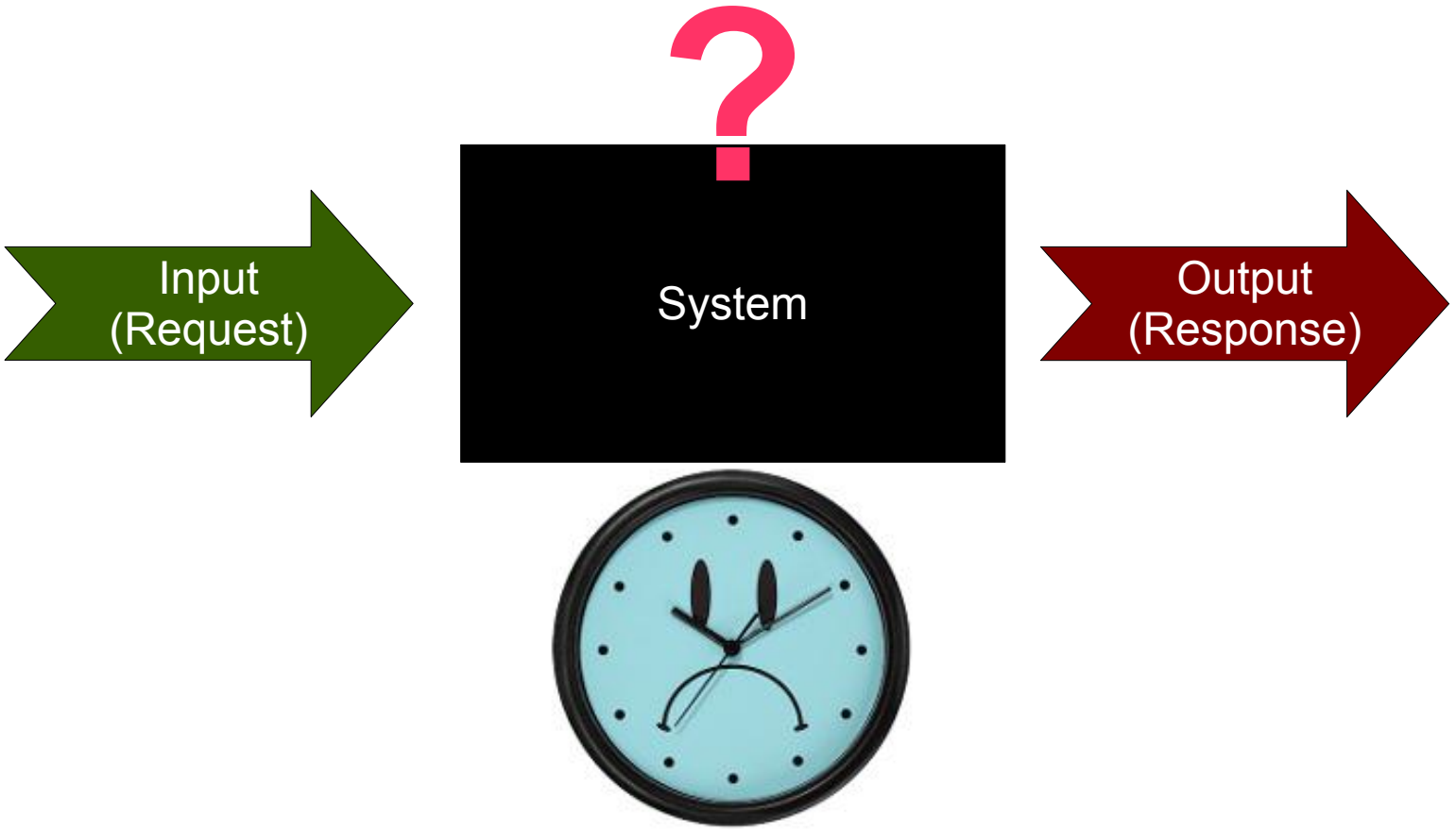
Latency Trace Catcher (LTC)

