Darf's ein bisschen mehr sein?

Cloud Monitoring mit PCP

Werner Keil

Developer Week 2017

27. Juni 2017

DWX Developer Week 2017

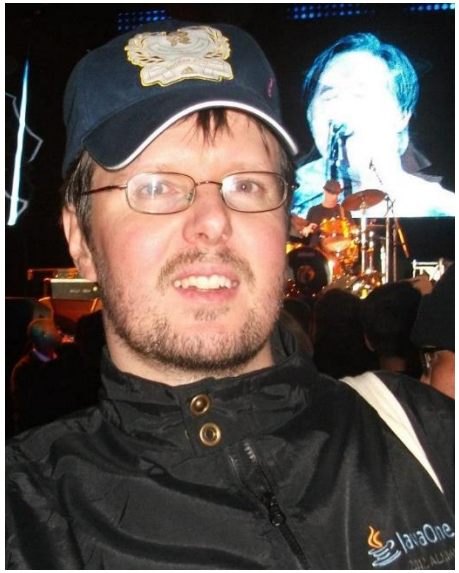
CATMedia

Agenda

- Einleitung
- Performance Co-Pilot
- Dropwizard Metrics
- Apache Sirona
- Eclipse MicroProfile
- StatsD
- Demo
- Q&A

CATMedia

Was bin Ich?



Werner Keil

Twitter @wernerkeil

- Consultant – Coach
- Creative Cosmopolitan
- Open Source Evangelist
- Software Architekt
- JCP EC Mitglied
- Eclipse UOMo Projektleiter
- JSR-363 Co Spec Lead
- IoT Enabler …

Email werner@catmedia.us

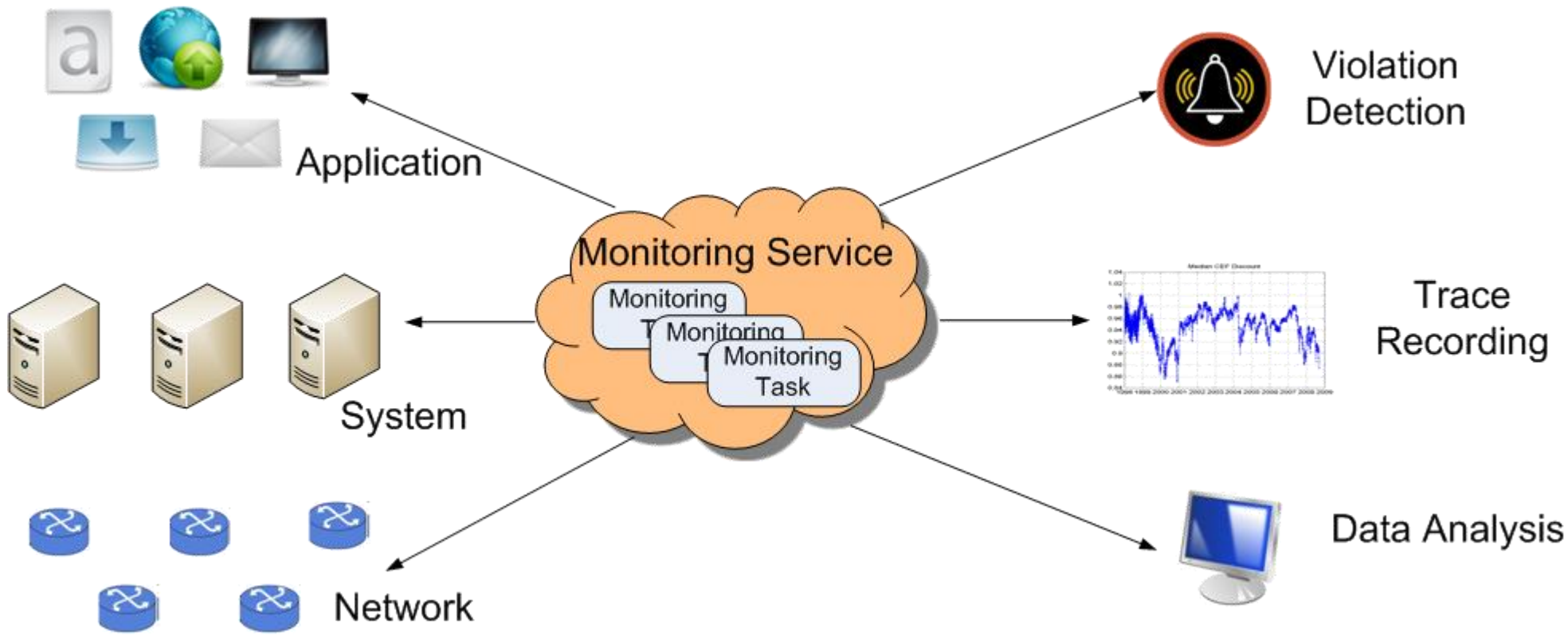
CATMedia

Was ist Monitoring?

Unter Monitoring von Anwendungen versteht man die systematische Erfassung, Messung und Analyse des Verhaltens der betreffenden Anwendungen, etwa CPU Auslastung, Speicherverbrauch oder andere Informationen wie die verwendeten Klassen und Threads.

A particular case is the monitoring of distributed applications, aka the Cloud where an the performance analysis of nodes and communication between them pose additional challenges.

Arten von Cloud Monitoring



CATMedia

Herausforderungen auf Systemebene

- Effizienz und Skalierbarkeit
  - Unterstützung Zehntausender Überwachungsaufgaben
  - Kosteneffizienz durch sparsame Nutzung von Ressourcen
- Monitoring QoS
  - Multi-Tenancy Umgebung
  - Minimierung von Ressourcenkonflikten zwischen Überwachungsaufgaben
- Implikation von Multi-Tenancy
  - Hinzufügen und Entfernen von Überwachungsaufgaben
  - Ressourcenkonflikte zwischen Überwachungsaufgaben

Performance vs. Anzahl der Server

60 Elemente pro Server, Updates einmal pro Minute

Anzahl der Server	Performance (Werte pro Sekunde)
100	100
1000	1000
10000	10000

600 Elemente pro Server, Updates einmal pro Minute

Anzahl der Server	Performance (Werte pro Sekunde)
100	1000
1000	10000
10000	100000

Monitoring Tipps

- Regelmäßig “Little’s Law” auf alle Daten anwenden ... Hier die generische Variante aus der Warteschlangen-Theorie:

  $Q = \lambda R$

- Length = Arrival Rate x Response Time
  - z.B. 10 MB = 2 MB/s x 5 s
- Utilization = Arrival Rate x Service Time
  - z.B. 20% = 0.2 = 100 ms/s x 2 s

Arten des Monitoring

Monitoring Logs

- Logstash
- Redis
- Elasticsearch
- Kibana Dashboard

Monitoring Performance

- Collectd
- Statsd
- PCP
- Graphite
- Datenbank (z.B.: PSQL)
- Grafana Dashboard

Monitoring Logs – Kibana Dashboard

Performance Monitoring

Wie wird dies traditionell gemacht?

