

P99 CONF

# Overcoming Variable Payloads to Optimize for Performance



**Armin Ronacher**

Principal Architect at Sentry

# Armin Ronacher

Principal Architect at Sentry



- Creator of Flask, Werkzeug, Jinja and many Open Source libs
- Keep things running at Sentry, make event processing go vroom
- Got to learn to love event processing pipelines
- Juggling three lovely kids



# Why Are We Here?

# Sentry Generates, Processes and Shows Events

The screenshot displays the Sentry web interface. On the left is a dark sidebar with navigation options: Projects, Issues, Performance, Releases, User Feedback, Alerts, Discover, Dashboards, Profiling (marked alpha), Monitors, Activity, Stats, Settings, Help, What's new (with a red notification badge), and Collapse. The main content area is titled 'Issues' and shows a list of unresolved issues. At the top, it indicates 'All Unresolved 252', 'For Review 90', 'Ignored', and 'Saved Searches'. Below this is a filter bar with 'My Projects', 'All Env', '14D', and a search query 'is:unresolved'. The issues list includes columns for actions (Resolve, Ignore, Mark Reviewed, Merge, Last Seen), a graph, and counts for Events, Users, and Assignees. The issues shown are:

Issue	Events	Users	Assignee
Error: fn(src/screens/EndToEndTestsScreen) Unhandled Promise Rejection (SENTRY-REACT-NATIVE-97)   17min ago   7mo old	65	65	Assignee
Error: onPress(src/screens/EndToEndTestsScreen) captureException test (SENTRY-REACT-NATIVE-96)   18min ago   7mo old	65	65	Assignee
React Native Test Message (SENTRY-REACT-NATIVE-8E)   18min ago   9mo old	98	98	Assignee
Error: apply(native) Thrown Error (SENTRY-REACT-NATIVE-4A)   27min ago   15mo old	3	1	Assignee
Error: onPress(index) Thrown Error (SENTRY-REACT-NATIVE-5Y)   35min ago   12mo old	1	1	Assignee
Error: anonymous(src/screens/EndToEndTestsScreen) Unhandled Promise Rejection (SENTRY-REACT-NATIVE-93)   37min ago   7mo old	37	37	Assignee
Error: onPress(src/screens/EndToEndTestsScreen)			Assignee

# Sentry Generates, Processes and Shows Events

The screenshot displays the Sentry web interface for an error event. On the left is a dark purple sidebar with navigation options: Sentry SDKs (Armin Ronacher), Projects, Issues, Performance, Releases, User Feedback, Alerts, Discover, Dashboards (alpha), Profiling, Monitors, Activity, Stats, and Settings. The main content area is divided into several sections:

- SCREENSHOT:** A small thumbnail of the application's 'Unit of Bugs' page with a 'View screenshot' button.
- TAGS:** A list of key-value pairs for the event: device (System Product Name (System manufacturer)), environment (production), gpu.name (NVIDIA GeForce GTX 1080), gpu.vendor (NVIDIA), handled (no), level (error), mechanism (Unity.LogException), os (Windows 10), os.name (Windows), release (0.1 (0da87ba75977)), unity.device.device\_type (Desktop), unity.gpu.supports\_instancing (true), unity.install\_mode (Unknown), unity.is\_main\_thread (true), and user (id:0efcedaf9e4133441bb379826b5fb1de).
- EXCEPTION:** Shows the exception type 'System.NullReferenceException' with the message 'Object reference not set to an instance of an object.' and tags: mechanism (Unity.LogException), handled (false). Below this is a code snippet from 'ThreadingSamples+c.c' in 'ThrowNull (<lambdab>)' at line 18, with the error line highlighted: 'public void ThrowNull() => throw null;'. The assembly path is 'Path: ss\unity-of-bugs\Assets\Scripts\BugFarmButtons.cs'.
- Summary:** 'LAST 24 HOURS: 0', 'LAST 30 DAYS: 8', 'LAST SEEN: 7 days ago in release 0.1 (fc89f1a80c41)', and 'FIRST SEEN: 2 months ago in release 0.1'. There is a 'Create Ownership Rule' button and a link to 'Track this issue in Jira, GitHub, etc.'.
- Tags:** A horizontal bar chart showing the distribution of tags: device (MS-7D25 (Micro-Star Inte... 55%), environment (production 55%), gpu.name (Intel(R) UHD Graphics 770 55%), gpu.vendor (Intel 55%), handled (no 100%), level (error 100%), mechanism (Unity.LogException 100%), and os (Windows 11 55%).