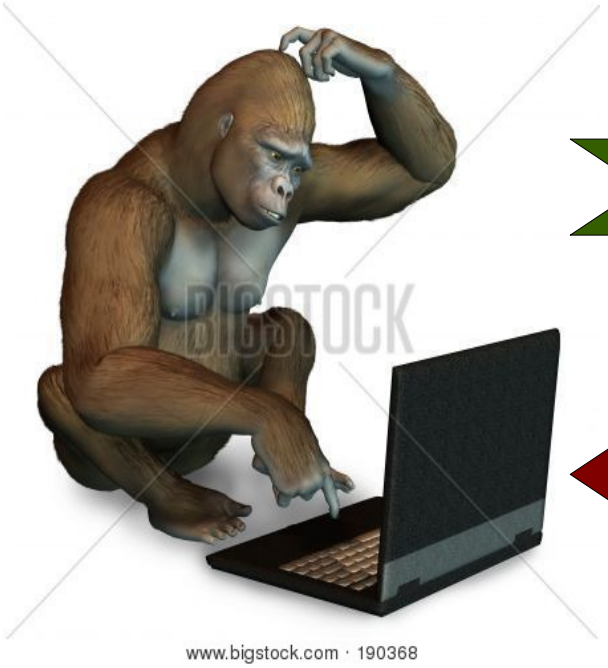
Summer Internship Project
Google, Inc.
New York, NY
2011
Host: Salim Virji

Latency

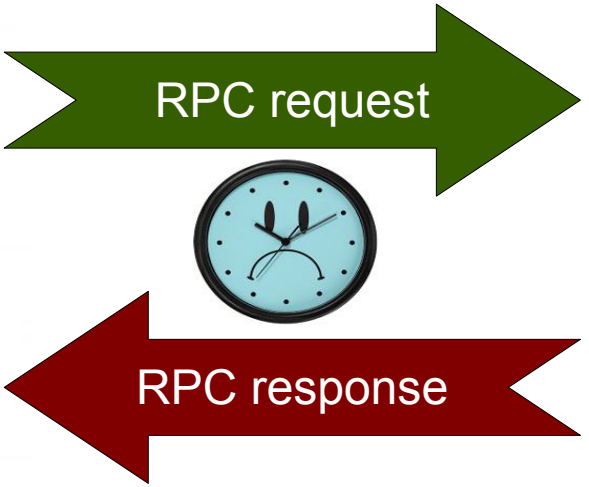


Delay during which the process from input to output is “hidden” (*latent*) in an opaque system.