- rsyslog/syslog-ng/journald
- top/iostat/vmstat/ps
- Mischung verschiedener Skript Sprachen (bash/perl/python)
- Tools je nach Plattform sehr verschieden
- Korrekte Analyse erfordert entsprechenden Kontext

Performance Co-Pilot

PCP

http://www.pcp.io

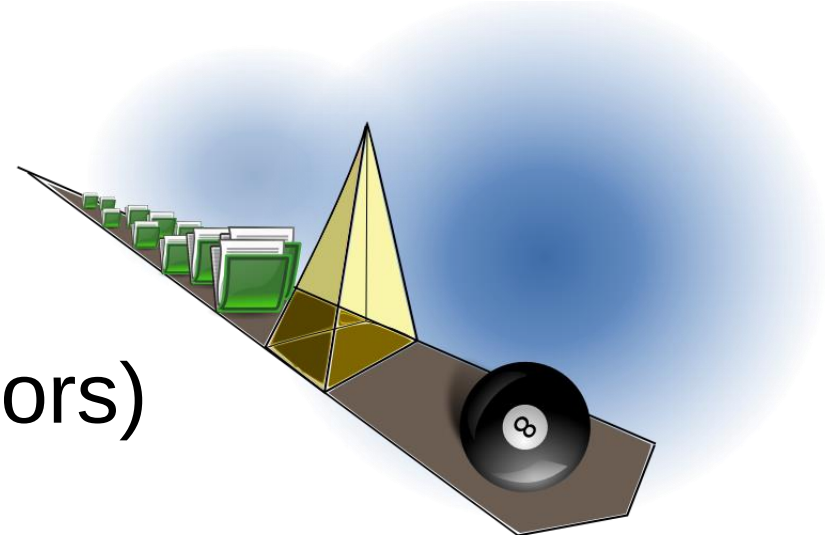
GitHub

https://github.com/performancecopilot

CTMedia

Was ist PCP?

- Open Source Toolkit
- Analyse auf Systemebene
- Live und Historisch
- Erweiterbar (Monitors, Collectors)
- Verteilt
- Unix-artiges Komponentendesign
- Cross-Plattform
- Universelle Maßeinheiten



CATMedia

PCP Grundlagen

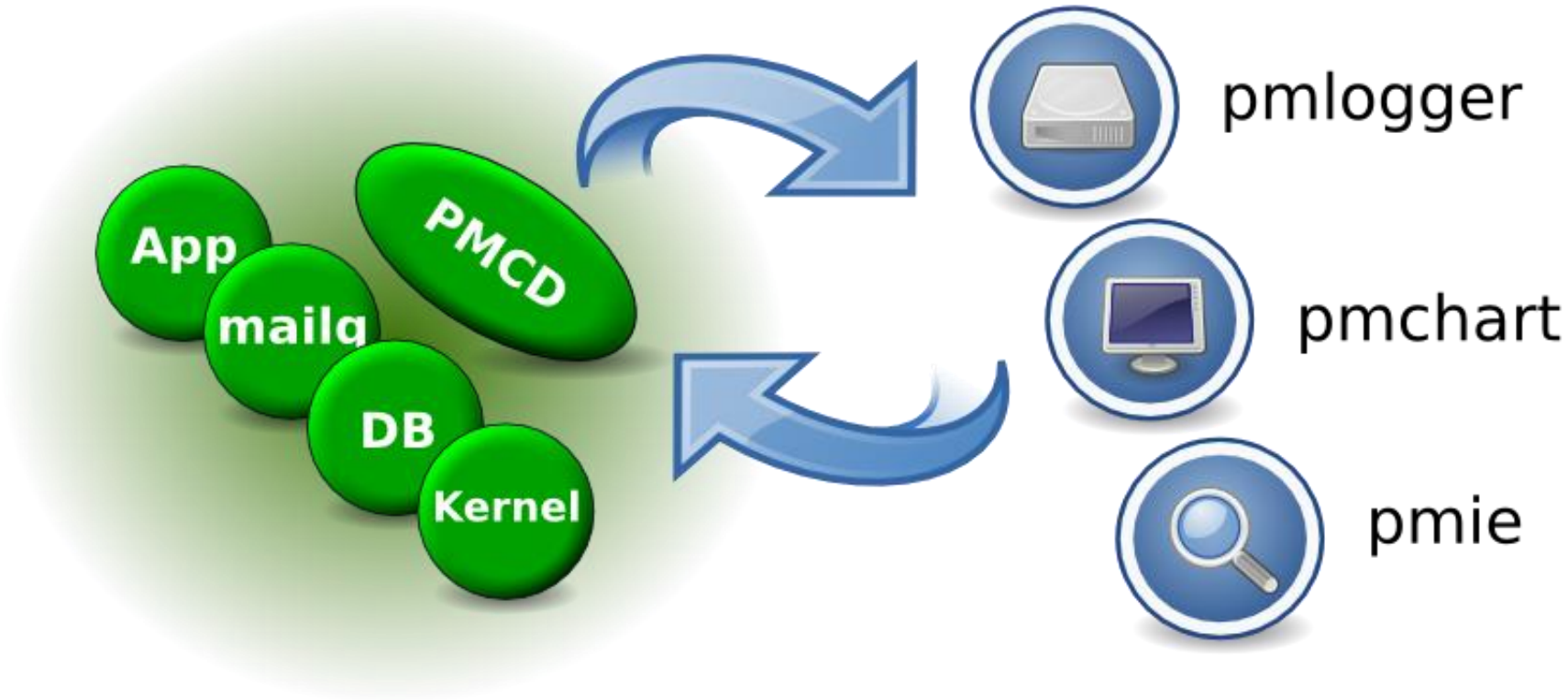
Agents and Daemons

PCP basiert auf zwei wesentlichen Bestandteilen:

1. Performance Metric Domain Agents
   - Agents
2. Performance Metric Collection Daemon
   - PMCD

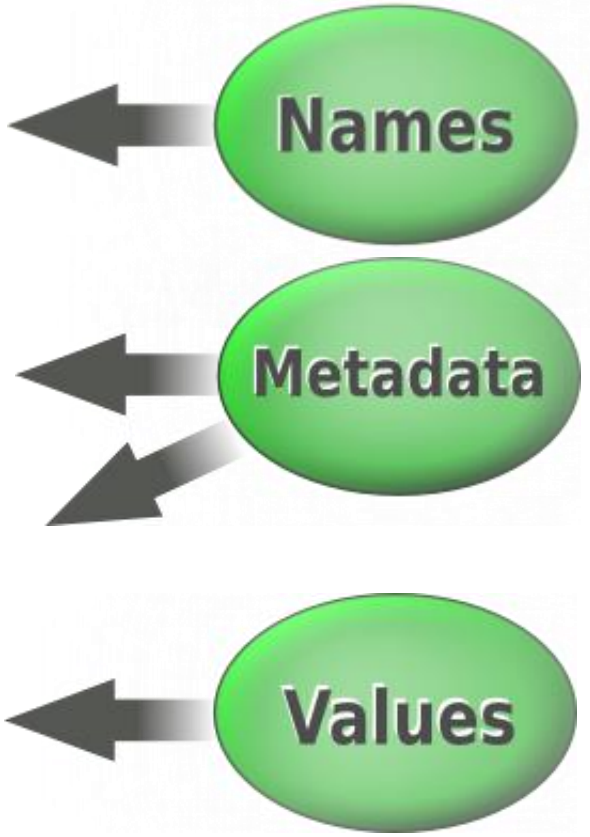
CATMedia

PCP Architektur



PCP Metriken

pminfo --desc -tT --fetch ***disk.dev.read***



disk.dev.read [*per-disk read operations*]