# Sentry Events

- Session Updates
- Transaction Events
- Metrics
- Reports
  - Messages
  - Structured Processed Crash Reports
  - Structured Unprocessed Crash Reports
  - Minidumps
  - Third Party Crash Formats
  - User Feedback
  - Profiles
  - Attachments
  - Client Reports

# Challenges

- Users want crash reports with low latency
- Variance of processing times of events from 1ms to 30 minutes
- How long an event takes, is not always known ahead of time
- What happens at the end of the pipeline can affect the beginning of it
- Part of the pipeline is an Onion that can extend closer and closer to the user

# Conservative Changes



# Touching Running Systems

- Sentry processes complex events from many sources
- Any change (even bugfix) can break someone's workflow
- We are treating very carefully

Things we try to avoid doing:

- Bumping Dependencies without reason
- Rewriting services as busywork

That doesn't mean we don't change the pipeline, but we are rather conservative.

# Terms and Things

# “The Monolith”

- Written in Python
- A massive and grown Django app
- Uses celery and rabbitmq historically for all queue needs
- Still plays a significant role in the processing logic
- Uses CFFI to invoke some Rust code

# Relay

- Written in Rust
- Our ingestion component
- Layers like an onion
- Stateful
- First level quota enforcement
- Aggregation
- Data normalization
- PII stripping

# Symbolicator

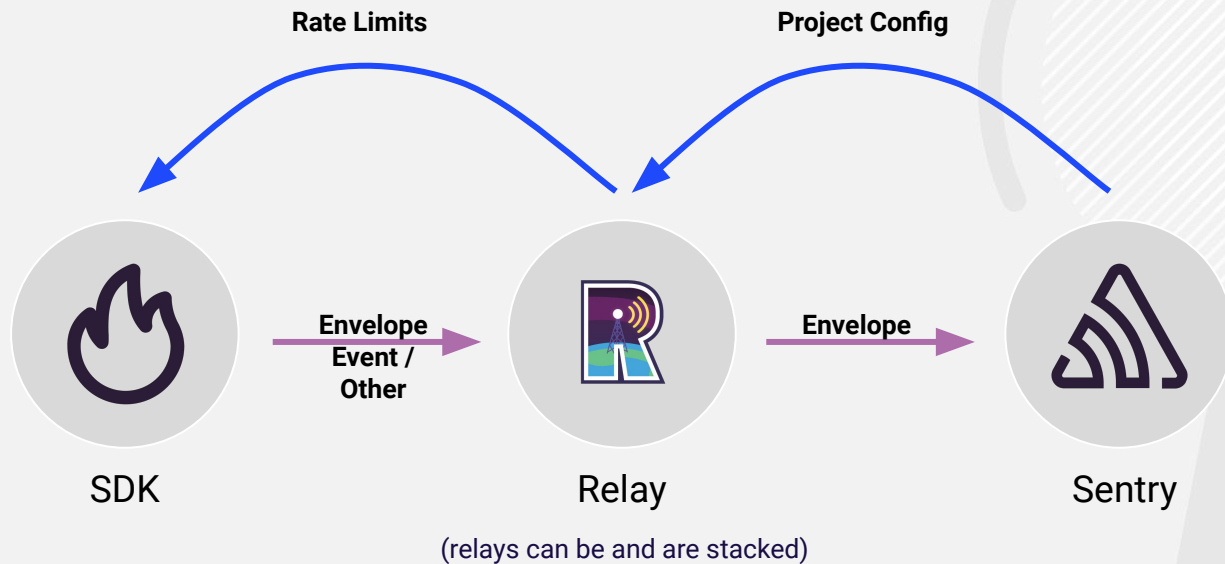
- Written in Rust
- Handles Symbolication
  - PDB
  - PE/COFF
  - DWARF
  - MachO
  - ELF
  - WASM
  - IL2CPP
- Fetches and Manages Debug Information Files (DIFs)
  - External Symbol Servers
  - Internal Sources

