SDKs are not services and what this means for SREs

Justin Coffey <j.coffey@criteo.com>

Engineering Director, SRE

@jqcoffey

Agenda

This talk covers work done by a data engineering team over a period of 7 years. It mixes storytelling with “insights” and a few code examples. The structure is as follows:

- ➢ Introduction, context and motivation
- ➢ A feature team becomes a service team
- ➢ A missed delivery and its analysis
- ➢ Rebooting
- ➢ Closing thoughts

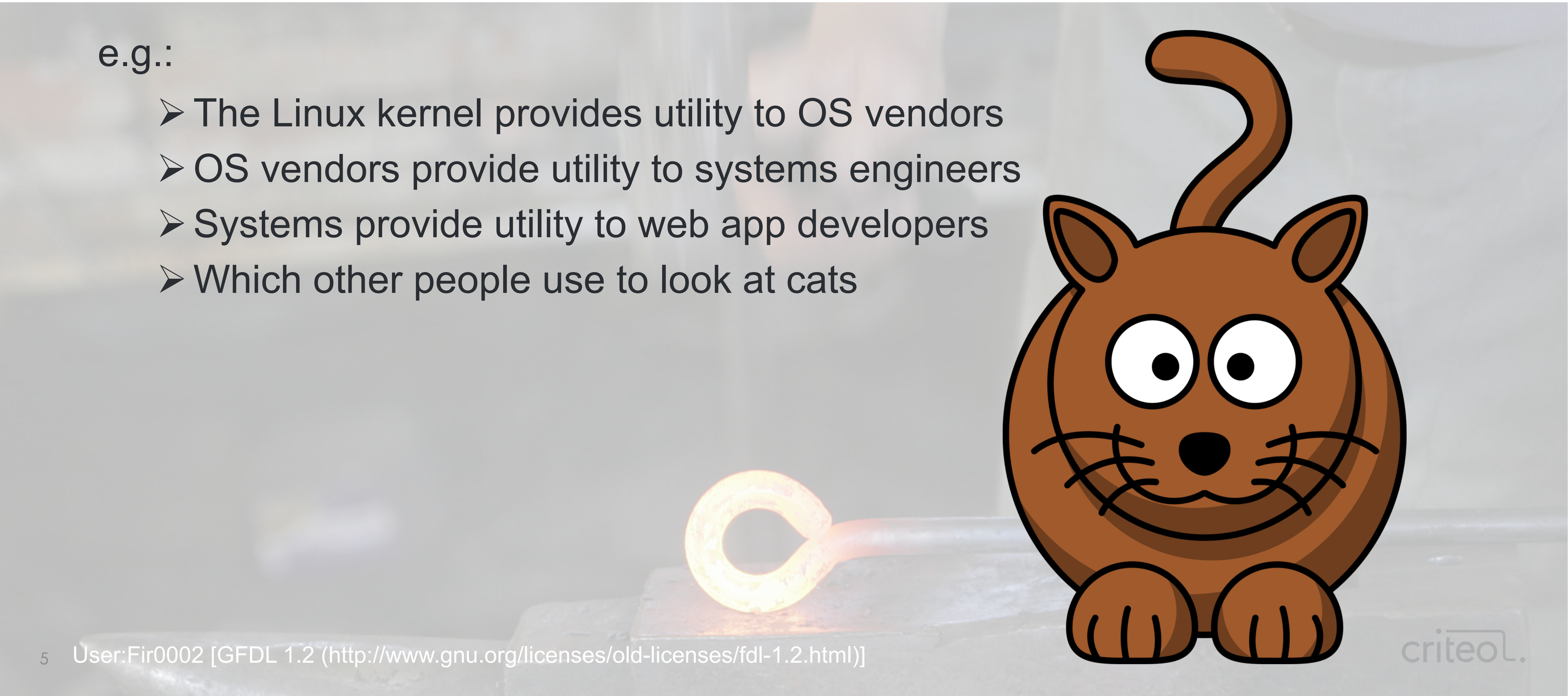
Introduction, context and motivation

Who am I?

