

A Working Theory-of-Monitoring

LISA 2013

Caskey L. Dickson (caskey@google.com)
Site Reliability Engineer, Google, Inc.

Metrics

"the assignment of numerals to things so as to represent facts and conventions about them"

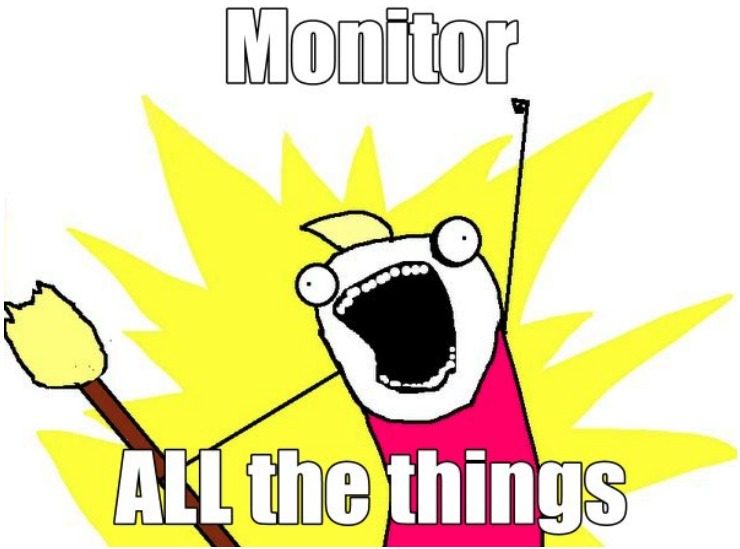
– S. S. Stevens 1946

Why a “theory”

Monitoring seems easy

It’s not.

Why?



If successful, we should be able to sensibly map many monitoring methods/modes into a good model with fidelity.

What do we monitor? (What's a metric?)

Named value at some time.

- Metric identity/name
  k-tuple within an identity space attached to each value
  <www-1.na-east.example.com, httpd(3321), foo.example.com, 200-ok-count>
  <*hostname*, *process*, *vhost*, *name*>
- Metric values (overlapping)
  Counters, Gauges, Percentiles… Nominal, Ordinal, Interval, Ratio… Derived…
- Timestamped
  httpd:www-1.na-east.example.com:200-ok-count@20131005T142155.867Z = 8505936

How can we monitor? (What's a metric?)

Resolution

How frequently are you reading a metric?
Every 6 seconds? 6 Minutes?

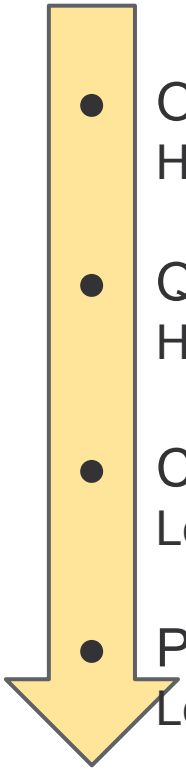
Latency

After reading, how long before we act on them?
Seconds, Minutes, Hours?

Diversity

Are you collecting many *different* metrics?
10, 25, 50, 100, 10K, 10M?

Why do we monitor?



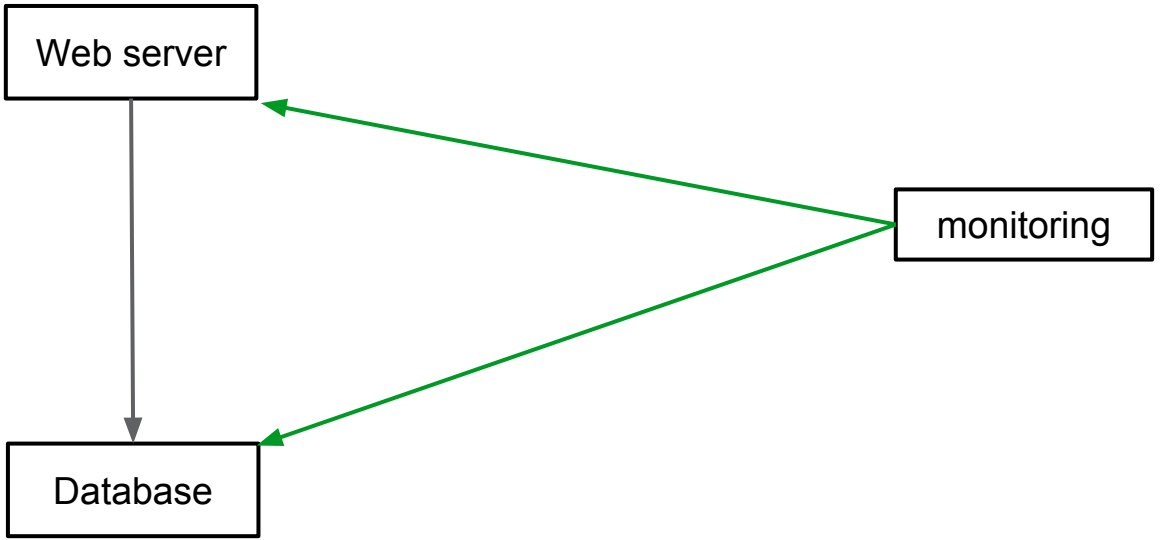
- Operational Health/Response (R+,L+,D+)
  High Resolution, Low Latency, High Diversity
- Quality Assurance/SLA (R+,L-,D+)
  High Resolution, High Latency, High Diversity
- Capacity Planning (R-,L-,D+)
  Low Resolution, High Latency, High Diversity
- Product Management (R-,L-,D-)
  Low Resolution, High Latency, Low Diversity

What about these?

- (R-,L+,D-) LB
- (R-,L+,D+)
- (R+,L+,D-)
- (R+,L-,D-)