Data Type: *32-bit unsigned int* InDom: *60.1*

Semantics: *counter* Units: *count*

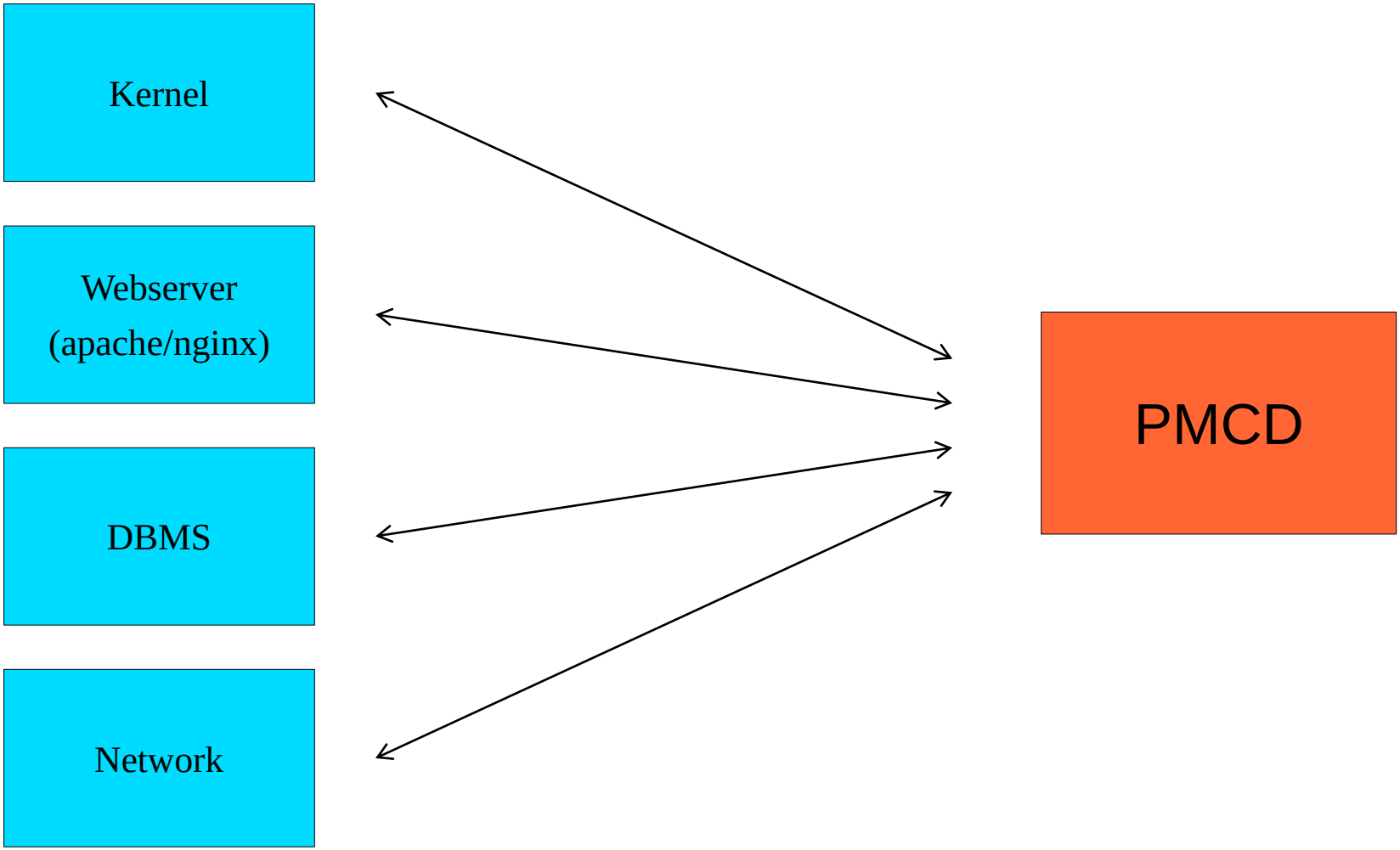
Help: *Cumulative count of disk reads since boot time*

Values:

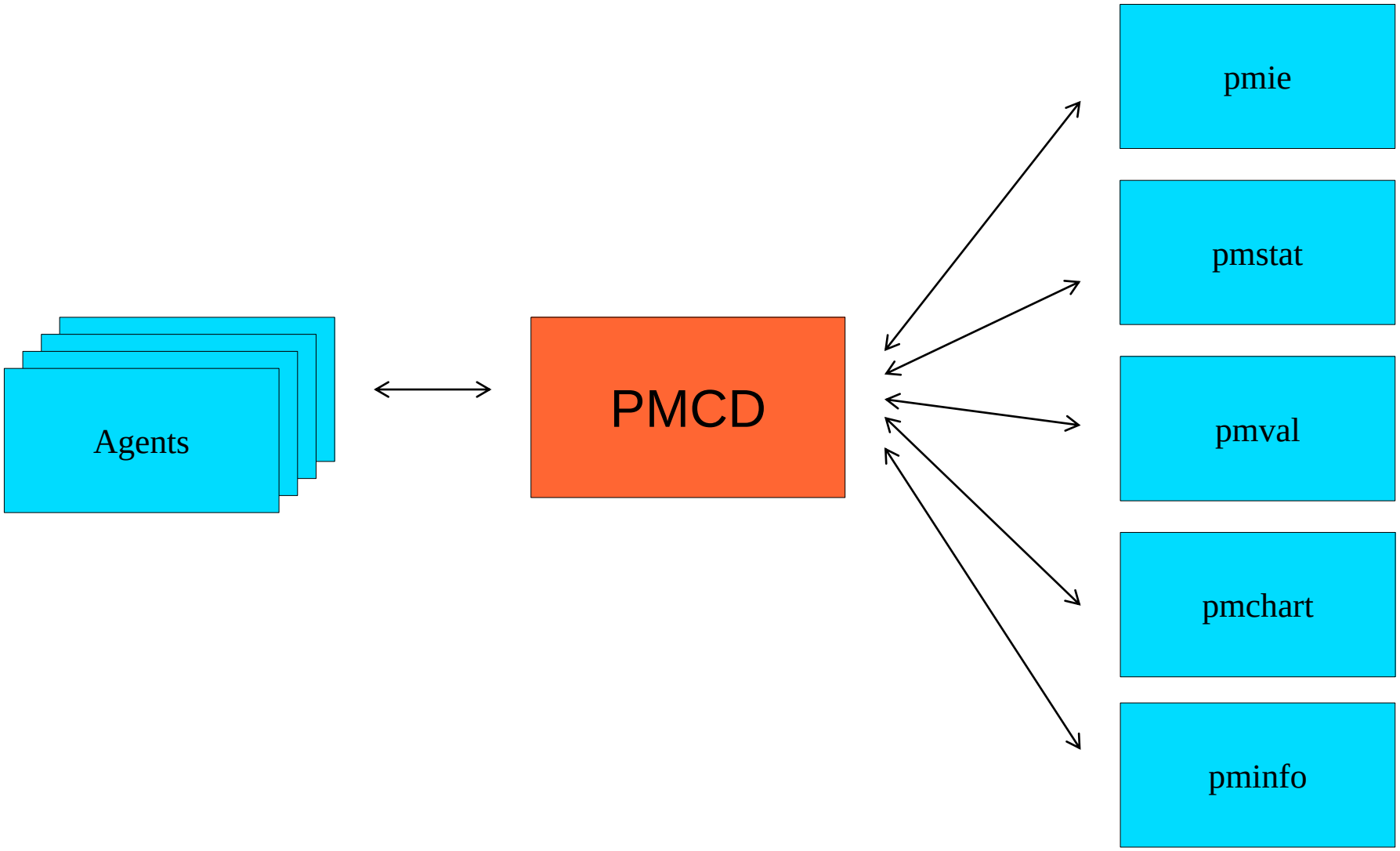
inst [0 or "***sda***"] value ***3382299***