I’m the manager of what is now called SRE Data Processing. Over the years the teams I’ve managed have had their hands in pretty much every aspect of data building, processing, reporting and accessing.

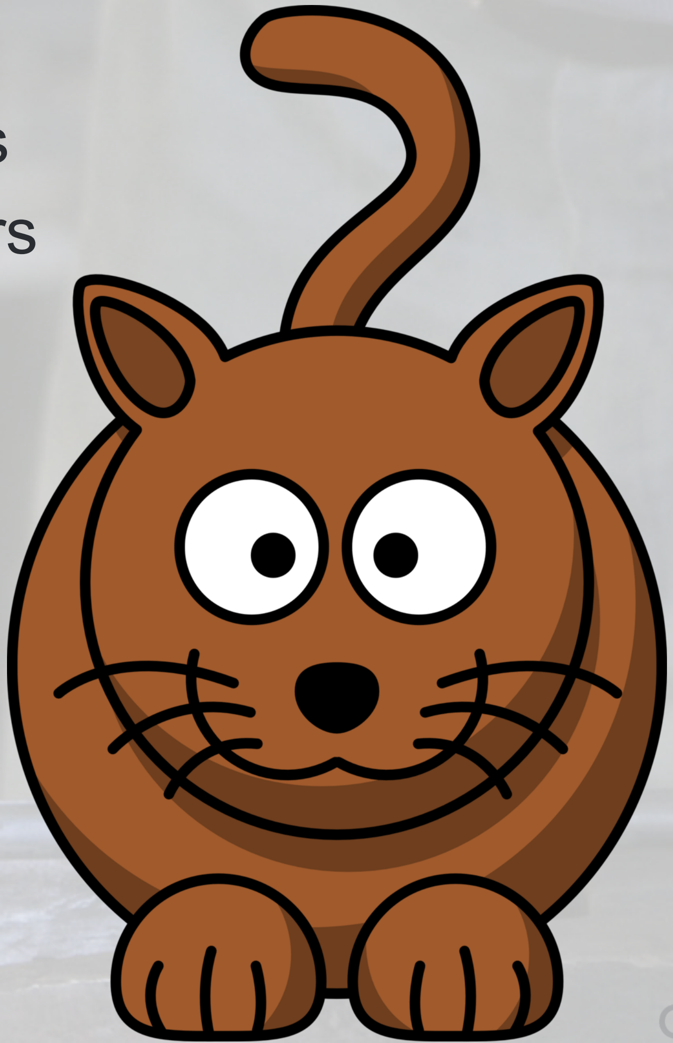
~~Most~~ all mistakes that I’ll highlight are my mine ☺.