Monitoring at scale

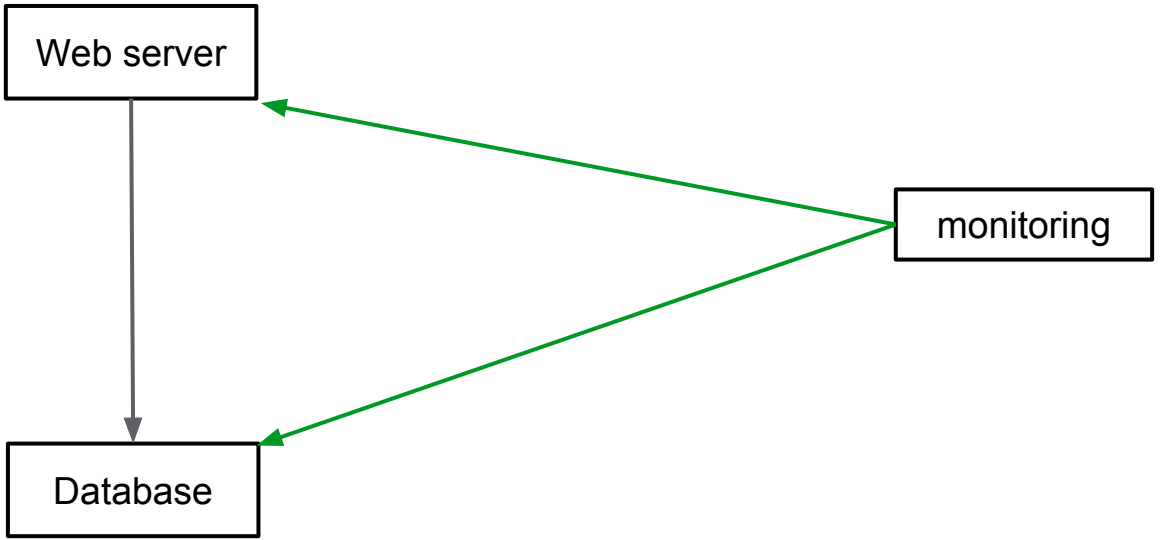
~ 25 metrics/server
= 50 metrics



0.16 metrics/second

Monitoring at scale

~ 25 metrics/server
= 50 metrics
100M active daily users
= 200K peak QPS

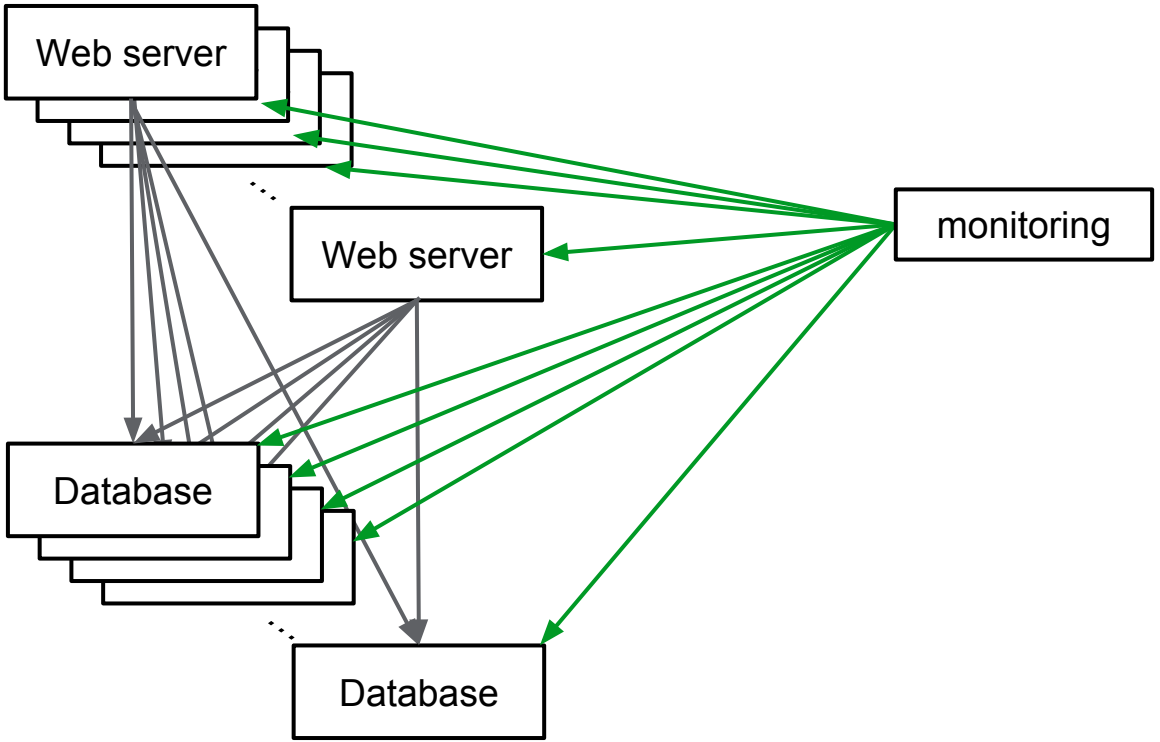


0.16 metrics/second

Monitoring at scale

~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= 25,000 metrics

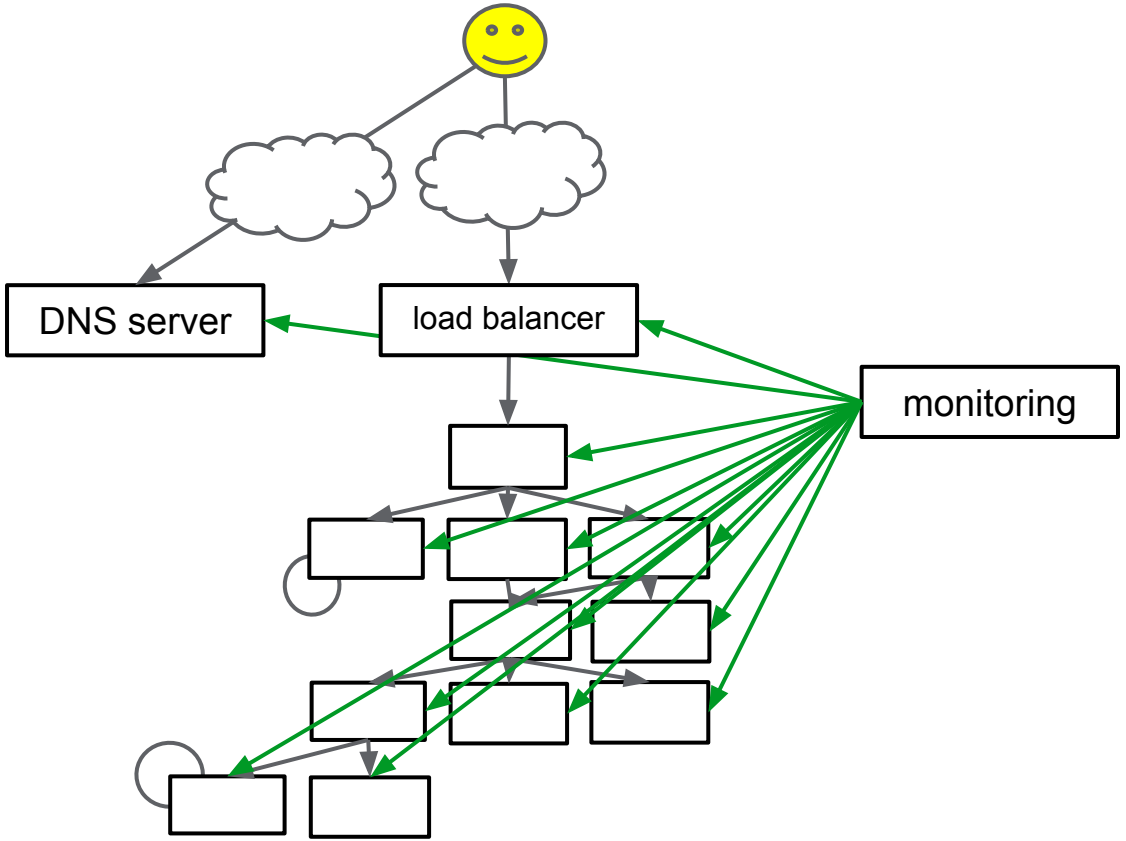
~166 metrics/second



Monitoring at scale

~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= ~~25,000 metrics~~

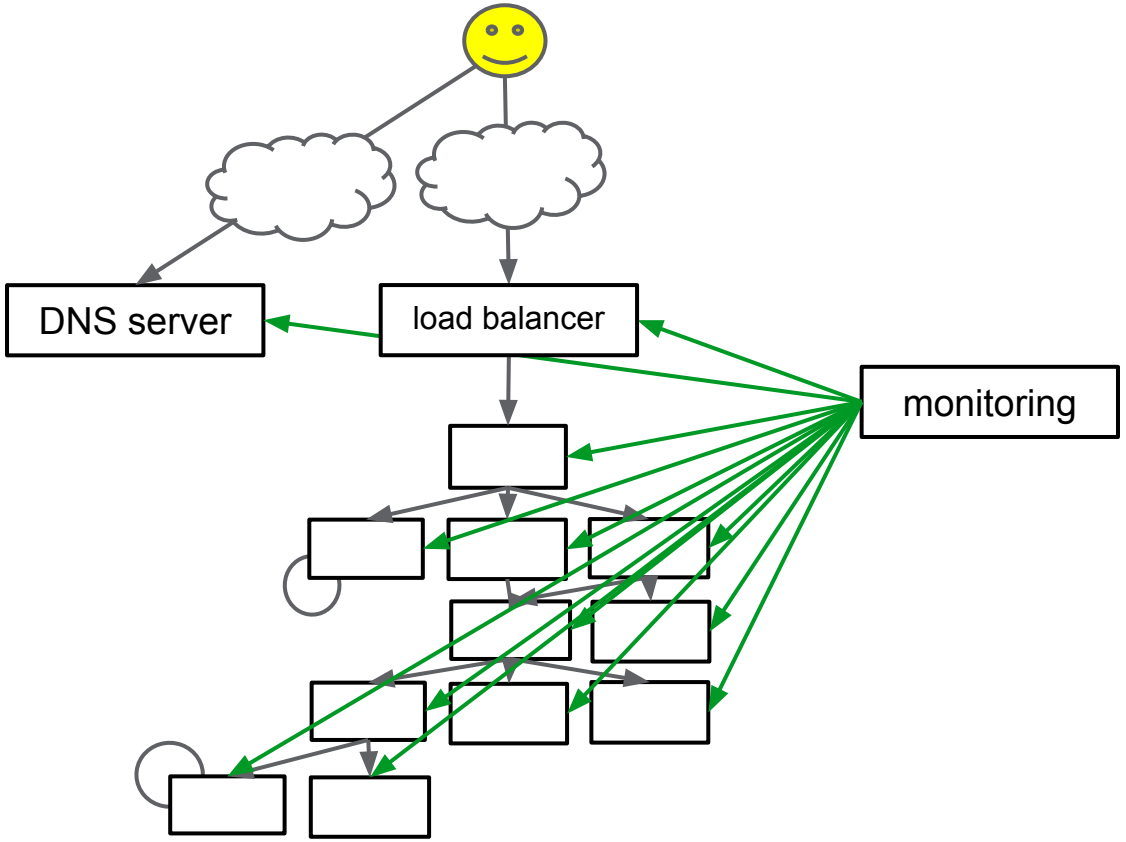
~166 metrics/second



Monitoring at scale

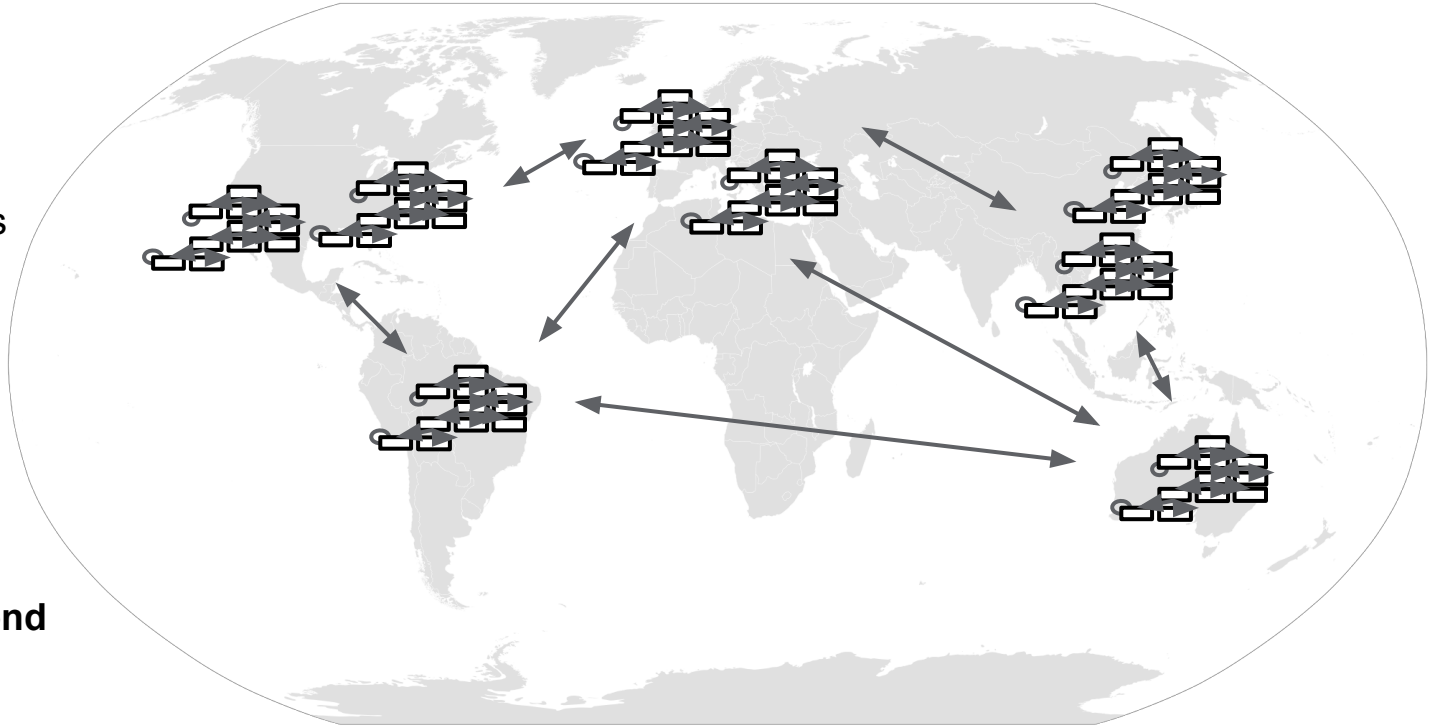
~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= ~~25,000 metrics~~
X 12 ‘types’ of servers
= 3,000,000 metrics

