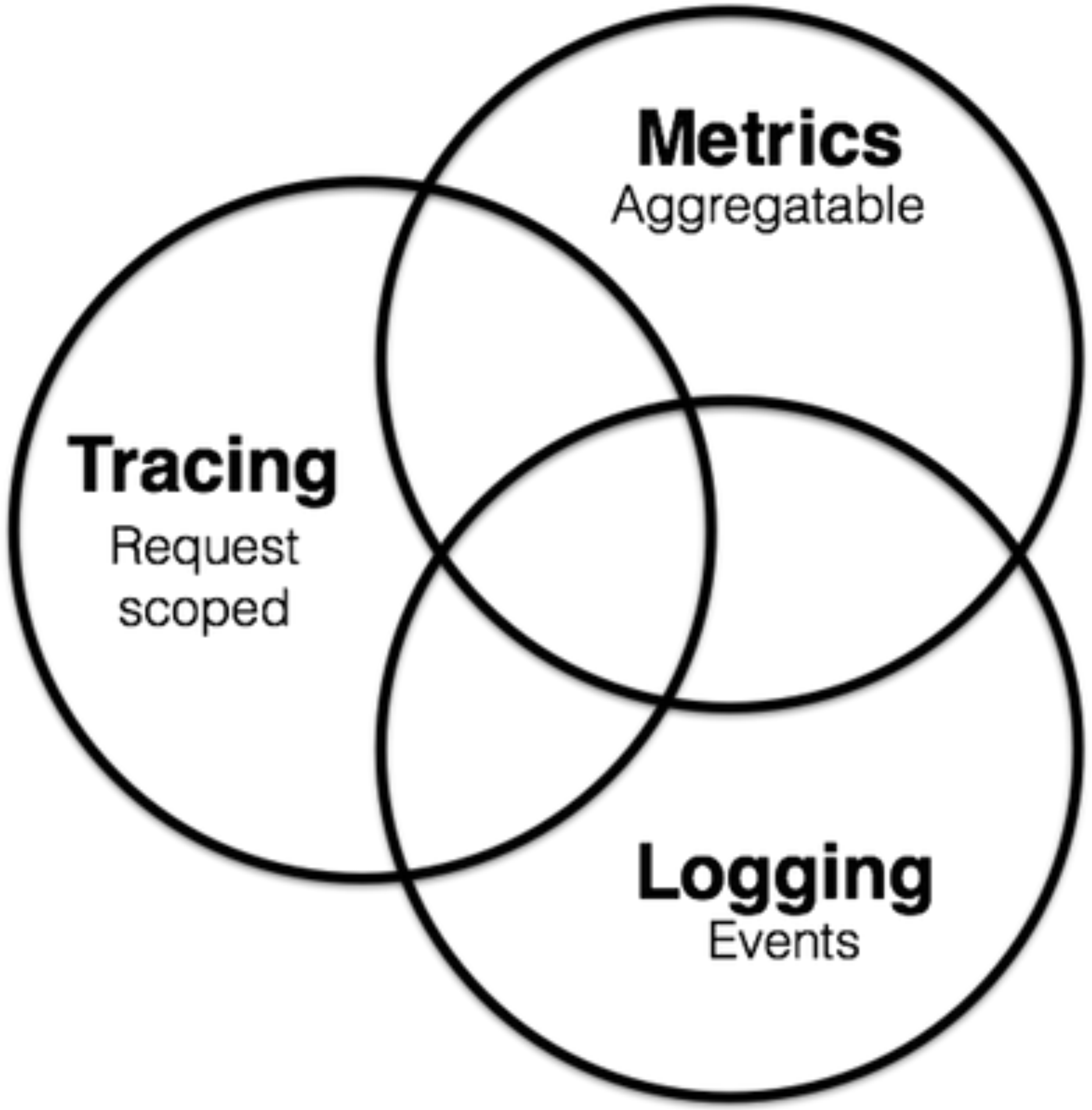
百应

系统监控实践

- 基于Micrometer & Prometheus & Grafana

目标：提升系统可观测性（Observability）

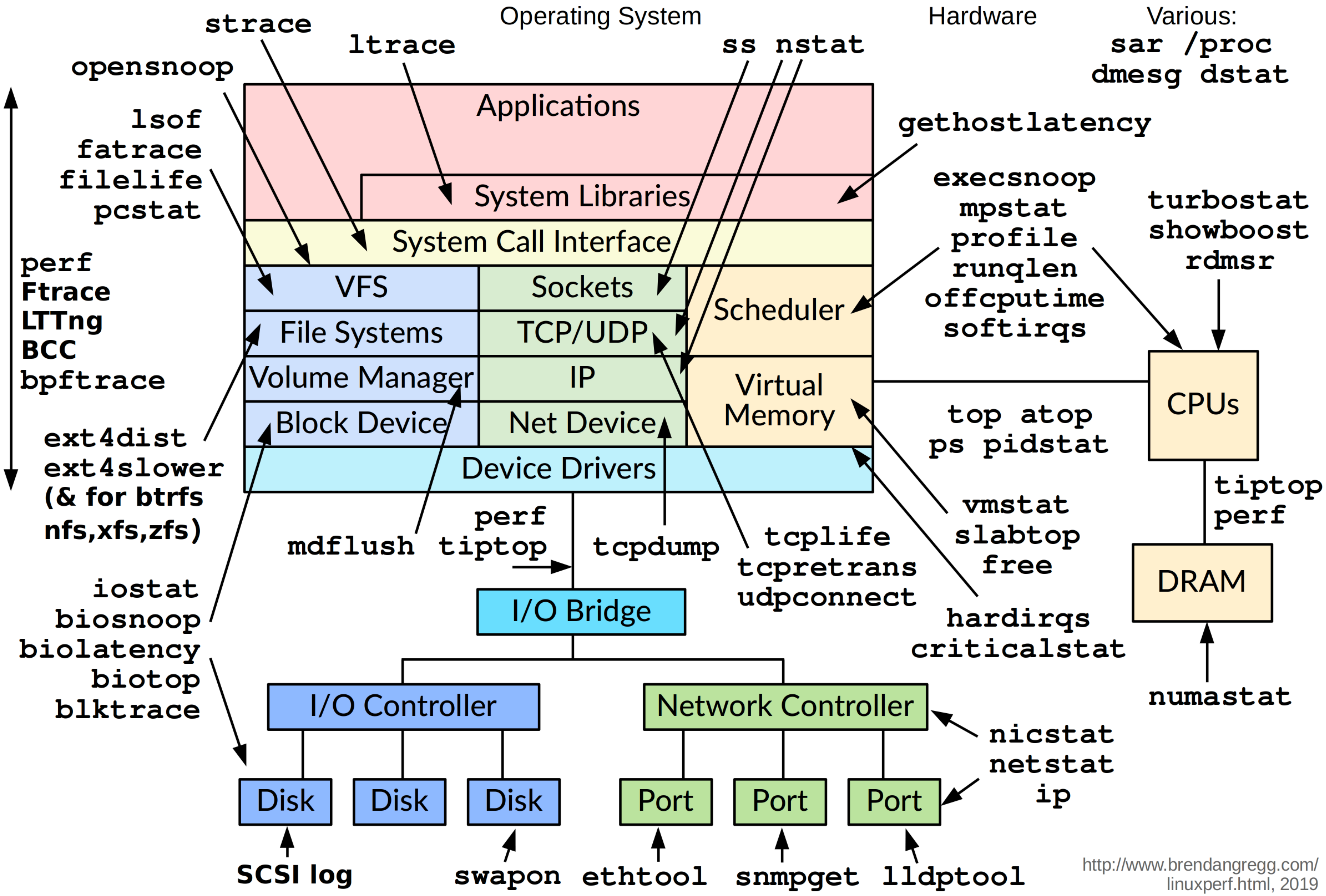
提升系统可观测性的三个途径



监控哪些指标

- The USE Method
- The Four Golden Signals
- The RED Method

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html, 2019

The USE Method

- **U**tilization：资源使用率，百分比
- **S**aturation：资源饱和度，例如任务队列长度
- **E**rror：错误数量

- http://www.brendangregg.com/usemethod.html

The Four Golden Signals

- Latency：请求RT
- Traffic：请求QPS
- Errors：异常数量
- Saturation：系统饱和度，例如dubbo线程池活跃数&排队数
- https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/