Deep Thought #1: software engineering is about providing utility

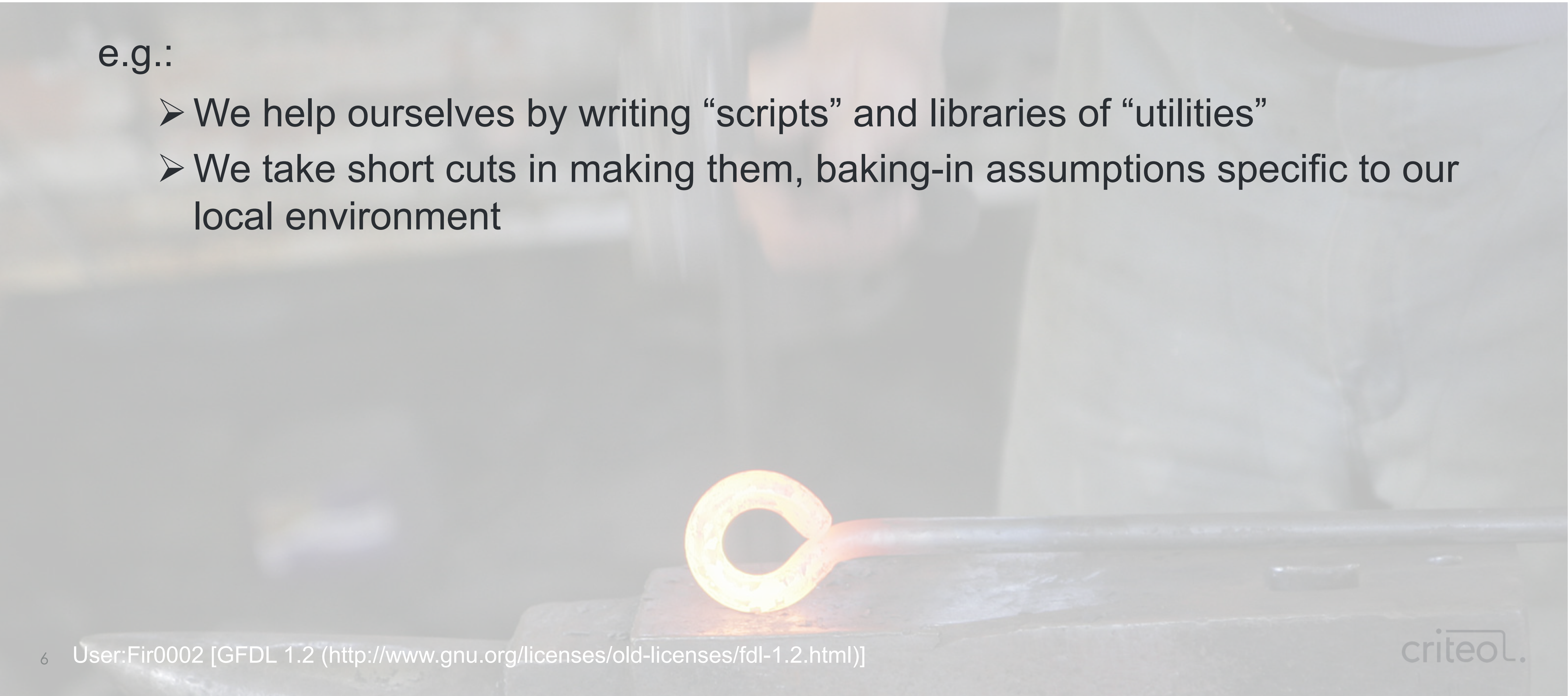


e.g.:

- ➢ The Linux kernel provides utility to OS vendors
- ➢ OS vendors provide utility to systems engineers
- ➢ Systems provide utility to web app developers
- ➢ Which other people use to look at cats



Deep Thought #2: not all software is destined for massive utility



e.g.:

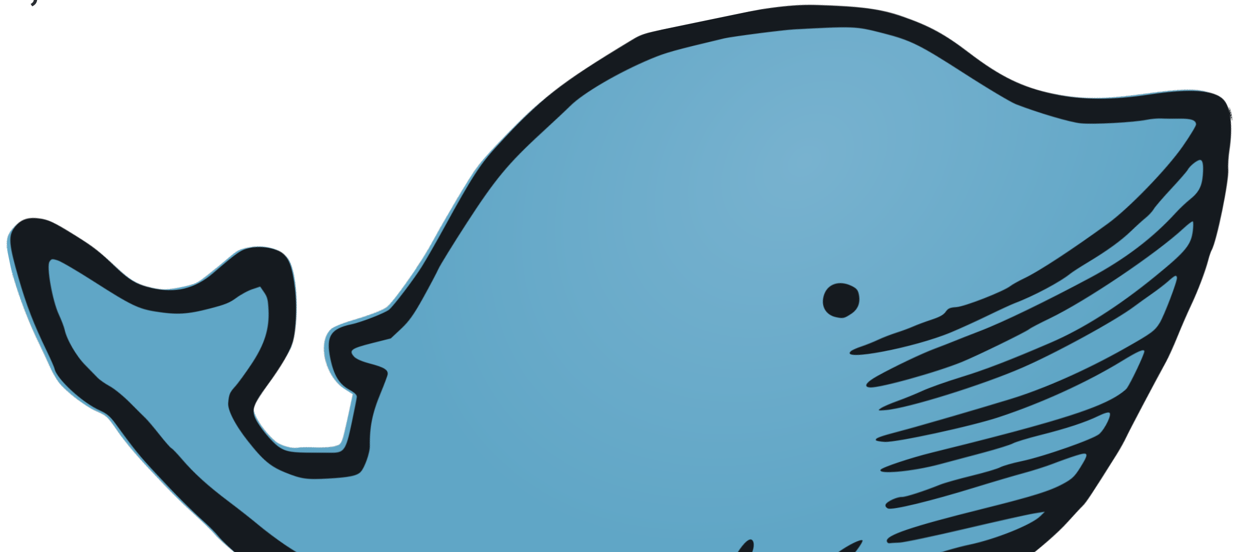
- ➢ We help ourselves by writing “scripts” and libraries of “utilities”
- ➢ We take short cuts in making them, baking-in assumptions specific to our local environment