10,000 metrics/second



Monitoring at scale

~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= ~~25,000 metrics~~
X 12 ‘types’ of servers
= ~~3,000,000 metrics~~
X 8/6 sites (N+2)
= 4,000,000 metrics
13,333 metrics/second



Monitoring at scale

~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= ~~25,000 metrics~~
X 12 ‘types’ of servers
= ~~3,000,000 metrics~~
X 8/6 sites (N+2)
= 4,000,000 metrics
13,333 metrics/second

O(**10K**) metrics/second
O(**32MB**) data / sweep



Monitoring at scale

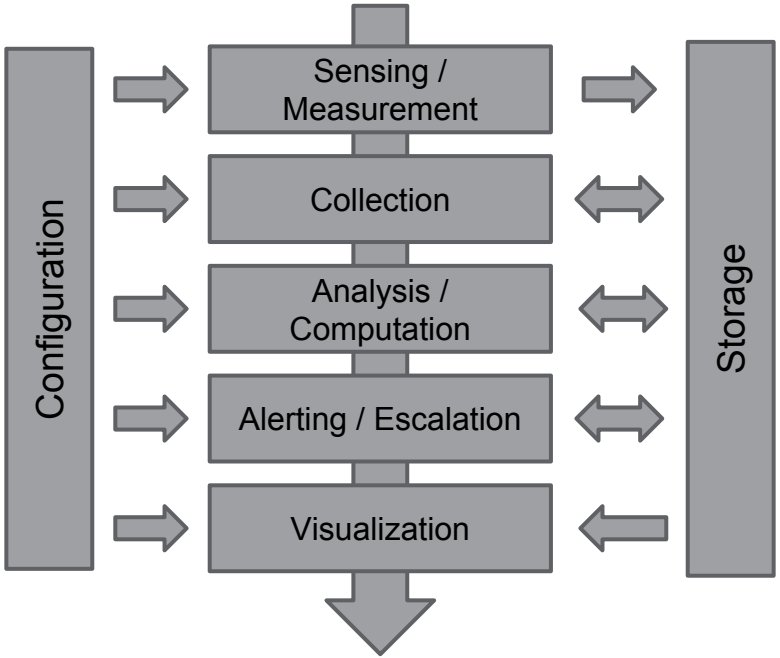
~ 25 metrics/server
= ~~50 metrics~~
100M active daily users
= 200K peak QPS
@ 20QPS/server
= 10,000 servers
= ~~25,000 metrics~~
X 12 ‘types’ of servers
= ~~3,000,000 metrics~~
X 8/6 sites (N+2)
= 4,000,000 metrics
13,333 metrics/second

O(**10K**) metrics/second
O(**32MB**)/sweep

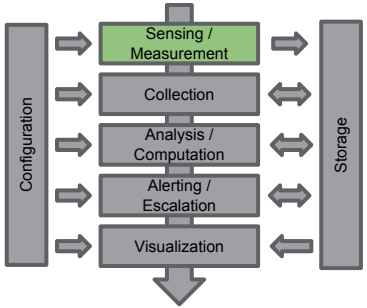
Ops @ 1 minute
=
O(50K) metrics/second
O(320MB)/sweep
O(460GB)/24 hours

What do we monitor? (recap)

- Named, timestamped values of differing types
- Gathered at high resolution
- Large quantities
- Many different consumers (downsampling, filtering, aggregation)
- Reliably


Configuration
Sensing / Measurement
Collection
Analysis / Computation
Alerting / Escalation
Visualization
Storage

Sensing / Measurement



The creation of metrics at some minimum level of abstraction. Generally raw counters plus some attributes.

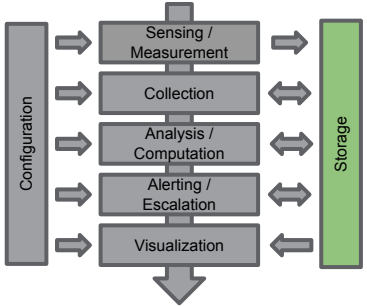
Different systems gather data at different speeds. top/ps/netstat are very immediate, sar somewhat less so, nagios much less so.

Different systems have different concepts of an individual unit for metric identity

No consistent interface



Storage



Placing of time series in a (readily?) accessible format
Raw, aggregated and post-computation metrics

Occurs in different formats at different stages

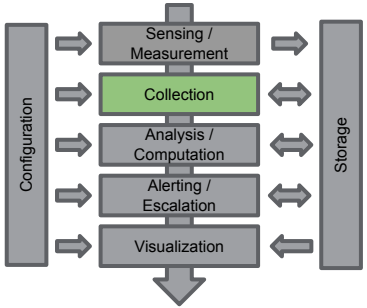
/var/log/syslog, /var/log/apache/access_log, /var/www/mrtg/*, /var/lib/rrdb/*.rrd, mysql/postgresql

I/O throughput

Structure limits analysis/visualization options



Collection



Bringing together many individual metrics in one place to support analysis.

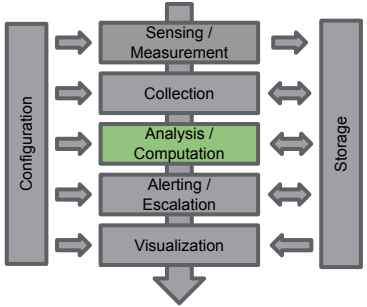
Metric identity needs to remain meaningful after aggregation.

Key for scalability

Many transports, smart and dumb.
multicast, TCP, rrdcached, SFTP, rsync



Analysis



Extraction of *meaning* from the raw data.

Often focused upon finding and detecting features or anomalies.

Some anomalies are important, others are...
merely interesting.

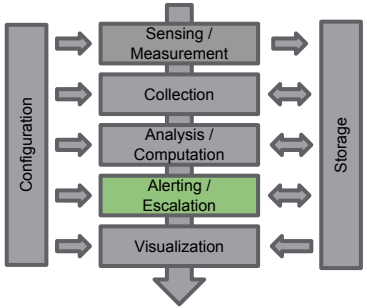


CPU constrained for throughput/depth

Lots of interesting research in autocorrelation

RAM constrained for metric volume

Alerting & Escalation



When anomalies are detected, something has to deal with promulgation of those conditions to interested parties. Some anomalies are urgent (short-term SLO critical) others are merely important.

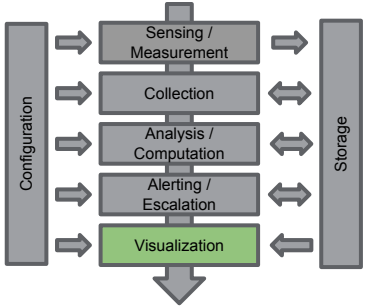


“Urgent” anomalies reflect conditions that without immediate operator intervention will lead to an outage or SLO excursion. Something is responsible for being noisy until someone comes to help.

Ideally this happens as infrequently as possible.

Google

Visualization

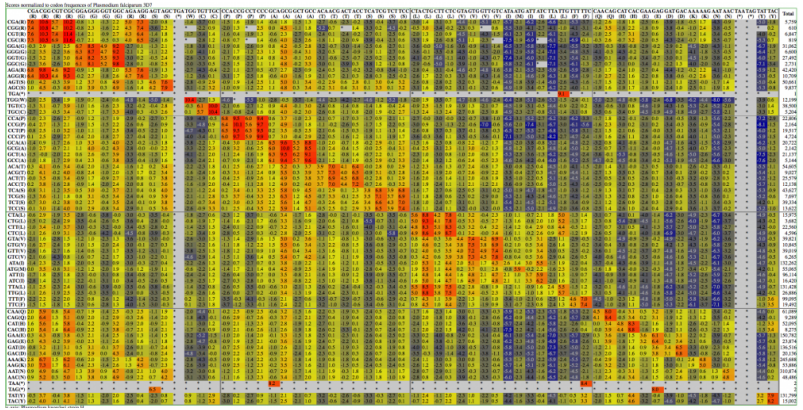


Meaningful visualization of the raw data can be the difference between staying within or exceeding your SLO.

Viewing more than 3 dimensions can be problematic for those of us who are still human.