RPCs



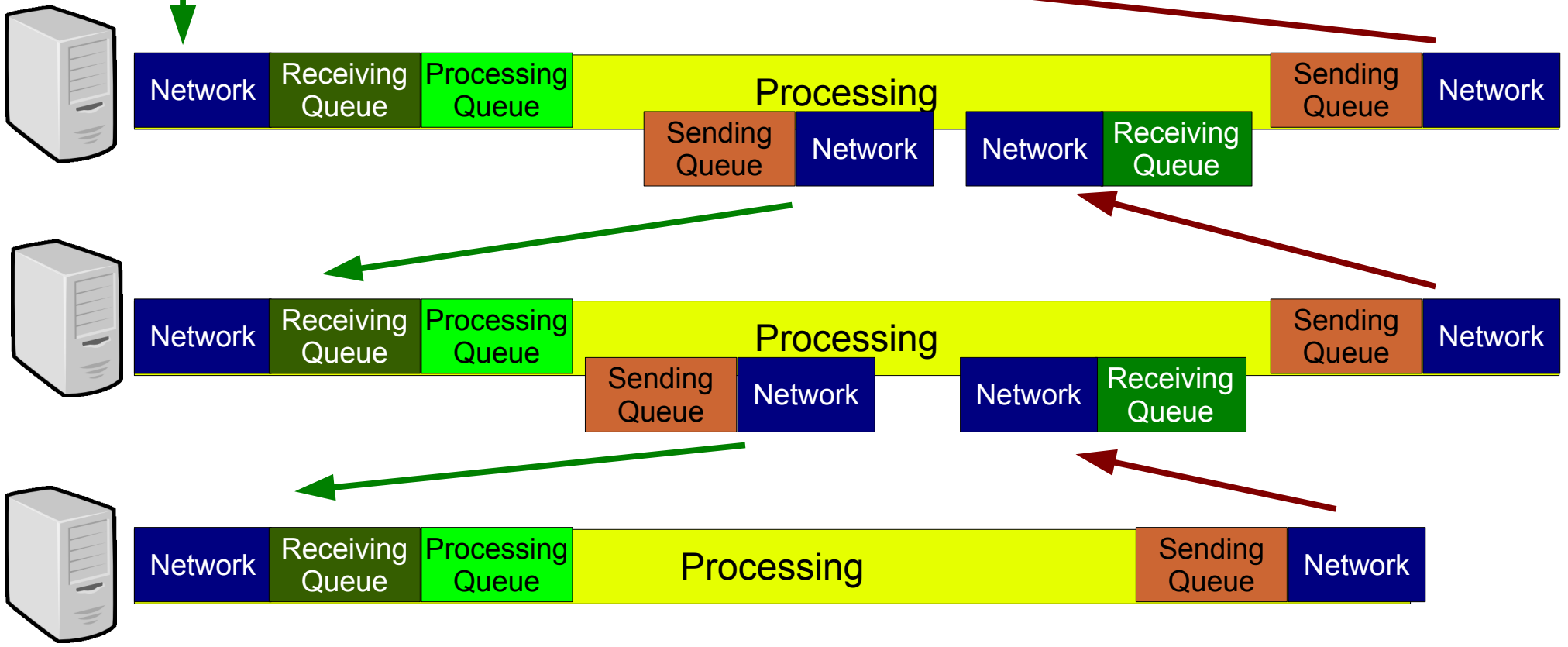
User with client



Remote host with server

In a distributed system, the *request* is often a *remote procedure call* (RPC) to a remote host. The *system* includes both the local and the remote hosts. Causes of delay are *hidden* from the user.

Nested RPCs



More typically, handling a request in a large distributed system will require nested RPCs from one specialized server to another.

Anomalous Latency

Network | Receiving Queue | Processing Queue | Processing

Sending Queue | Network

RPC may time out.

Network | Receiving Queue | Processing Queue | Processing

STUCK!

Idle 

A complete *trace* of this nested RPC would reveal where it had gotten stuck —and give us a clue as to why.

The Problem

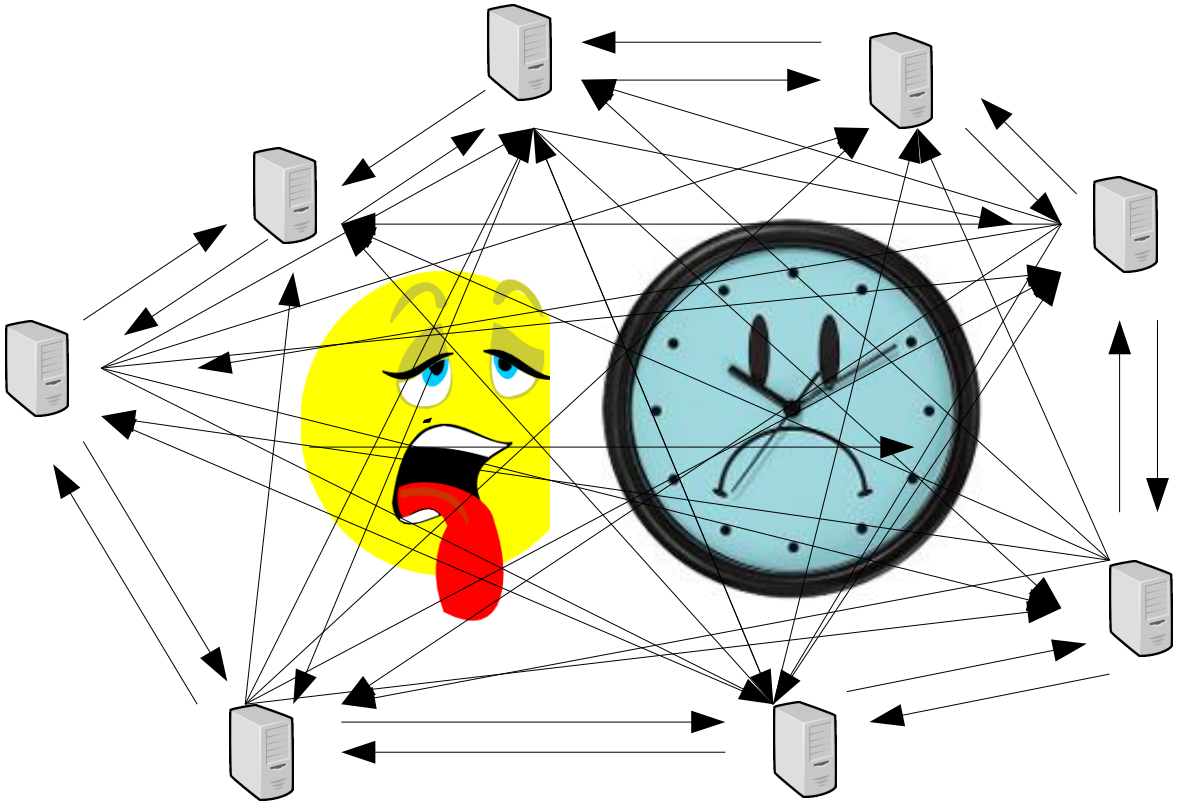
- Get the **traces** of slow RPCs out of “hiding”
- Do so in **near-real time**
- **Alert** administrators automatically and in a timely fashion
- Make traces available for **longer-term storage**