inst [1 or "***sdb***"] value ***178421***

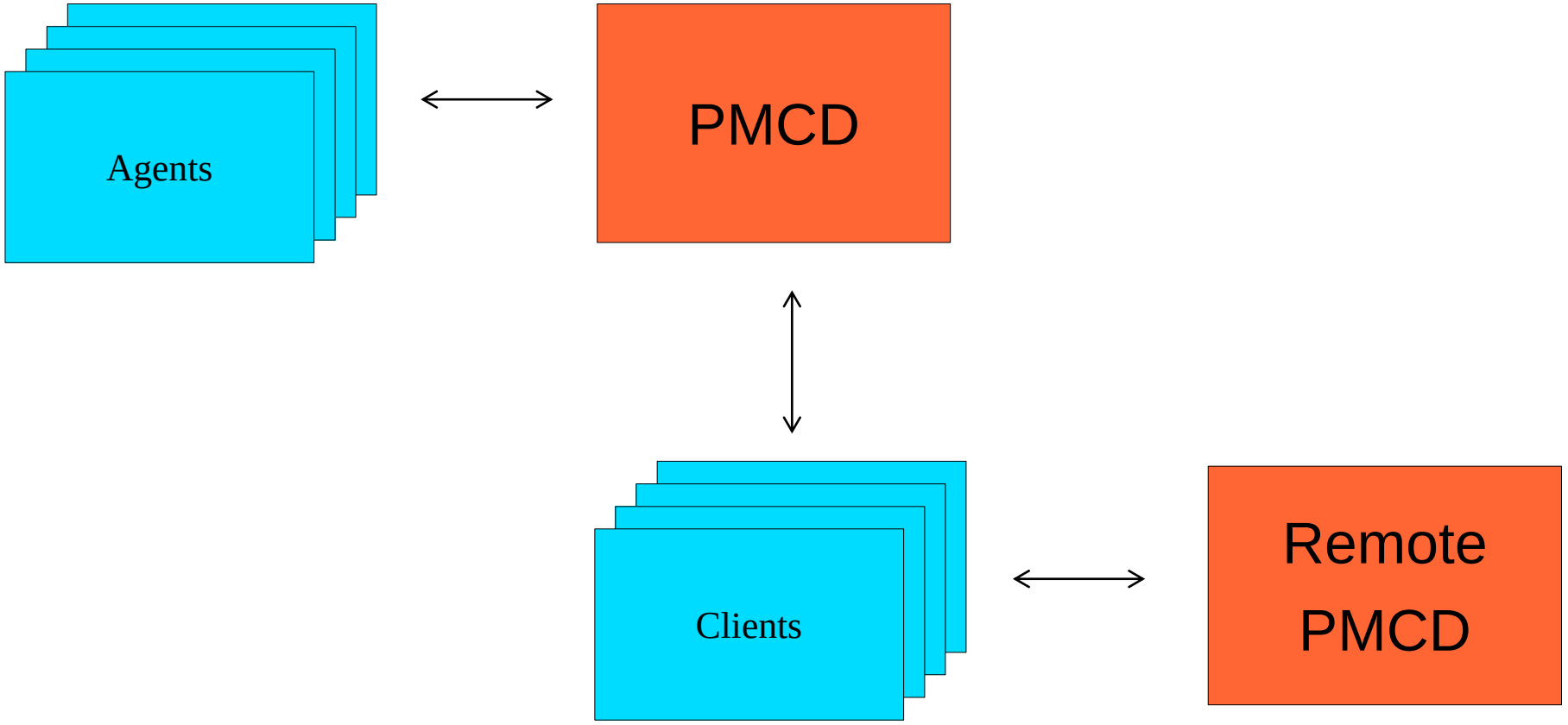
PCP Agenten



PCP Clients



PCP Remote Clients



PCP Datenmodell

- Metrics come from one source (host / archive)
- Source can be queried at any interval by any monitor tool
- Hierarchical metric names
  e.g. disk.dev.read and
  aconex.response_time.avg