Goal-oriented

Read and *apply* your Tufte/Few

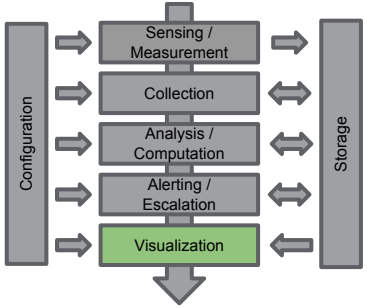


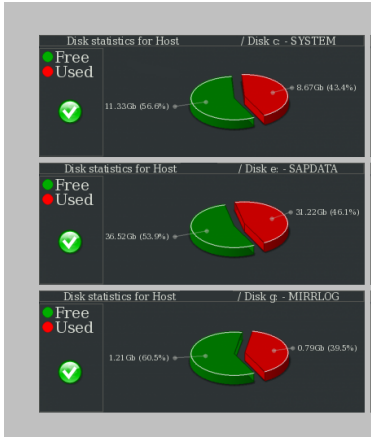
Google

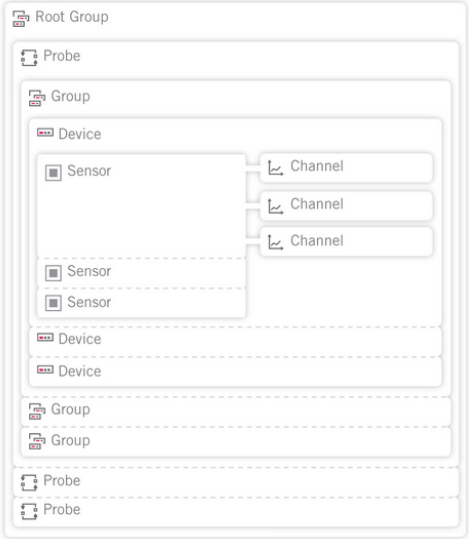
Visualization and Actionability

Some visualizations are less than useful.
Disk space is a commonly graphed metric which is un-actionable without derivatives.

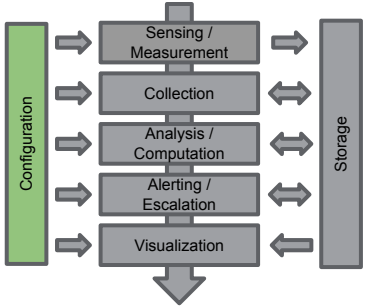
Not all views have the same taxonomy.







Configuration



Affects every layer

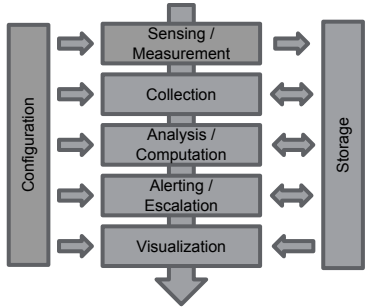
Needs configuration management

Complicates distributed systems

Limits change velocity

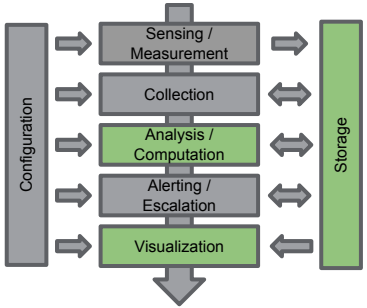


Why do we monitor? (repeat)



- Operational Health/Response (R+,L+,D+)
  High Resolution, Low Latency, High Diversity
- Quality Assurance/SLA (R+,L-,D+)
  High Resolution, High Latency, High Diversity
- Capacity Planning (R-,L-,D+)
  Low Resolution, High Latency, High Diversity
- Product Management (R-,L-,D-)
  Low Resolution, High Latency, Low Diversity

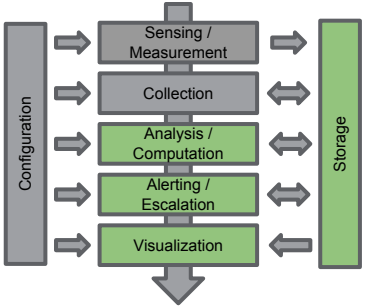
Product Management (R-,L-,D-)



Mostly synthesized/reprocessed metrics (KPIs vs. SLIs)

- Lots of historic data in **storage** for long-term views
- **Analysis** of synthesized metrics from concrete metrics
  - 7-day actives
  - Conversion rates
- Easy to understand **visualizations** of resulting metrics

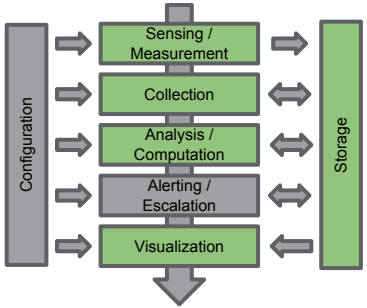
Capacity Planning (R-,L-,D+)



Evaluation of current serving capacity

- Calculation of proxy metrics
- Impact of changes to serving capacity
- Cost per user
- Efficiency
- Alerting when capacity limits approaching

Quality Assurance/SLA (R+,L-,D+)

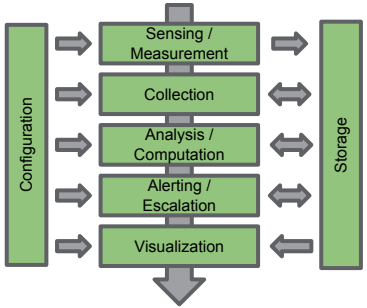


- Includes developer support
- Collect data from both narrow and wide views

  (**Sensing** high resolution process behavior and system-metrics)
- Offline and real-time performance **analysis**, tracing

  (**Collection** and **storage** of data from diverse runs)
- Not necessarily real-time
- Useful **visualizations** to aid understanding

Operational Health/Response (R+,L+,D+)



The hardest use case

- Immediate, up to date metrics (low latency **collection**)
- Encompassing the entire fleet (broad **collection** coverage, many sensors incorporated)
- Real-time computation of thresholds and alerts (high speed **analysis**)
- Reliable and flexible **alerting**
- **Storage** of enough timeseries at high enough resolution for comparison (XXXGB/day * 730 days)
- Simple **configuration** of global monitoring perspective

A moment please...

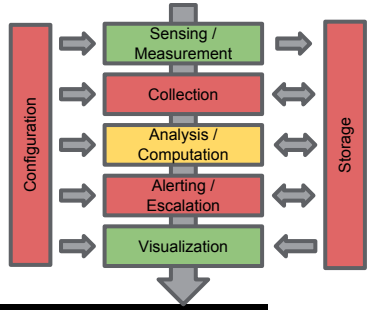
All the systems to be discussed have inherent, undeniable value and I have personally used and benefited from them and mean no disrespect to the implementers and maintainers of them.

Personally I use these systems, in the past I have relied upon them for production services I was responsible for.

This is NOT a criticism of those products, rather an indication of where they stop short of one particular hypothetical ideal.

Google

/bin/top (host + process health)



Sensing: /proc, /sys, syscalls(1)
Collection: while(true);
Analysis: Summing and sorting
Alerting: Sort to top
Visualization: ordered lists, dynamic sorting
Storage: none
Configuration: runtime shortcuts

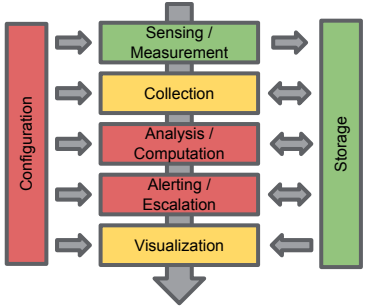
```
top - 18:54:30 up 67 days,  3:05,  2 users,  load average: 1.60, 1.03, 0.48
Tasks: 113 total,   1 running, 112 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.7%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.7%us,  1.3%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  2.7%us,  7.7%sy,  0.0%ni,  5.0%id, 84.6%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  1.3%sy,  0.0%ni, 97.3%id,  1.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    503132k total,   496152k used,     6980k free,    41340k buffers
Swap:        0k total,        0k used,        0k free,   195772k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
15070 httpd     20   0  105m  87m 1144 D    8 17.8   0:06.18 httpd
 1032 mediatom  20   0 1160m  47m  816 S    0  9.6  38:51.31 mediatomb
 6521 root      20   0 83476  46m  36m S    0  9.5   0:04.65 apt-get
 6643 caskey    20   0 26840 8000 1616 S    0  1.6   0:00.54 bash
 6236 root      20   0  107m 4236 3156 S    0  0.8   0:00.06 sshd
  456 syslog    20   0  244m 3280  460 S    0  0.7   4:46.73 rsyslogd
 1303 root      20   0  743m 3080  284 S    0  0.6  14:36.75 ushare
31304 root      20   0 2042m 2584 1492 S    0  0.5   0:00.14 console-kit-dae
    1 root      20   0 24432 1768  696 S    0  0.4   0:22.14 init
```

Google

/bin/sar (host health)

Basically some of top + timeseries




```
Linux 2.6.18-194.el5PAE (dev-db)        03/26/2011      _i686_  (8 CPU)

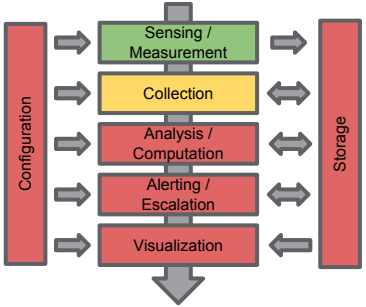
01:27:32 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
01:27:33 PM       all      0.00      0.00      0.00      0.00      0.00    100.00
01:27:34 PM       all      0.25      0.00      0.25      0.00      0.00     99.50
01:27:35 PM       all      0.75      0.00      0.25      0.00      0.00     99.00
Average:          all      0.33      0.00      0.17      0.00      0.00     99.50

Linux 2.6.18-194.el5PAE (dev-db)        03/26/2011      _i686_  (8 CPU)

07:28:06 AM kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact
07:28:07 AM   6209248   2097432     25.25    189024   1796544    141372      0.85   1921060     88204
07:28:08 AM   6209248   2097432     25.25    189024   1796544    141372      0.85   1921060     88204
07:28:09 AM   6209248   2097432     25.25    189024   1796544    141372      0.85   1921060     88204
Average:      6209248   2097432     25.25    189024   1796544    141372      0.85
```