...so that ...

The Goals

- Engineers can fix individual problem quickly
- Engineers can find patterns and fix the underlying causes of excess latency.

Main Challenge



How to gather rich, useful information about the system without burdening that same system and causing artificial, impossibly high latency!

Other Challenges

- Leverage existing Google tools as much as possible, BUT ...
- Avoid touching production code wherever possible
- Make *minimal* changes to production code
- Collaborate with several project teams (storage system, maintenance, diagnostic tools, monitoring tools …), each with its own distinct goals and POV
- Design a system from scratch, get it approved, and code it up, with unit tests, in a summer.

Google Resources (1)

- **TraceBuilder***
  Daemon running alongside target server (same machine)
  - Periodically builds traces from all RPC messages
  - Stores traces for ~5 minutes in a **FIFO** buffer
  - Handles TraceFetcher requests
- **TraceFetcher***
  Central server that fetches a trace by trace ID
  - Fetches from all servers involved in servicing the RPC and assembles traces together
  - Stores traces in a central storage depot (TraceBin)

* Fake name to protect Google.

Google Resources (2)

- **TraceBin***
  Central storage, keeps traces for ~2 weeks.
  - Graphic Web UI
  - Analytical and statistical tools
- **ServerMonitor***
  Central daemon per cluster or data center
  - Collects information all its target servers expose and update periodically
  - Provides statistics (Web pages)
  - Can alert maintenance engineers (e-mail, pager).

* Fake name to protect Google.