# Ingest Consumer

- Shovels Pieces from the Relay supplied Kafka stream onwards
  - Events
  - User Reports
  - Attachment Chunks
  - Attachments
- Does an initial routing of events to the rest of pipeline

# What's Flowing?

# Ingestion Side

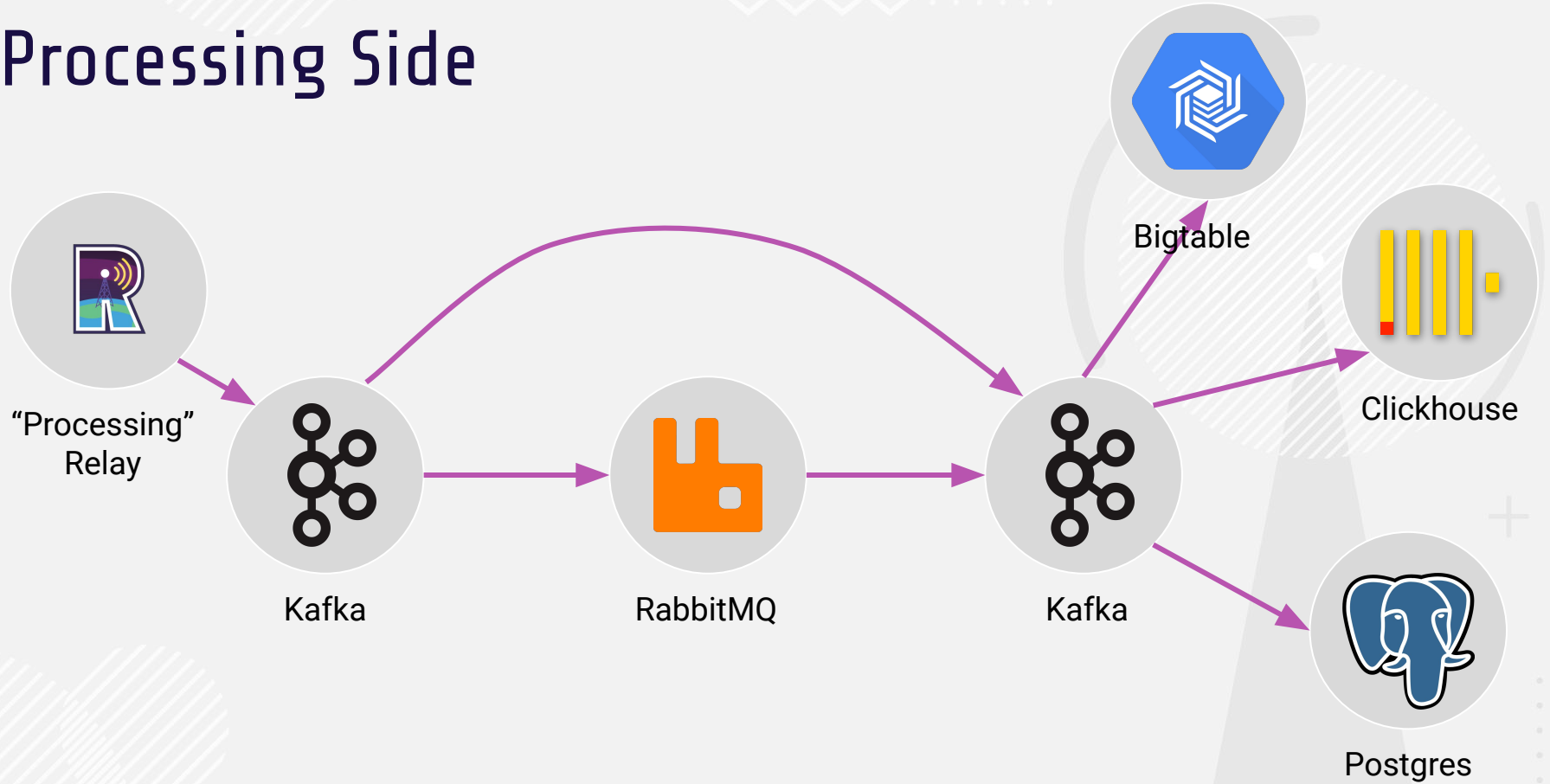




# Ingestion Traffic

- POP Relays accepts around 100k events/sec at regular day peak and rejects around 40k/sec
- Processing relays process around 150k events/sec at regular day peak
- Global Ingestion-Level Load Balancers see around 200k req/sec at regular peak

# Processing Side



# Kafka Traffic

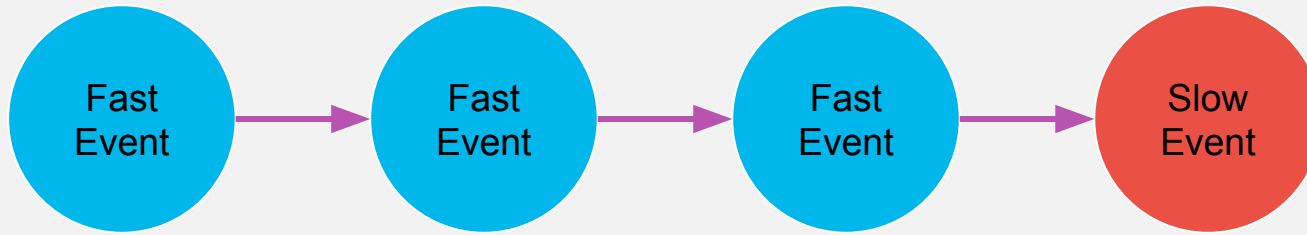
- All relay traffic makes it to different Kafka topics
- Important ones by volume:
  - Sessions/Metrics
  - Transactions
  - Error events
  - Attachments
- Based on these event types, initial routing happens
- **The biggest challenge are error events**

# Error Event Routing

- Ahead of time, little information is available to determine how long an event will take
- Cache status can greatly affect how long it takes
  - JavaScript event without source maps can take <1ms
  - JavaScript event that requires fetching of source maps can take 60sec or more
  - Native events might pull in gigabytes of debug data, that's not yet hot
- A lot of that processing still happens in legacy monolith

# The Issue with Variance

# Head of Line Blocking within Partition



# Our Queues: Kafka and RabbitMQ

- Kafka has inherent head-of-line blocking
- Our Python consumers have language limited support for concurrency
- Writing a custom broker on top of Kafka carries risks
- Historically our answer was to dispatch from Kafka to Rabbit for high variance tasks

# We're Not Happy with RabbitMQ

- As our scale increases, we likely will move to Kafka entirely
- This switch will require us to build a custom broker
- So far the benefits of that have not yet emerged
- It works good enough for now™