Google

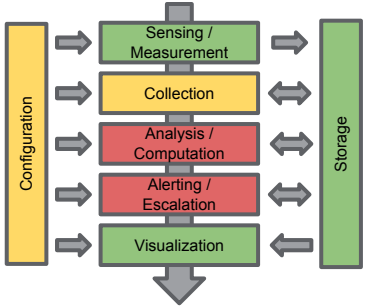
*trace (process behavior)



Sensing: dtrace/strace/ltrace process wrapper
Collection: single instance
Analysis: None
Alerting: N/A
Visualization:
None
Storage: None
Configuration:
command line

```
[pid 11783] __libc_start_main(0x407420, 1, 0x7fff75b6aad8, 0x443cc0, 0x443d50 <unfinished ...>
[pid 11783] geteuid()                           = 1000
[pid 11783] getegid()                           = 1000
[pid 11783] getuid()                            = 1000
[pid 11783] getgid()                            = 1000
[pid 11783] setuid(1000)                        = 0
[pid 11783] malloc(91)                          = 0x00cf8010
[pid 11783] XtSetLanguageProc(0, 0, 0, 0x7f968c9a3740, 1) = 0x7f968bc16220
[pid 11783] ioctl(0, 21505, 0x7fff75b6a960)     = 0
[pid 11783] XtSetErrorHandler(0x42bbb0, 0x44f99c, 0x669f80, 146, 0x7fff75b6a72c) = 0
[pid 11783] XtOpenApplication(0x670260, 0x44f99c, 0x669f80, 146, 0x7fff75b6a72c) = 0xd219a0
[pid 11783] IceAddConnectionWatch(0x42adc0, 0, 0, 0x7f968c9a3748, 0 <unfinished ...>
[pid 11783] IceConnectionNumber(0xd17ec0, 0, 1, 0xcfb138, 0xd17c00) = 4
[pid 11783] <... IceAddConnectionWatch resumed> ) = 1
[pid 11783] XtSetErrorHandler(0, 0, 1, 0xcfb138, 0xd17c00) = 0
[pid 11783] XtGetApplicationResources(0xd219a0, 0x6701c0, 0x66b220, 34, 0) = 0
[pid 11783] strlen("off")                       = 3
```

mrtg



Sensing:

SNMP, subprocess, 2 metrics max

Collection:

Centralized scraping over SMTP

Local processes

Analysis:

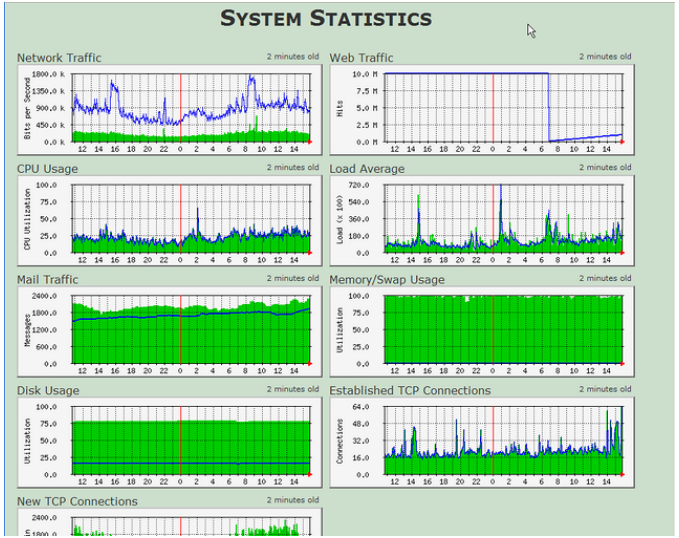
Basic math

Alerting:

None

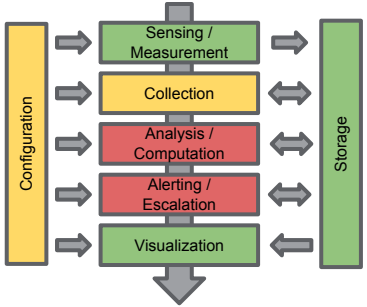
Visualization:

day/week/month/year graphs 2 variables



Google

mrtg



Operations:

Ideal for netops, no alerting though

Product Management:

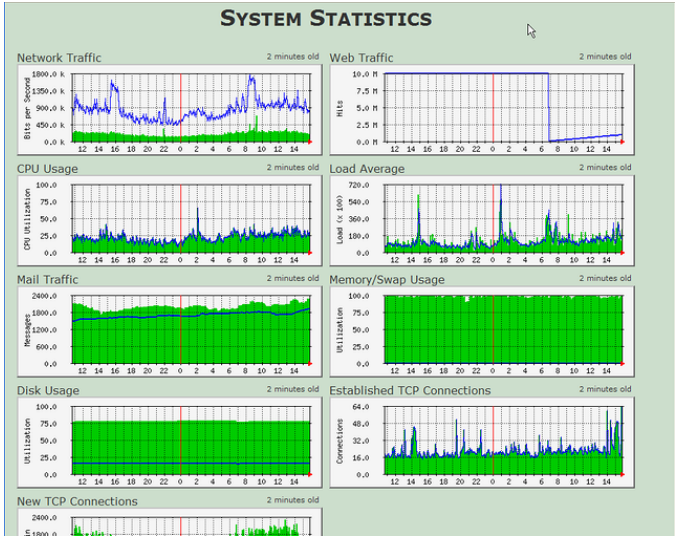
None

Capacity Planning:

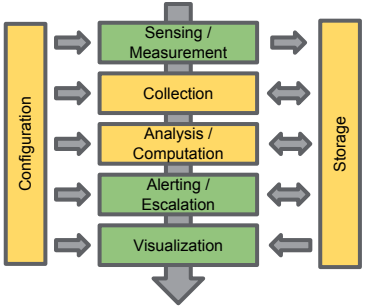
Ideal for network ops and host health

Q/A, SLA:

None



Nagios



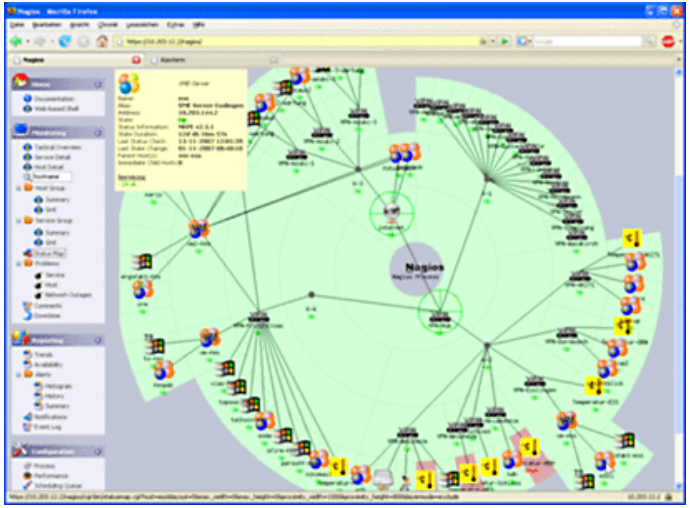
Sensing:
- Subprocesses and plugins, LOTS of plugins

Collection:
- Centralized scraping
- Support for forwarding metrics

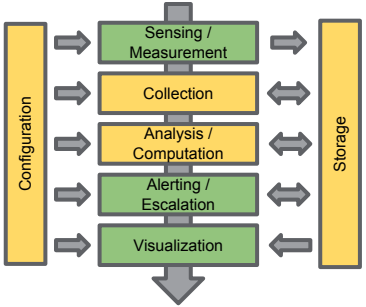
Analysis: At sensing time

Alerting:
- Configurable alarms and emails

Visualization:
- Basic graphs of check results
- Dependency chains



Nagios



Operations:

Good for simple operations, basic alert support

Redundant (N+M) configurations more difficult

Product Management:

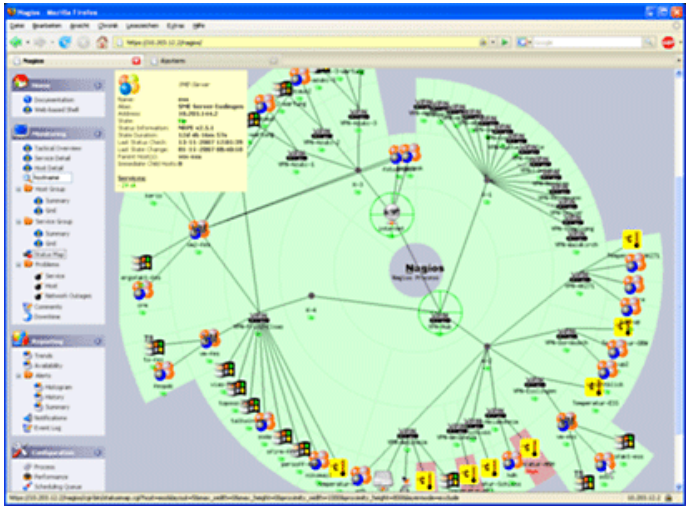
N/A, heavily focused on up/down checks

Capacity Planning:

N/A

Q/A, SLA:

N/A, poor/no timeseries visualization



Google

Ganglia

Sensing:
- gmond on nodes
- extensions/plugins

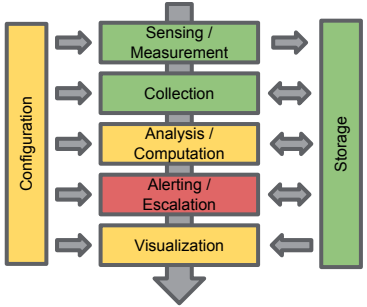
Collection:
- multicast, UDP, TCP polls

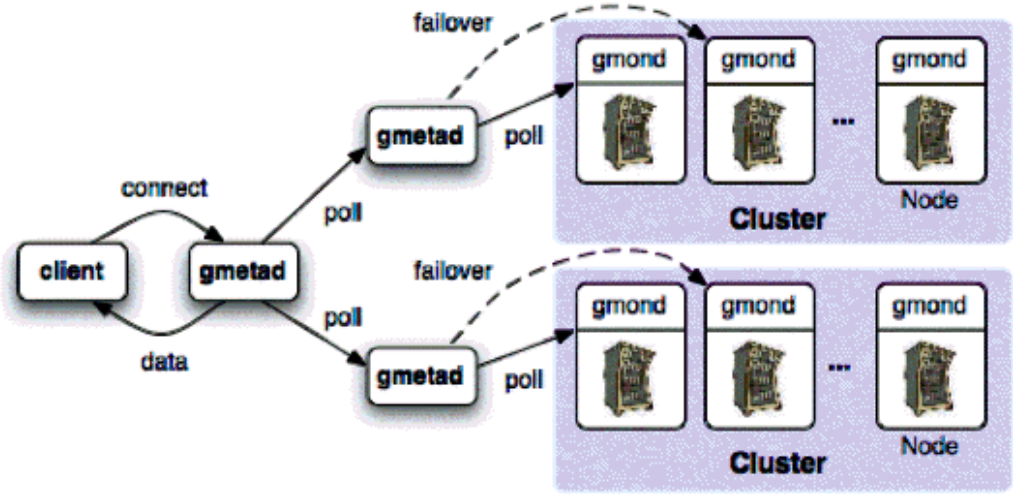
Analysis:
- value_threshold
- external (nagios)

Storage: rrdtool/rrdcached

Alerting: N/A

Visualization: ganglia-web





Google

Ganglia

Operations:

Unsuited, no alerting built in

Can feed nagios/other

Product Management:

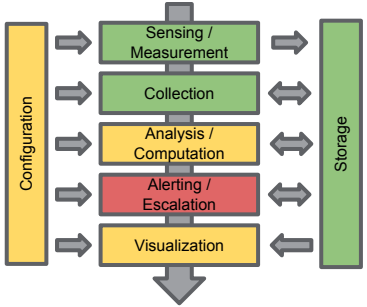
Cluster ops focus

Capacity Planning:

Well suited

Q/A, SLA:

Historic views

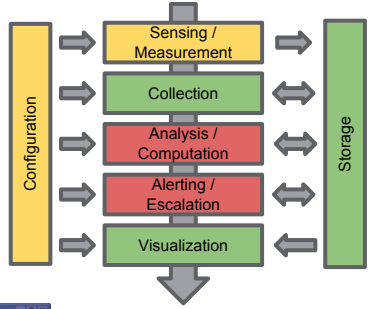


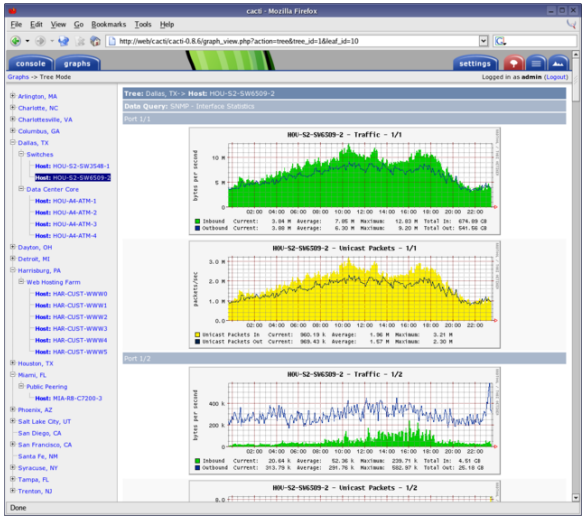


Google

Cacti (MRTG++)

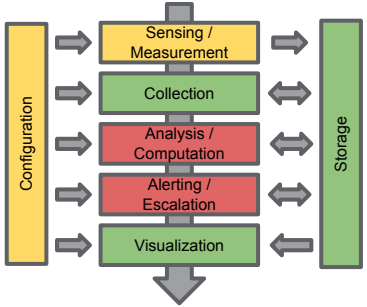
Sensing:
    Poller, cron based
Collection:
    Primarily SNMP
Analysis:
    Basic summing
Storage: rrdtool, MySQL
Alerting: N/A
Visualization:
    Static graphs





Google

Cacti (MRTG++)



Operations:

No alerts limits utility to diagnostics

Product Management:

Well suited

Capacity Planning:

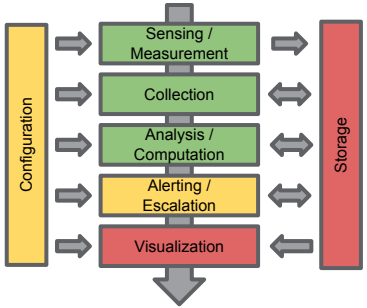
Well suited

Q/A, SLA:

Well suited



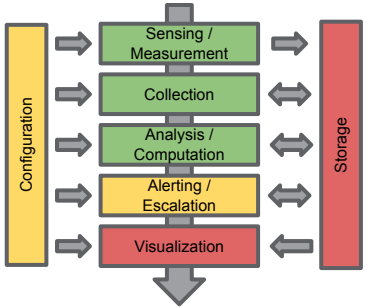
Sensu



Sensing: Arbitrary JSON emitters “Checkers”
Collection: RabbitMQ JSON event bus
Analysis:
    Handlers
Storage: N/A
Alerting:
    Handlers
Visualization: N/A



Sensu



Operations:

Configurable collection layer, handlers and checkers

Product Management:

N/A

Capacity Planning:

N/A

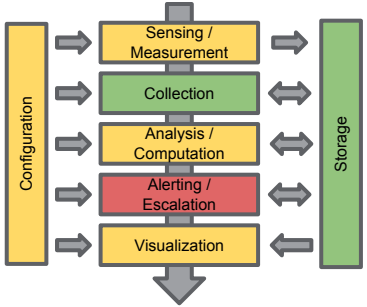
Q/A, SLA:

Can feed live data to other technologies

sensu

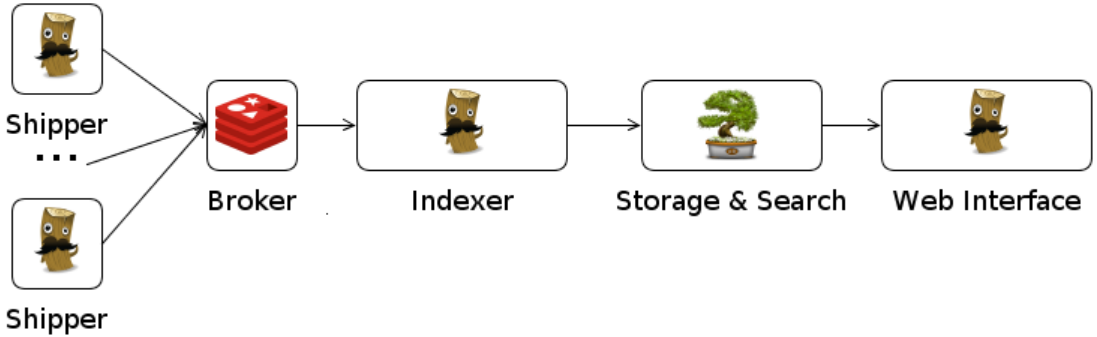
Google

Logstash

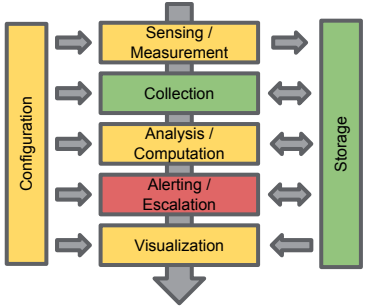


Sensing:
Deployable log thrower
Collection:
MQ (Redis)
Analysis:
Indexer
Storage:
ElasticSearch
Alerting: N/A
Visualization:
Kibana (ES)





Logstash



Operations:

Historical view of systems, searching for incident info

Product Management:

N/A

Capacity Planning:

N/A

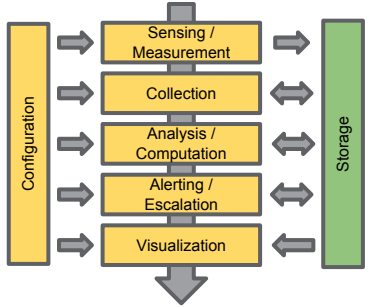
Q/A, SLA:

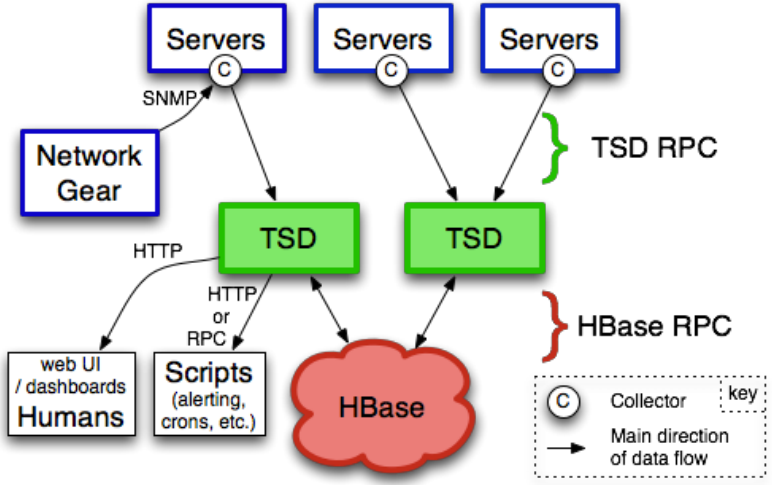
Tracing of individual problem cases,
cross correlation among different log sets