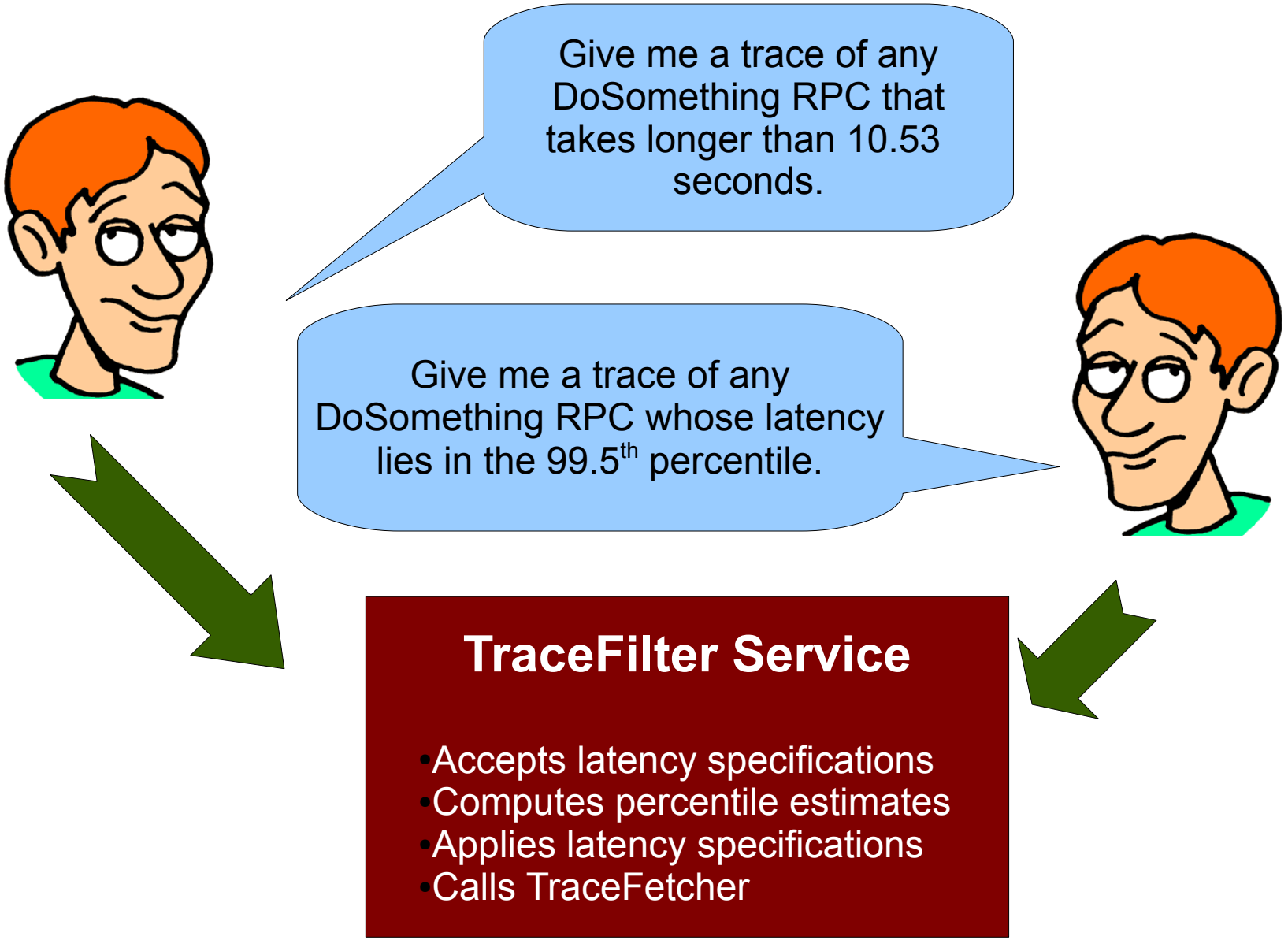
Solution Approach (1)

- Specialized **TraceBuilder**
  - Uses a *filter* to determine eviction from the trace buffer
- **TraceFilter** service
  - Implements filtering based on administrator's specifications
  - Calls TraceFetcher
  - Exposes list of reported traces for ServerMonitor

Solution Approach (2)

- Central **ThresholdBroadcaster** (server and client)
  - Takes user's threshold specifications
  - Sends them to all instances of TraceFilter in the cluster/data center
- Specialized **ServerMonitor**
  - Gathers list of reported traces and alerts engineers.

TraceFilter Service



Protocol Buffer Definition

```
enum ThresholdType {
    THRESHOLD_LITERAL = 0;
    THRESHOLD_PERCENTILE = 1;
}

message Threshold {
    optional string name = 1;
    optional double value = 2;
    optional ThresholdType threshold_type = 3;
}

message SetThresholdRequest {
    repeated Threshold entries = 1;
}

service TraceFilter {
    rpc SetThreshold(SetThresholdRequest) returns (EmptyMessage);
}
```