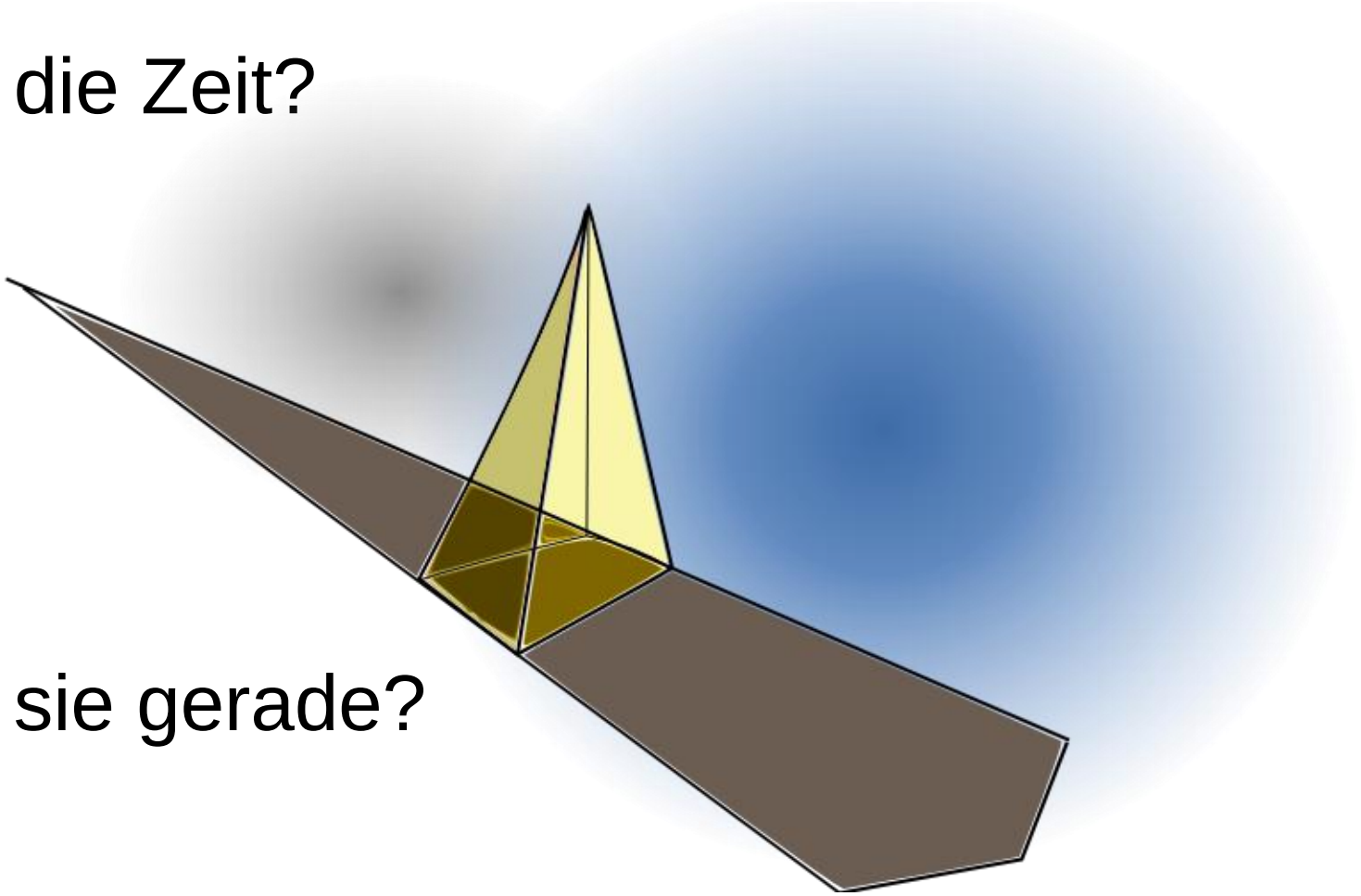
CATMedia

PCP Datenmodell (2)

- Metrics are singular or set-valued ("instance domain")
- Metadata associated with every metric
  - Data type (int32, uint64, double, ...)
  - Data semantics (units, scale, ...)
  - Instance domain

Performance Timeline

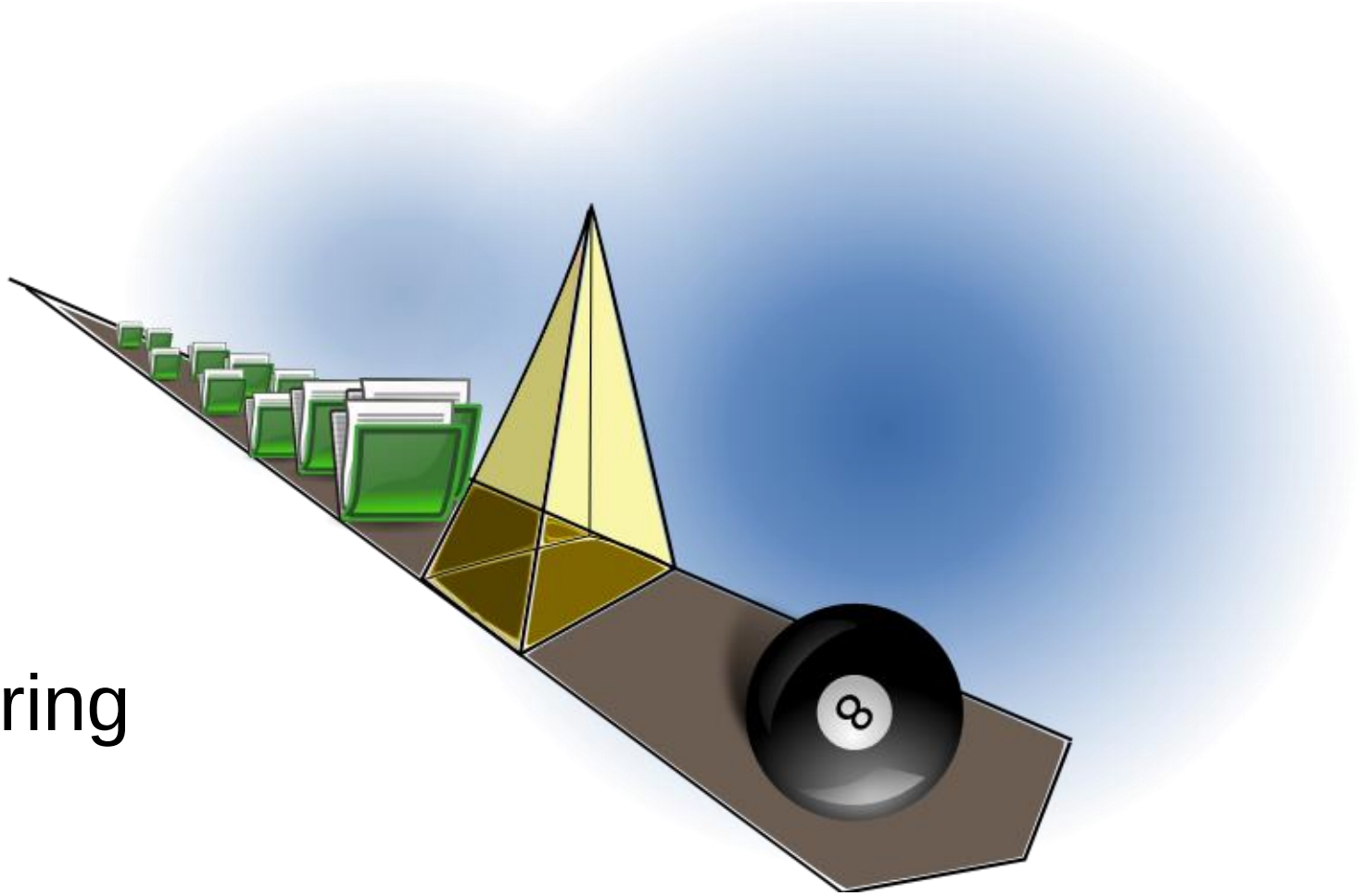
- Wohin geht die Zeit?



- Wohin geht sie gerade?
- Wohin wird sie gehen?