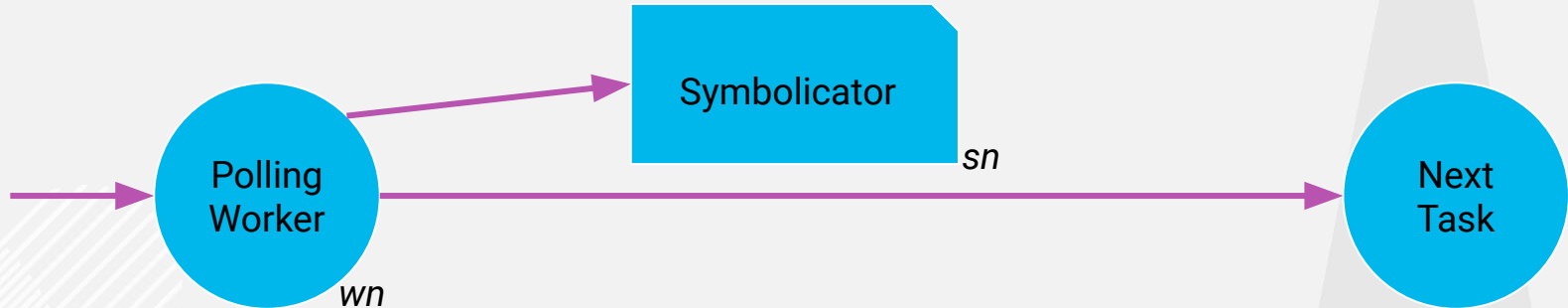


# Tasks on RabbitMQ

- Tasks travel on RabbitMQ queues
- Event payloads live in redis
- Python workers pick up tasks as they have capacity available
- Problem: polling workers

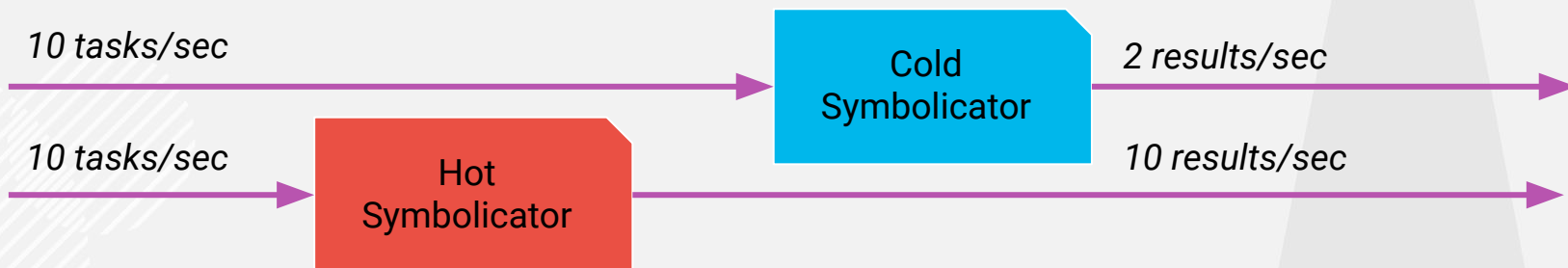
# Polling Workers

- Some tasks poll the internal symbolicator service
- For that a Python worker dispatches a task via HTTP to the stateful symbolicator service
- Python worker polls that service until result is ready which can be minutes
- Requires symbolicators to be somewhat evenly configured and loaded



# Incident: Symbolicator Tilt

- Fundamental flaw: tasks are pushed evenly to symbolicators
- Not all symbolicators respond the same
- A freshly scaled up symbolicator has cold caches
- This caused scaling up to have a negative effect on processing times
- Workaround: **cache sharing**
- Long term plan: symbolicator picks up directly from RabbitMQ or Kafka



# Backpressure Control

# Implicit Backpressure Control

- Our processing queue has insufficient backpressure control
- At the head of the queue we permit almost unbounded event accumulation
- Pausing certain parts of the pipeline can cause it to spill too fast into RabbitMQ (goes to swap)

# Deep Load Shedding

# Pipeline Kill-Switches

- Problem: for some reason bad event data makes it into the pipeline
- Due to volume we cannot track where the data is in the pipe and we likely can't reliably prevent it from propagating further
- Solution: flexible kill-switches
- Drop events that match a filter wherever that filter is applied

# Loading Kill-Switches

```
sentry killswitches pull \  
  store.load-shed-group-creation-projects \  
  new-rules.txt
```

Before: <disabled entirely>

After:

```
DROP DATA WHERE  
  (project_id = 1) OR  
  (project_id = 2) OR  
  (project_id = 3)
```