OpenTSDB

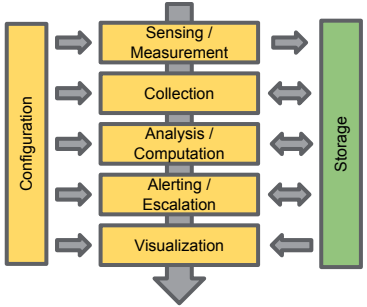
Sensing:
    Custom clients
Collection:
    TSD RPC
Analysis:
    External
Storage:
    Complete storage layer
Alerting: N/A
Visualization: N/A





Google

OpenTSDB



Operations:

Can handle the volume

Product Management:

N/A

Capacity Planning:

N/A

Q/A, SLA:

N/A

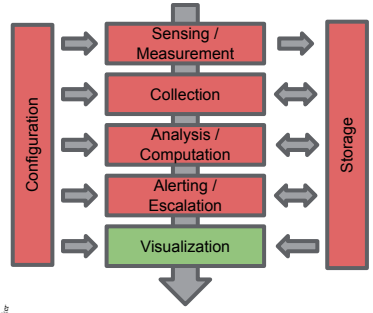
```
mysql.bytes_received 1287333217  327810227706 schema=foo host=db1
mysql.bytes_sent     1287333217 6604859181710 schema=foo host=db1
mysql.bytes_received 1287333232  327812421706 schema=foo host=db1
mysql.bytes_sent     1287333232 6604901075387 schema=foo host=db1
mysql.bytes_received 1287333321  340899533915 schema=foo host=db2
mysql.bytes_sent     1287333321 5506469130707 schema=foo host=db2
```

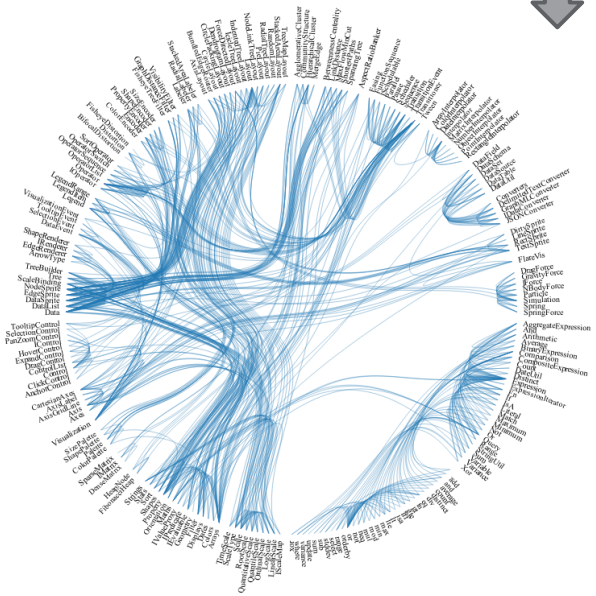
Google

D3.js

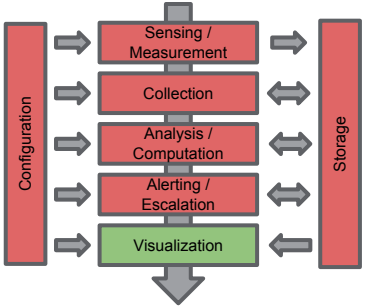
Sensing: N/A
Collection: N/A
Analysis: N/A
Storage: N/A
Alerting: N/A
Visualization:

Very nice *interactive* charts of prepared data sets





D3.js



Operations:

Data exploration of limited value

Product Management:

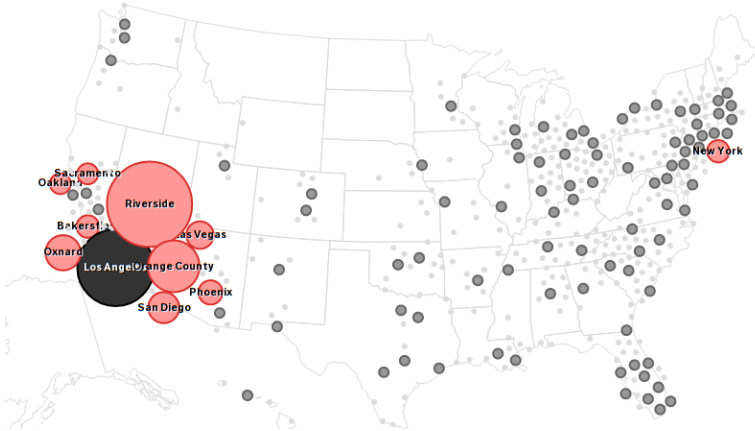
Good discovery and goal seeking

Capacity Planning:

Interactive searching for hidden dependencies

Q/A, SLA:

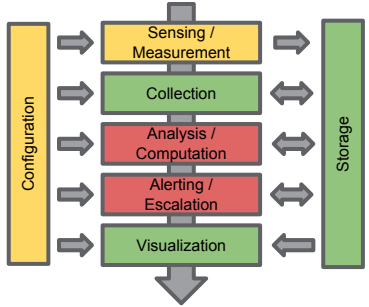
Great potential for exploring traces and dependencies

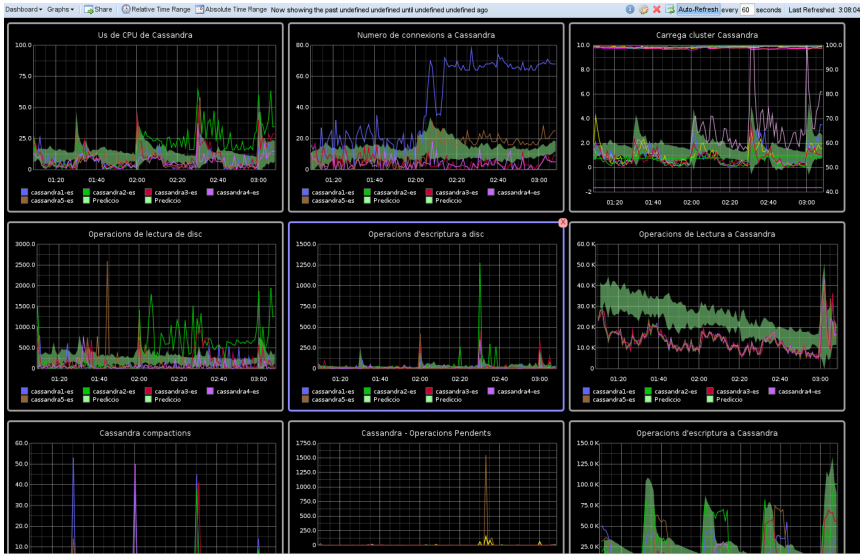


Google

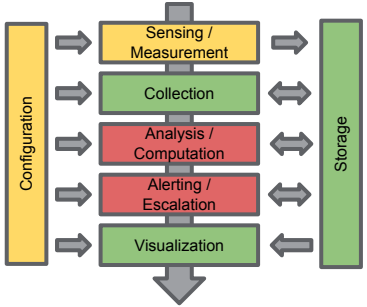
Graphite

Sensing:
DIY, name+value
Collection:
Custom messaging protocol
Analysis: N/A
Storage: Carbon+Whisper
file-per-metric
Alerting: N/A
Visualization:
Static config of complex graphs





Graphite



Operations:

Command-line graph creation, limited interactive web

Product Management:

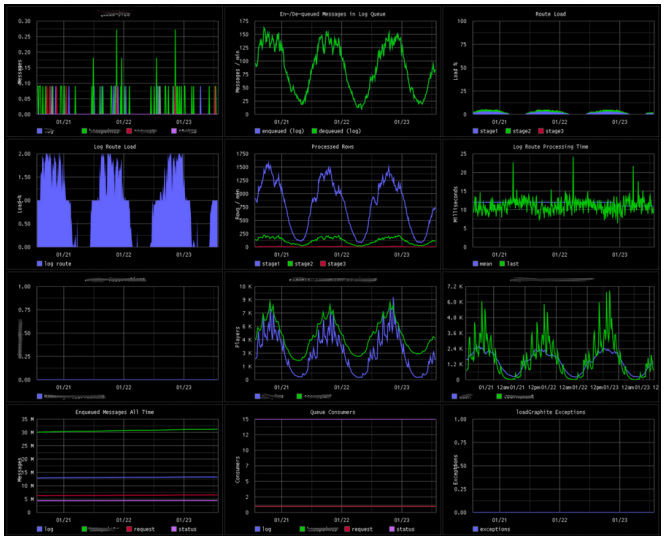
Great for visualization

Capacity Planning:

Also good for visualization

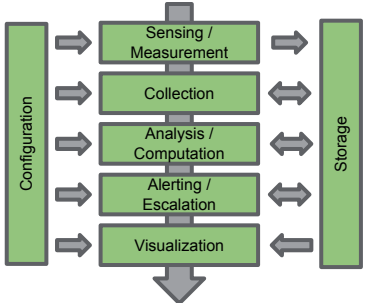
Q/A, SLA:

Can visualize, but lacks interactivity



Google

Shinken (Nagios + Graphite + CM)