Performance Timeline – PCP Toolkit

- Archive



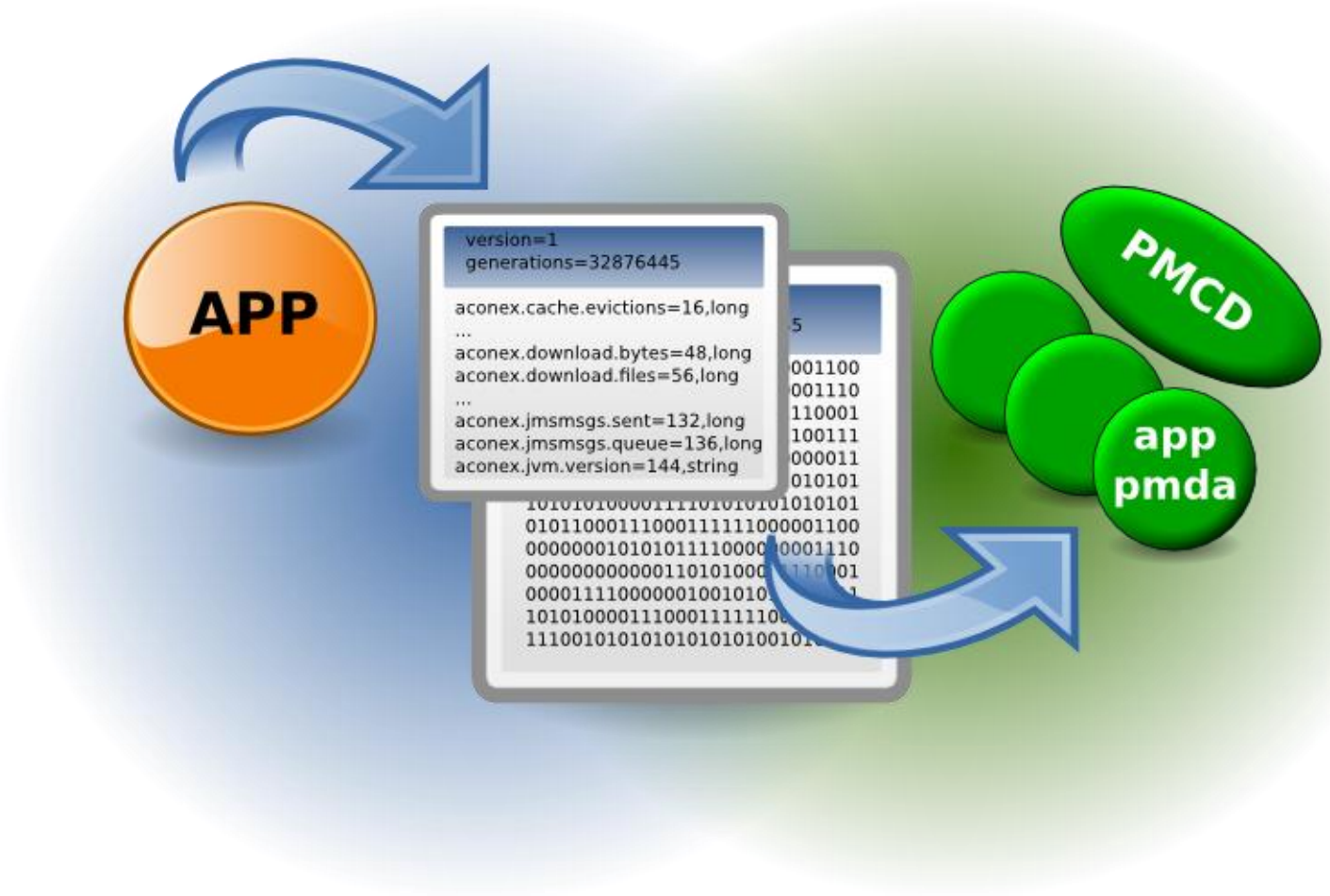
- Live Monitoring
- Modellierung und Statistische Vorhersage

Performance Timeline – PCP Toolkit

- Yesterday, last week, last month, ...
- All starts with pmlogger
  - Arbitrary metrics, intervals
  - One instance produces one PCP archive for one host
  - An archive consists of 3 files
    - Metadata, temporal index, data volume(s)
- pmlogger_daily, pmlogger_check
  - Ensure the data keeps flowing
- pmlogsummary, pmwtf, pmdumptext
- pmlogextract, pmlogreduce

CTMedia

Custom Instrumentation (Anwendungen)



PCP – Parfait

Parfait besteht aktuell aus 4 wesentlichen Teilen:

- Monitoring
- DXM
- Timing
- Requests

CATMedia

Parfait – Monitoring

- This is the ‘original’ PCP bridge metrics (heavily modified)
- Simple Java objects (MonitoredValues) which wrap a value (e.g. AtomicLong, String)
- MonitoredValues register themselves with a registry (container)

Parfait – Monitoring (2)

- When values changes, observers notice and output accordingly
  - PCP
  - JMX
  - Other (Custom/Extended)
- Very simple to use
- ‘Default registry’ (legacy concept)

Parfait – DXM

- This is the PCP output side of aconex-pcp-bridge
- Rewritten to use the new non-custom MMV PMDA
- Advantages:
  - Flexible, standardized, less maintenance work
- Disadvantages
  - Have to assign ID to each metric
- Map metrics names to ‘pseudo-PCP’ names, e.g.:
  - aconex.controllers.time.blah → aconex.controllers[mel/blah].time
- Placement of brackets is significant (determines PCP domains)

Parfait – Timing

- Logs the resources consumed by a request (an individual user action)
- Relies on a single request being thread-bound (and threads being used exclusively)
- Basically needs a Map<Thread, Value>
- Take the value for a Thread at the start, and at the end
- Delta is the ‘cost’ of that request

Parfait – Timing Beispiel

```
[2010-09-22 15:02:13,466 INFO ][ait.timing.Log4jSink][http-8080-Processor3
gedq93kl][192.168.7.132][20][] Top  taskssummaryfeatures:tasks
      taskssummaryfeatures:tasks          Elapsed time: own 380.146316 ms,
total 380.14688 msTotal CPU: own 150.0 ms, total 150.0 ms   User CPU: own
140.0 ms, total 140.0 msSystem CPU: own 10.0 ms, total 10.0 ms    Blocked
count: own 40, total 40 Blocked time: own 22 ms, total 22 msWait count:
own 2, total 2    Wait time: own 8 ms, total 8 ms     Database execution
time: own 57 ms, total 57 ms  Database execution count: own 11, total 11
      Database logical read count: own 0, total 0     Database physical
read count: own 0, total 0    Database CPU time: own 0 ms, total 0 ms
      Database received bytes: own 26188 By, total 26188 By  Database sent
bytes: own 24868 By, total 24868 By  Error Pages: own 0, total 0   Bobo
execution time: own 40.742124 ms, total 40.742124 ms   Bobo execution
count: own 2, total 2   Bytes transferred via bobo search: own 0 By, total
0 By  Super search entity count: own 0, total 0 Super search count: own 0,
total 0     Bytes transferred via super search: own 0 By, total 0 By
      Elapsed time during super search: own 0 ms, total 0 ms
```


CATMedia

Parfait – Requests

- As well as snapshotting requests after completion, for many metrics we can see meaningful ‘in-progress’ values
- Simple JMX bean which ‘walks’ in-progress requests
- Tie in with ThreadContext (MDC abstraction)
- Include UserID
- ThreadID

CATMedia

PCP – Speed

Golang Implementation des PCP Instrumentation API

Speed besteht aktuell aus 3 wesentlichen Teilen:

- Client
- Registry
- Metric

CATMedia

PCP – Speed Metric

- SingletonMetric
  - This type defines a metric with no instance domain and only one value. It requires type, semantics and unit for construction, and optionally takes a couple of description strings. A simple construction

```
metric, err := speed.NewPCPSingletonMetric(
    42,                                                          // initial value
    "simple.counter",                                            // name
    speed.Int32Type,                                             // type
    speed.CounterSemantics,                                     // semantics
    speed.OneUnit,                                              // unit
    "A Simple Metric",                                          // short description
    "This is a simple counter metric to demonstrate the speed API", // long descr
)
```


CATMedia

PCP für Container – Cgroup Accounting

- [subsys].stat files below /sys/fs/cgroup
- individual cgroup or summed over children
- **blkio**
- IOPs/bytes, service/wait time – aggregate/per-dev
- Split up by read/write, sync/async
- **cpuacct**
- Processor use per-cgroup - aggregate/per-CPU
- **memory**
- mapped anon pages, page cache, writeback, swap, active/inactive LRU state