The RED Method

- **R**ate：Traffic
- **E**rror
- **D**uration：Latency

- https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/ — Tom Wilkie from Grafana

监控哪些指标

- Rate – QPS
- Error – 异常数量/占比
- Duration – RT
- **Saturation** – 看具体场景，例如队列长度等
- **Utilization** – 看具体场景，例如QPS * RT / Workers

监控工具

MICROMETER
application monitoring

An application metrics facade for the most popular monitoring tools. Think SLF4J, but for metrics.

https://micrometer.io/

工具之间的差异

- 指标维度：是否支持tag*
- 数据聚合：客户端 / 服务端
- 数据上报：推 / 拉

Micrometer

```
MeterRegistry registry = Metrics.globalRegistry;
Meter meter = Counter
      .builder("byai.apm.method.error")
      .tags(Tags.of("application", application))
      .register(registry);


((Counter) meter).increment();
```

MeterRegistry

创建并持有Meter，每个监控系统都有一个MeterRegistry实现

- **SimpleMeterRegistry**
  - 默认实现，存储但不输出数据
- **CompositeMeterRegistry**
  - 组合多个 Registry，实现输出到多个监控系统
- **Global registry**
  - 系统默认提供了一个静态全局的 CompositeMeterRegistry，通过 Metrics.globalRegistry 获取
  - 默认 Spring Boot 注册的所有 registries 都会绑定到 global registry

Meter

生成监控值

- **有唯一的名称**
  - 用相同名称注册不会生成新的 Meter，而是返回之前生成的
- **有类型**
  - COUNTER：计数器，单调递增，例如异常数量
  - GAUGE：瞬时值，可增可减，例如CPU使用率
  - TIMER：RT，记录总时间，总调用次数，计算RT均值
  - DISTRIBUTION_SUMMARY：使用直方图分段统计，实现百分位统计RT

Time Series Database

- identifier –> (t0, v0), (t1, v1), (t2, v2), (t3, v3), ...

MeterFilter

- 允许/拒绝meter注册
- 修改meter，包括名称，tag等
- 注：建议使用filter全局控制percentile及histogram

Memory footprint

- R = Ring buffer length. We assume the default of 3 in all examples. R is set with `Timer.Builder#distributionStatisticBufferLength`.
- B = Total histogram buckets. Can be SLA boundaries or percentile histogram buckets. By default, timers are clamped to a minimum expected value of 1ms and a maximum expected value of 30 seconds, yielding 66 buckets for percentile histograms, when applicable.
- I = Interval estimator for pause compensation. 1.7 kb
- M = Time-decaying max. 104 bytes
- Fb = Fixed boundary histogram. 30b * B * R
- Pp = Percentile precision. By default is 1. Generally in the range [0, 3]. Pp is set with `Timer.Builder#percentilePrecision`.
- Hdr(Pp) = High dynamic range histogram.
  - When Pp = 0: 1.9kb * R + 0.8kb
  - When Pp = 1: 3.8kb * R + 1.1kb
  - When Pp = 2: 18.2kb * R + 4.7kb
  - When Pp = 3: 66kb * R + 33kb

Memory footprint

Pause detection	Client-side percentiles	Histogram	Formula	Example
Yes	No	No	I + M	~1.8kb
Yes	No	Yes	I + M + Fb	For default percentile histogram, ~7.7kb
Yes	Yes	Yes	I + M + Hdr(Pp)	For the addition of a 0.95 percentile with defaults otherwise, ~14.3kb
No	No	No	M	~0.1kb
No	No	Yes	M + Fb	For default percentile histogram, ~6kb
No	Yes	Yes	M + Hdr(Pp)	For the addition of a 0.95 percentile with defaults otherwise, ~12.6kb

Q&A