Criteo

You may not know us, but we know you (as the joke goes).

- We do personal recommendation in advertisements (aka “retargeting”)
- We observe a large portion of the Internet, reach 1B+ users per month
- We have ~1000 potential contributors to data production (600+ in R&D)

tl;dr, we’re not small.



Motivation

This talk has long been brewing inside my head. It’s a 7-year post mortem of what a bunch of amazing engineers did together.

I’m obliged to say that, while I am here at the grace of Criteo, these are my thoughts and analysis. Criteo is a large and diverse organization that encourages open and honest debate. This talk is part of that process.

A feature team becomes a service team

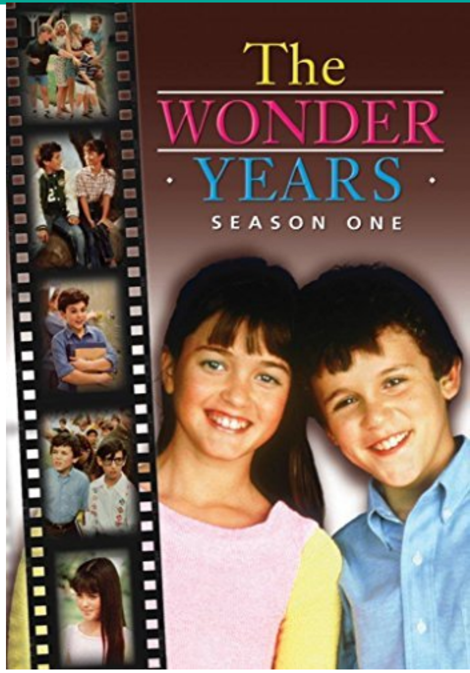
The Early Years

The team in question was formed to fix failing ETL pipelines.

We made quick work of individual jobs, but deployment and orchestration was not obvious.

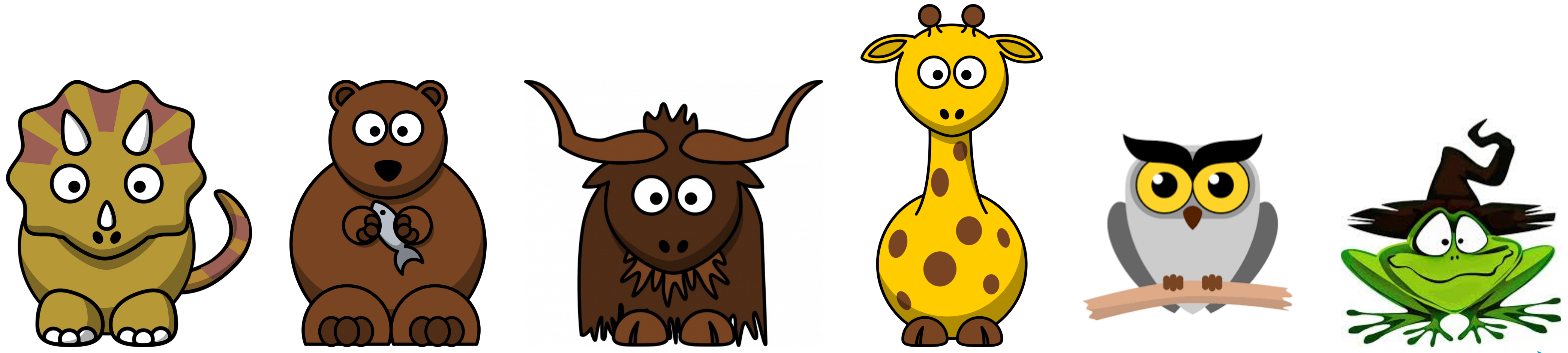
Being UNIX-y, we scripted ourselves out of the problem.

