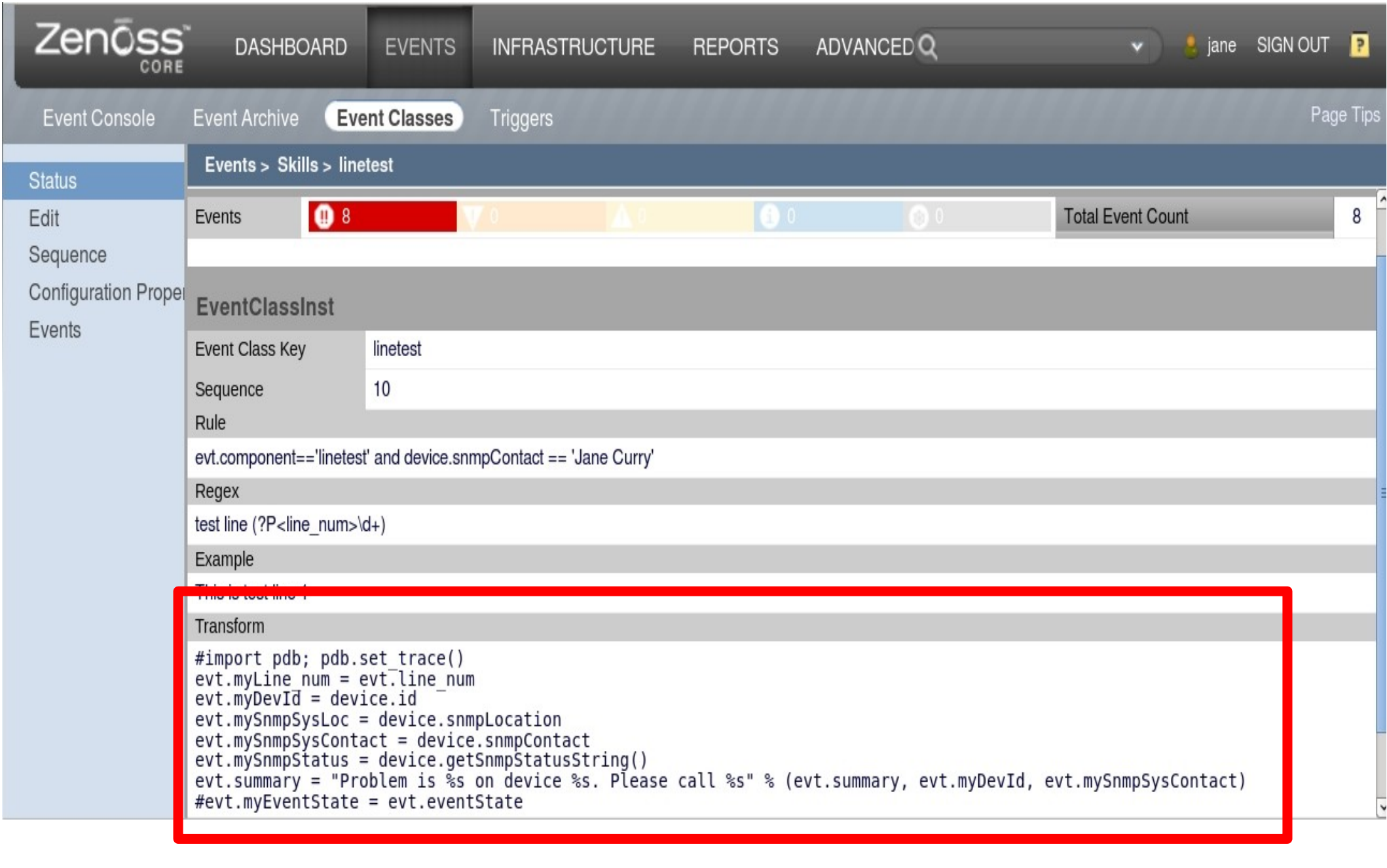
Event Life Cycle – generation to transform