Sensing:
- Nagios plugins/Receiver

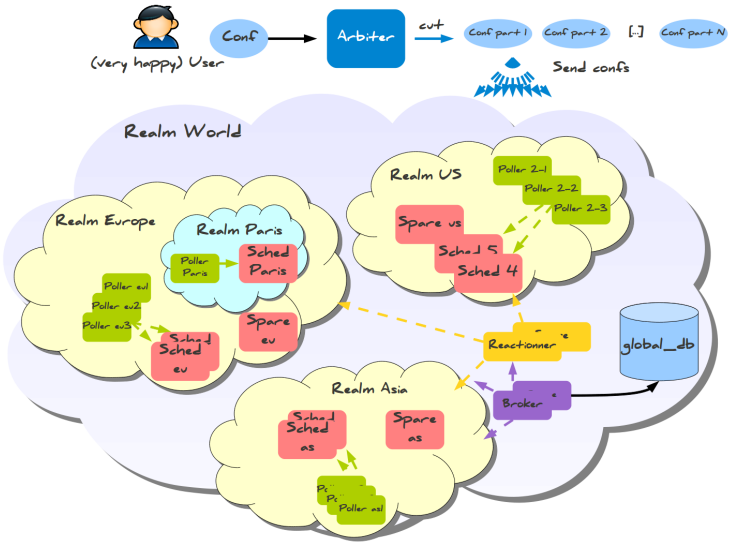
Collection:
- Scheduler/Poller/Receiver

Analysis:
- Reactioner/Broker

Storage: RRDtool

Alerting:
- Reactioner

Visualization:
- Sadly not much better than Nagios



Google

Shinken

Operations:

Much better CM than Nagios

Product Management:

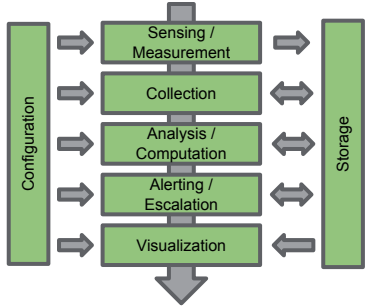
N/A

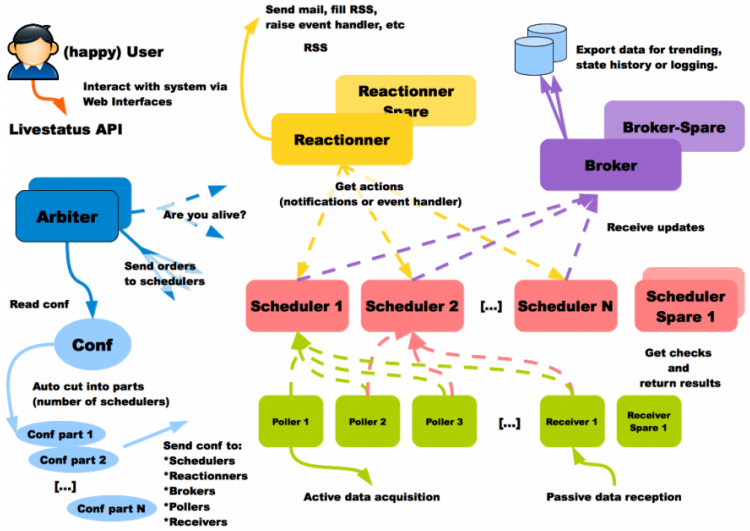
Capacity Planning:

N/A

Q/A, SLA:

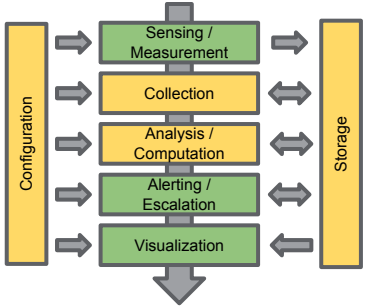
N/A





Google

“Cloud Monitoring”



Lots and lots of vendors

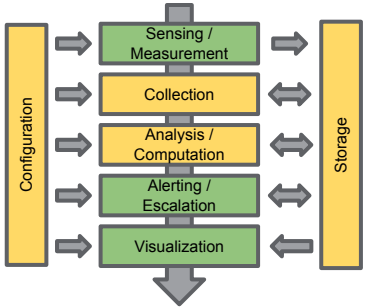
AlertSite, Bijk, CopperEgg, Dotcom Monitor, GFI Cloud, Kaseya, LogicMonitor, Monitis, MonitorGrid, Nimsoft, ManageEngine, Panopta, Pingdom, Scout, ServerDensity, Shalb SPAE, CloudTest, ...

SaaS offerings

Remote collection, local agents, push and pull

Implementation black boxes

In the Real World™



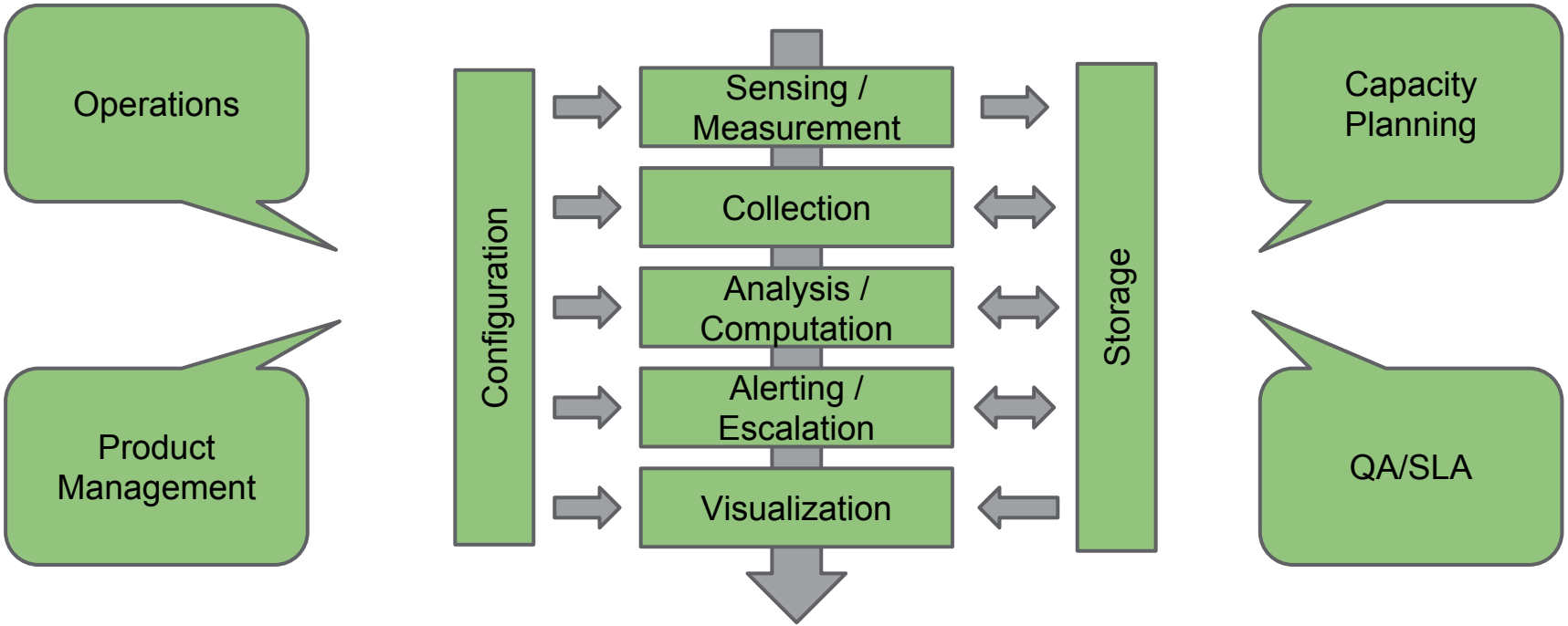
All of the above.

Nagios + Graphite + Sensu + Logstash + Ganglia

Interoperability is limited at the interface layer.

MQ based solutions are promising glue.

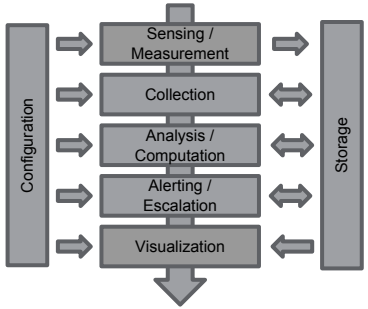
Interactive graphs are inspiring.


Operations
Product Management
Configuration
Sensing / Measurement
Collection
Analysis / Computation
Alerting / Escalation
Visualization
Storage
Capacity Planning
QA/SLA

Google

Thanks!

Criticisms
Questions
Comments
Feedback
Hate Mail



caskey@{google,gmail,twitter,sf,github,...}

Join us! go to google.com/jobs

Appendix

Extra stuff, just in case.

Here, have a sleepy cat.



100M users explained

- 100M users
- each user uses the app 10 times a day
  - 1 billion user accesses per day
- each user access causes 10 requests
  - HTML page, images, dynamic requests, query flow
  - so 10 billion requests a day
- means an average of about 100000 queries a second
  - actually not, because the internet users are not distributed equally around the world
  - and don't use the app at the same times equally
- so more like 200000 queries a second
- let's say each query requires 10 disk seeks
  - amortized; some use more, some use less
- what do we need to serve that?

10K servers explained

- let's say a disk does about 100 disk seeks per second
- 2000000 seeks per second mean 20000 disks
- we could try cramming 20000 disks into one server
  - but that'd be a very large and expensive server
  - and we found out a while ago that it's more economical to use lots of small servers rather than one big one
  - also called "warehouse scale computing"
- at 2 disks per server, 10000 servers
- 40 per rack
- fills 250 racks
- about 150 meters of rack space