It worked pretty well and we were happy ☺



We’re Popular

We start getting requests for help. Lot’s of them. We were flattered!



We still don’t know who this guy is.

Surprise!

We are victims of our own success and are now the proud owners of most of Criteo’s batch data production.

Problem 1: We have very little monitoring and no user isolation

Problem 2: We don’t actually own most of the code that’s executing

Surprise! #2!

The legacy billing data pipeline, a hypercritical system is failing. We won the no-bid contract to fix it.

New requirements: strict SLOs, SOX compliance, correctness

Old Problems: see previous slide

New Problems, New System

Our scripts will no longer cut it.

We need a real scheduler and to redistribute ownership of jobs.

We decide to write a scheduler library in Scala.

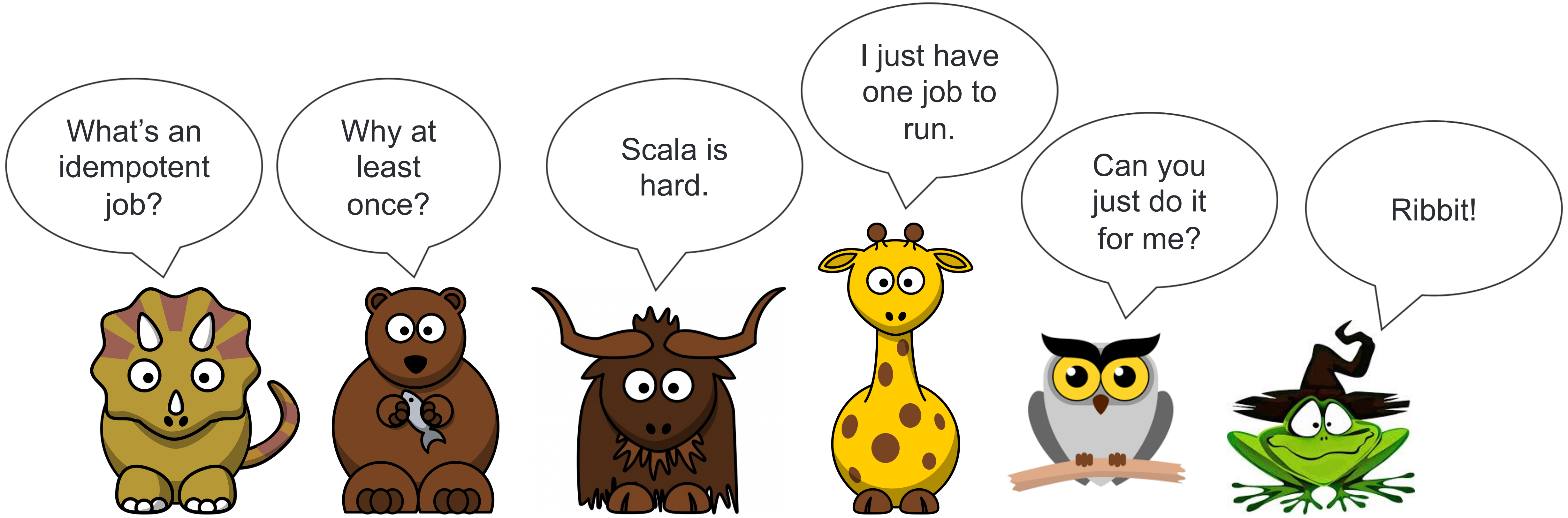
https://github.com/criteo/cuttle

```
import com.criteo.cuttle._
import com.criteo.cuttle.platforms.local._
import com.criteo.cuttle.timeseries._
import java.time.ZoneOffset.UTC
import java.time._
import scala.concurrent.duration._

object Scheduler {
  def main(args: Array[String]): Unit = {
    // define the start of our project to yesterday, UTC
    val start = LocalDate.now.minusDays(1).atStartOfDay.toInstant(UTC)

    // define simple jobs that will execute every hour
    val hello = Job("hello", hourly(start)) { implicit e =>
      // client code goes here...
      e.streams.info(s"${e.context.start} - Hello,")
    }

    val world = Job("world", hourly(start)) { implicit e =>
      // client code goes here...
      e.streams.info(s"${e.context.start} - World!")
    }

    val project = CuttleProject("Hello World") {
      // declare the order of operations
      world dependsOn hello
    }

    // start the scheduler
    project.start()
  }
}
```

A missed delivery and its analysis

The adoption rate is tepid at best

We slayed the billing data pipeline problem and reduced operational headaches in the process. What gives?