*** simplified ***

TraceFilter Logic: Reporting

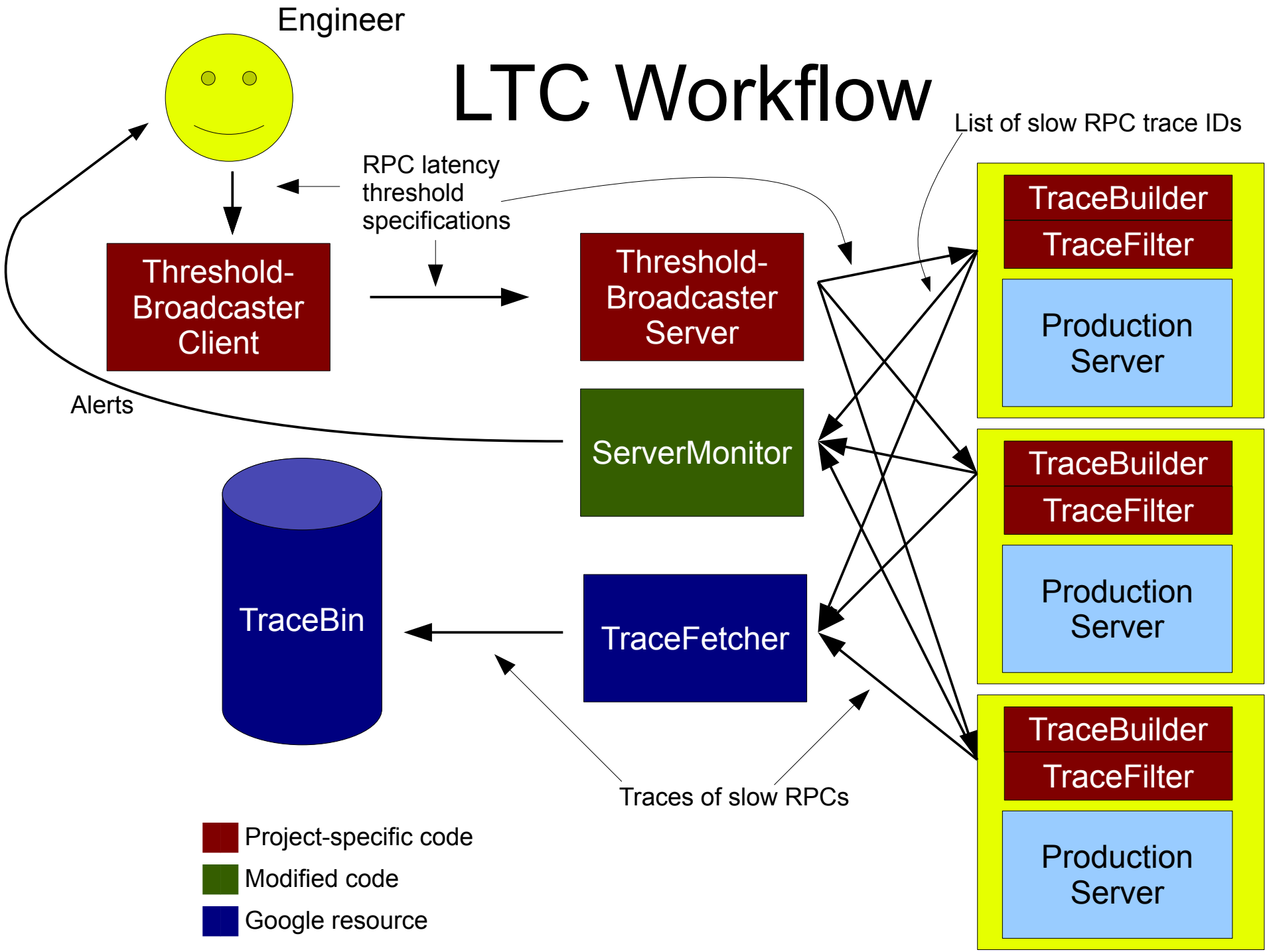
if (trace is complete && trace RPC has spec && trace latency ≥ spec threshold)
    call TraceFetcher with trace ID to have it fetch the trace and store it in TraceBin
    put trace ID on *pending* list
    expose trace ID to ServerMonitor

TraceFilter Logic: Eviction

```
if (trace buffer is full)
    for each trace in trace buffer (from the back)
        if (trace RPC has spec)
            if (trace is not complete || trace is on pending list)
              keep in trace buffer
            else discard trace
        else discard trace
     accept no new traces until the next cycle
```

TraceBuilder Specialization

- **Modify** “normal” (production) code minimally
  class TraceBuilder
  - Abstract trace buffer eviction policy (default FIFO)
  - Insert “hooks” for special intervention (default no-ops)
- **Specialize**
  class ltc::TraceBuilder : public TraceBuilder
  - Overrides eviction policy to use TraceFilter
  - Passes completed traces to TraceFilter for checking


Engineer
LTC Workflow
List of slow RPC trace IDs
RPC latency threshold specifications
Threshold-Broadcaster Client
Threshold-Broadcaster Server
TraceBuilder
TraceFilter
Production Server
Alerts
ServerMonitor
TraceBuilder
TraceFilter
Production Server
TraceBin
TraceFetcher
TraceBuilder
TraceFilter
Production Server
Traces of slow RPCs
Project-specific code
Modified code
Google resource

Thank You

- Salim Virji, host *extraordinaire*
- Many other people at Google whose names and team names are probably company secrets ☺.