CATMedia

PCP für Container – Namespaces

- Example: **cat /proc/net/dev**
- Contents differ inside vs outside a container

- Processes (e.g. **cat**) in containers run in different network, ipc, process, uts, mount namespaces

- Namespaces are inherited across fork/clone
- Processes within a container share common view

PCP Container Analyse – Ziele

- Allow targeting of individual containers
- e.g. ***/proc/net/dev***
- pminfo --fetch network
- vs
- pminfo –fetch –container=crank network
- Zero installation inside containers required
- Simplify your life (dev_t auto-mapping)
- Data reduction (proc.*, cgroup.*)

PCP Container Analyse – Mechanismen

- pminfo -f –host=**acme.com** –container=**crank** network
- Wire protocol extension
- Inform interested PCP collector agents
- Resolving container names, mapping names to cgroups, PIDs, etc.
- setns(2)
- Runs on the board, plenty of work remains
- New monitor tools with container awareness

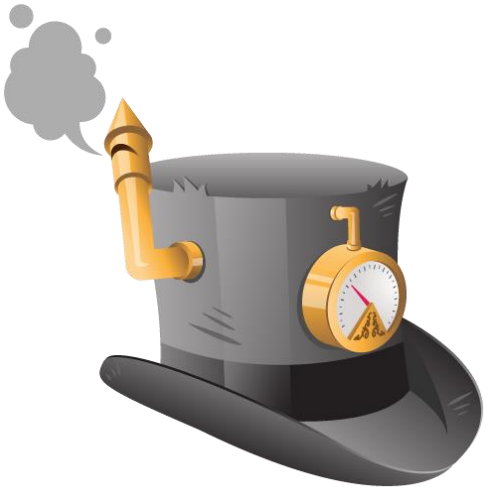
Was ist Dropwizard Metrics?

- Code instrumentation
- Meters
- Gauges
- Counters
- Histograms
- Web app instrumentation
- Web app health check

CATMedia

Metrics Reporters

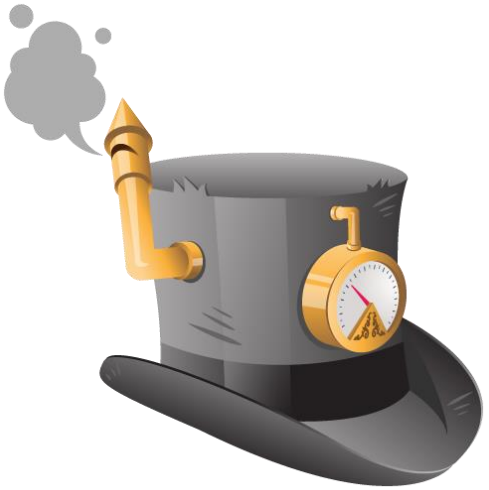
- Reporters
  - Console
  - CSV
  - Slf4J
  - JMX
- Advanced reporters
  - Graphite
  - Ganglia



CATMedia

Metrics 3rd Party Erweiterungen

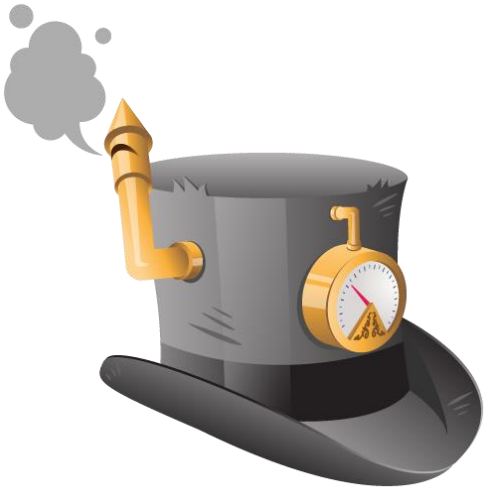
- AspectJ
- InfluxDB
- StatsD
- Cassandra
- Spring



CATMedia

Metrics Grundlagen

- MetricsRegistry
  - A collection of all the metrics for your application
  - Usually one instance per JVM
  - Use more in multi WAR deployment
- Names
  - Each metric has a unique name
  - Registry has helper methods for creating names



```
MetricRegistry.name(Queue.class, "items", "total")
//com.example.queue.items.total
MetricRegistry.name(Queue.class, "size", "byte")
//com.example.queue.size.byte
```


CATMedia

Metrics Elemente

- Gauges
  - The simplest metric type: it just returns a *value*

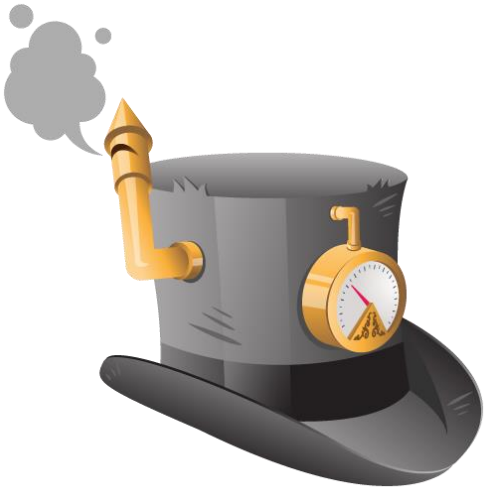
```
final Map<String, String> keys = new HashMap<>();
  registry.register(MetricRegistry.name("gauge", "keys"),
        new Gauge<Integer>() {

@Override
public Integer getValue() {
  return keys.keySet().size();
}
});
```

Metrics Elemente (2)

- Counters
  - Incrementing and decrementing 64.bit integer

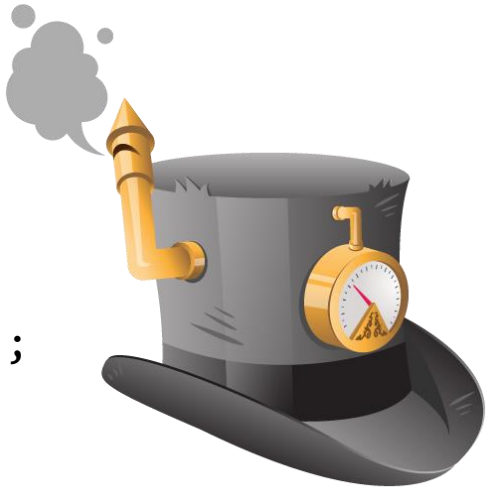
```
final Counter counter=
     registry.counter(MetricRegistry.name("counter",
        "inserted"));
counter.inc();
```



CATMedia

Metrics Elemente (3)

- Histograms
  - Measures the distribution of values in a stream of data



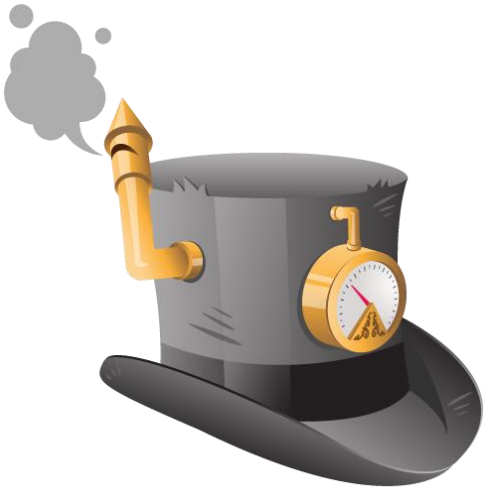
```
final Histogram resultCounts =
 registry.histogram(name(ProductDAO.class, "result-counts");
resultCounts.update(results.size());
```