What's an idempotent job?
Why at least once?
Scala is hard.
I just have one job to run.
Can you just do it for me?
Ribbit!

A first analysis

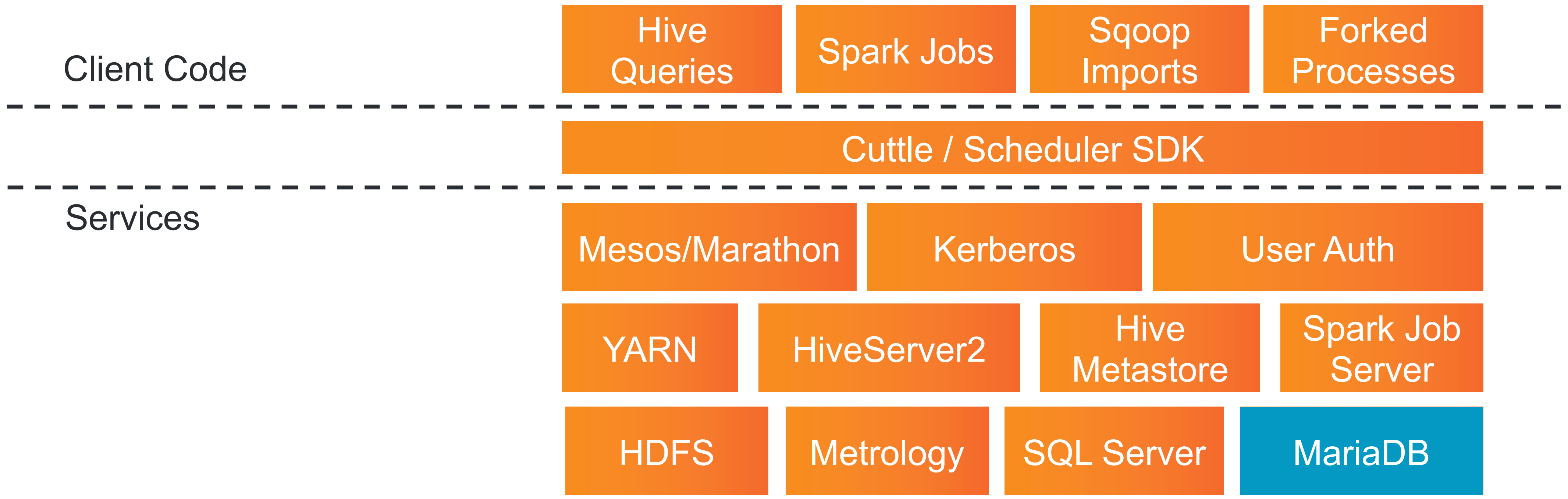
1. Folks don’t know the particulars of our domain
2. Scala is harder to pickup than we thought

After many rounds of training we were able to get new data pipelines on our scheduler, but we couldn’t find takers for the legacy jobs.

This hybrid world establishes itself as the new norm.

Going further in our analysis

We start to see broad adoption of our scheduler, but many teams still seem unhappy with it. Our toil, mostly from support, is high. Why?



It took 6 months to migrate from MySQL to ~~MySQL~~ MariaDB

Release lag when combined with fast moving infrastructure causes high toil for SDK maintaining teams and their clients.

A mid-term post mortem

Our scheduler is everywhere, but the day to day is hard for everyone.

Mistake 1: We made incorrect assumptions about the level of understanding our clients had in the domain in which we were experts.