Should the changes be applied? [y/N]: y



# Look into Relay

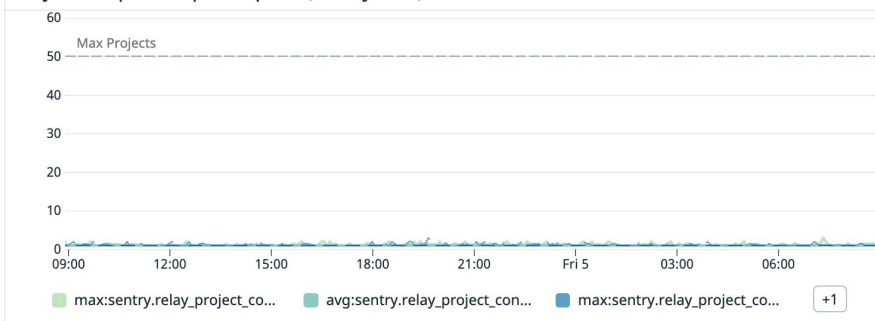
# Communication Channels

- Relay to Relay: HTTP
- Relay to Processing Pipeline: Kafka
- Relay state updates:
  - Relay -> Relay via HTTP
  - Relay to Internal HTTP and direct redis cache reads

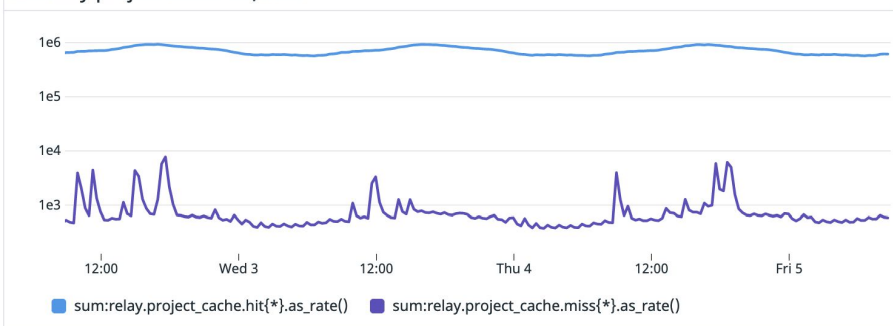
# Project Config Caches

- Innermost relays fetch config directly from Sentry
- Sentry itself persists latest config into redis
- Relay will always try to read from that shared cache before asking Sentry

Projects requested per request (Sentry POV)

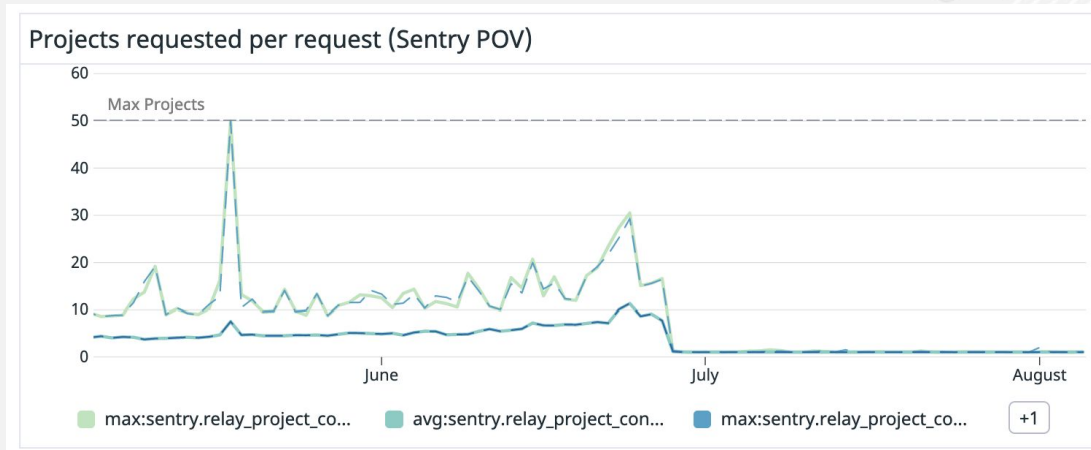


Memory project cache hit/miss



# Proactive Cache Writing

- We used to expire configs in cache liberally
- Now most situations will instead proactively rewrite configs to cache





Armin Ronacher  
[armin@sentry.io](mailto:armin@sentry.io)  
@mitsuhiko