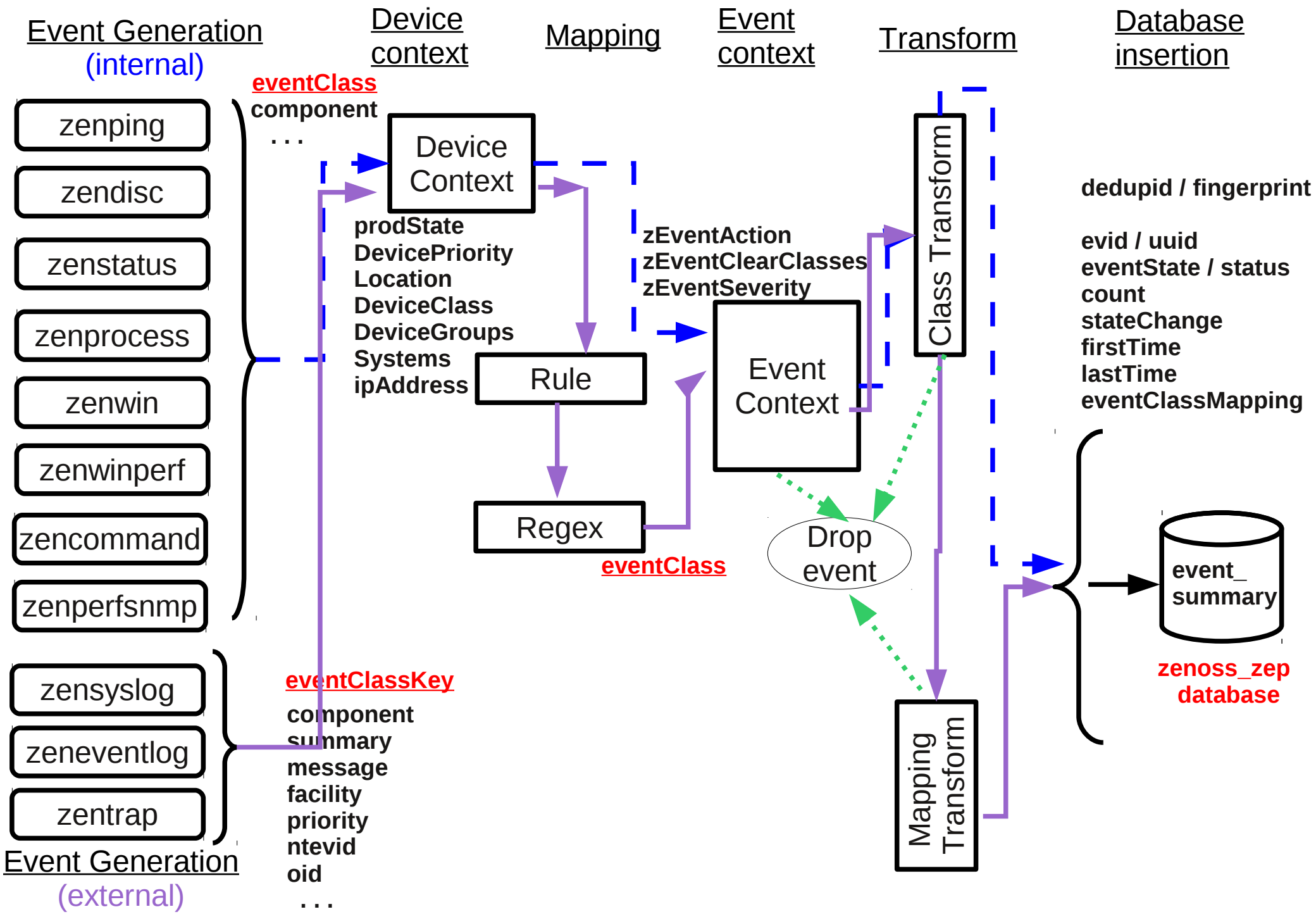
Event transforms

- Transforms can be applied to an event class mapping or to an event class
- Python statement(s) to modify:
  - Any available fields of the event
  - Any available property of the device
- Transform can create user-defined event fields
- Transform only applied if Rule / Regex satisfied
- Since 2.4 cascading event class transforms apply event class transform(s) *then* mapping transform

Event class mapping transform



Event Life Cycle – generation to database insertion



Database insertions

- All events inserted into **event_summary** table
- Some fields only assigned at insertion time:

  * count * stateChange *dedupid *eventState

  * firstTime * lastTime * evid * eventClassMapping
- Automatic duplicate detection based on:

  * device * component * eventClass

  * severity * eventKey * summary †
- dedupid made by concatenating above fields
- In Zenoss 4, this is known as the **fingerprint**
- Created by zeneventd; applied by zeneventserver
- Old event updated with details of new event

Resolution

- User closes event (eventState = Closed)
- Event context zEventAction = history / drop
- Transform sets evt_action to history / drop
- Clearing events will automatically clear all **similar** events, based on:

  * component UUID * eventClass *eventKey

  - if componentUUID exists; otherwise:

  * eventClass * device * component * eventKey

- zEventClearClasses zProperty on clearing events, **also** clears similar defined events (based on stated class, plus same conditions)
- eventState will be Cleared

Skills 1st

Ageing

- Event Manager provides for:
  - Events with severity < Error get eventState = Aged after 4 hours (configurable)
  - Event archive threshold default of 3 days after which Closed, Cleared and Aged events moved from event_summary to event_archive table
  - Delete Archived Events Older Than (days) really deletes data
- Deleting data now much faster - daily partitions
- Python script ZenDeleteHistory from Zenoss 3 does not exist or have equivalent function

Conclusions

- Understanding the architecture is vital
- Understanding the fields of an event is crucial
- Understanding the attributes of devices is important
- Knowledge of Python is required to write transforms
- Good knowledge of SNMP and syslog helps
- Working knowledge of SQL syntax helps
- 3-day Zenoss Event Management Workshop

http://www.skills-1st.co.uk/products/courses/

Skills 1st

Questions