- Meters
  - Measures the rate at which a set of events occur

```
final Meter meter = registry.meter(MetricRegistry.name("meter", "inserted"));
meter.mark();
```

Metrics Elemente (4)

- Timers
  - A histogram of the duration of a type of event and a meter of the rate of its occurrence



```
Timer timer = registry.timer(MetricRegistry.name("timer",
   "inserted"));

Context context = timer.time();
//timed ops
context.stop();
```

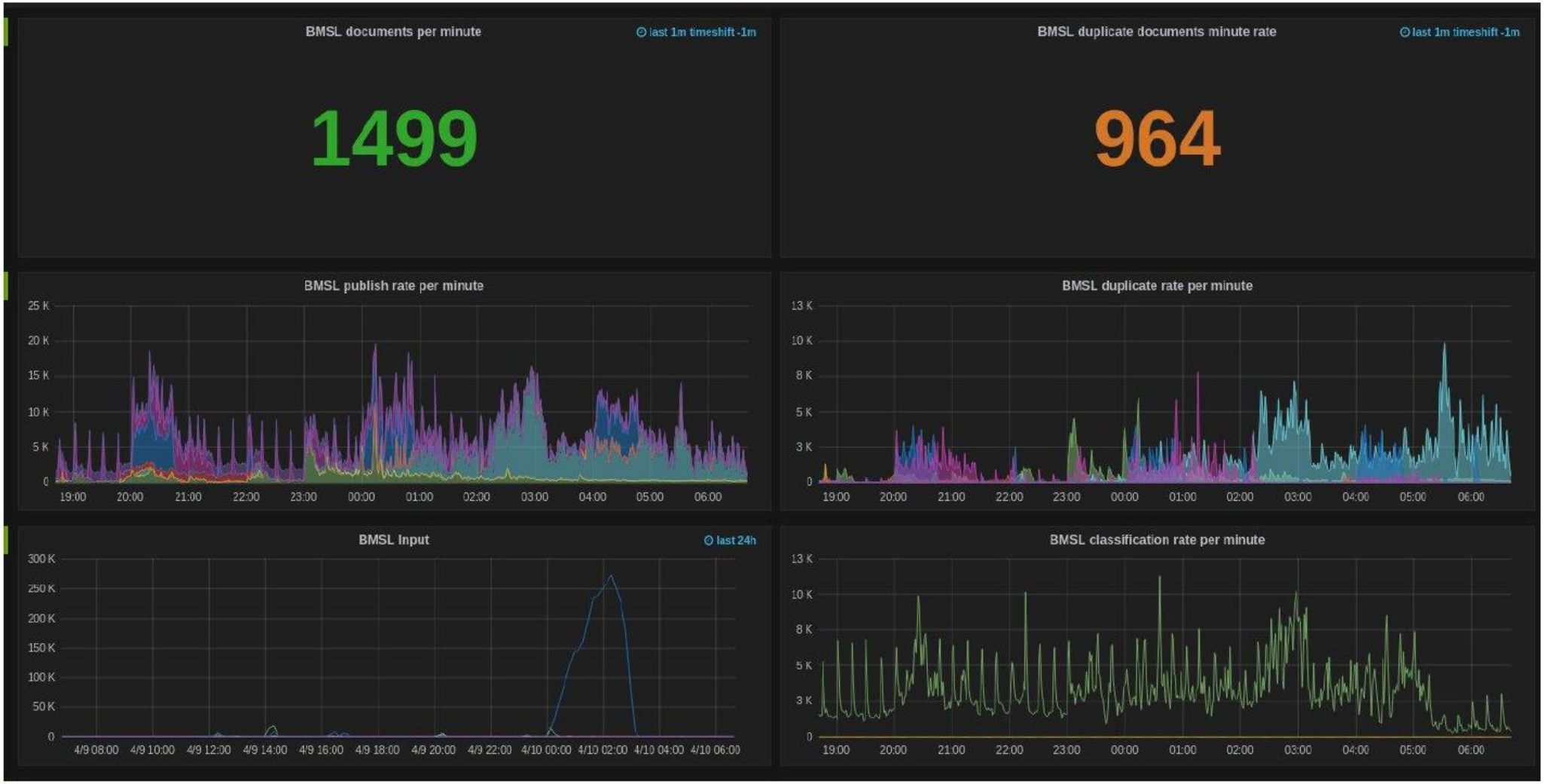
Metrics – Graphite Reporter

```
final Graphite graphite = new Graphite(new InetSocketAddress("graphite.example.com", 2003));
final GraphiteReporter reporter = GraphiteReporter.forRegistry(registry)
.prefixedWith("web1.example.com")
.convertRatesTo(TimeUnit.SECONDS)
.convertDurationsTo(TimeUnit.MILLISECONDS)
.filter(MetricFilter.ALL)
.build(graphite);
reporter.start(1, TimeUnit.MINUTES);
```

Metrics can be prefixed

Useful to divide environment metrics: prod, test

Metrics – Grafana Anwendungsübersicht



Apache Sirona – Inspiriert durch JaMon

Sirona Grundlagen

- Repository
  - The repository is a singleton for the JVM. It is the entry point to get access to counters and gauges.

```
public interface Repository extends Iterable<Counter> {
    Counter getCounter(Counter.Key key);
    void clear();
    StopWatch start(Counter counter);

    Map<Long, Double> getGaugeValues(long start, long end, Role role);
    void stopGauge(Role role);
}
```

Sirona Bestandteile

- Counter
  - A counter is a statistic and concurrency holder. It aggregates the information provided computing the average, min, max, sum of logs, ….

```
public interface Counter {
    Key getKey();
    void reset();
    void add(double delta);
    AtomicInteger currentConcurrency();
    int getMaxConcurrency();
    double getMax();
    double getMin();
    long getHits();
    double getSum();
    double getStandardDeviation();
    double getVariance();
    double getMean();
    double getSecondMoment();
```

Sirona Bestandteile (2)

- Gauge
  - A gauge is a way to get a measure. It is intended to get a history of a metric.

```
public interface Gauge {
    Role role();
    double value();
}
```

- StopWatch
  - A StopWatch is just a handler for a measure with a counter.

```
public interface StopWatch {
    long getElapsedTime();
    StopWatch stop();
}
```

Was ist Eclipse MicroProfile?

- Configuration
- Failover / Circuit Breaker
- Web app health check
- Telemetry / Monitoring
- Security (Web Token)

CATMedia

Was ist StatsD?

A network daemon that runs on the **Node.js** platform and listens for statistics, like counters and timers, sent over UDP or TCP and sends aggregates to one or more pluggable backend services (e.g., Graphite).

StatsD was inspired (heavily) by the project (of the same name) at Flickr.

Demo

CATMedia

Q & A

CATMedia

Links

Performance Co-Pilot
http://www.pcp.io

Dropwizard Metrics
http://metrics.dropwizard.io

MicroProfile Health

https://github.com/eclipse/microprofile-health

Apache Sirona
http://sirona.apache.org/

StatsD
https://github.com/etsy/statsd/wiki

CATMedia