Mistake 2: We chose a language with a steep learning curve that was overkill for most tasks

Mistake 3: Our SDK approach blurred the lines between who was responsible for what in the face of failures or infrastructure changes

Rebooting

This isn’t working, part deux

Gut instinct got us pretty far, but now we need to strategimize, er I mean, we need some strategery. Turns out we didn’t need to look too far.

SQL

The nice properties of SQL and the RDBMS model

- SQL is self describing and easily parsed
- It is a DSL that runs in a sandbox—the RDBMS—via RPC
- All of our users are already proficient in it and documentation abounds

Taken all together as platform this should provide high ease of use for clients while reducing our operational burden.

Scheduling with SQL

```
SELECT
  -- the schema is defined here
  day,
  foo,
  count(1) as foos
FROM
  -- the parent dependencies are here
  bar
WHERE
  -- the specific subset of the table (aka partition)
  day = ?
GROUP BY
  day, foo
```

```
-- give the dataset a name
CREATE DATASET fact_foo_daily AS

  -- define a partitioning value enumeration
  FOR t in TIMESERIES('@daily')

-- 'day' has been promoted to a partition value
INSERT CAST(t as DATE) as day

SELECT
  -- the rest of the schema remains here
  foo,
  count(1) as foos
FROM
  bar
WHERE
  -- parameterize the parent with the partition value
  day = CAST(t as DATE)
GROUP BY
  foo
```

SQL vs SDK approaches

```sql
-- give the dataset a name
CREATE DATASET fact_foo_daily AS

  -- define a partitioning value enumeration
  FOR t in TIMESERIES('@daily', RETENTION => 3 DAYS),
      p in ('AS', 'EU', 'US')

-- partition values
INSERT CAST(t as DATE) as day, p as platform

SELECT
  -- the rest of the schema remains here
  foo,
  count(1) as foos
FROM
  bar
WHERE
  -- parameterize the parent with partition values
  day = CAST(t as DATE)
  and platform = p
GROUP BY
  foo
```

```scala
import com.criteo.cuttle._
import com.criteo.cuttle.platforms.local._
import com.criteo.cuttle.timeseries._
import java.time.ZoneOffset.UTC
import java.time._
import scala.concurrent.duration._

object Scheduler {
  def main(args: Array[String]): Unit = {
    val start = LocalDate.now.minusDays(1).atStartOfDay.toInstant(UTC)

    val hws = for (p <- List('AS', 'EU', 'US')) yield {

      val hello = Job("hello", hourly(start)) { implicit e =>
        e.streams.info(s"${e.context.start}/$p - Hello,")
      }

      val world = Job("world", hourly(start)) { implicit e =>
        e.streams.info(s"${e.context.start}/$p - World!")
      }

      world dependsOn hello
    }

    val project = CuttleProject("Hello World") {
      hws.reduce(_ and _)
    }

    project.start()
  }
}
```

Executing non-SQL with SQL

With a Foreign Transformation Interface (FTI) extension, our system can execute arbitrary code using the same scheduling language.

```
CREATE DATASET fact_foo_daily AS
  FOR t in TIMESERIES('@daily')
INSERT CAST(t as DATE) as day

SELECT
  foo,
  foos
FROM RUN_PYTHON(
  -- the name and version of the Python artifact
  'com.criteo:foos_py:12345',
  -- this will determine the input data's location to pass via CLI arguments
  DATA_FILES(SELECT *
             FROM bar
            WHERE day = CAST(t AS DATE)),
  )
  -- this declares the types for the schema
  AS result (foo STRING, foos INTEGER)
```

Exporting data with SQL

With a COPY extension we can also declare exports to some other system.

```
CREATE DATASET fact_foo_daily AS
  FOR t in TIMESERIES('@daily')
INSERT CAST(t as DATE) as day

SELECT
  foo,
  count(1) as foos
FROM
  bar
WHERE
  day = CAST(t as DATE)
GROUP BY
  foo

COPY VERTICA_EXPORT('vertica.criteo.com')
```

Research and over communicate

Before we launch full scale development, we practice high quality engineering practices like:

- ➢ Read papers[1]
- ➢ Build a minimal prototype to flesh out our model and validate it
- ➢ Formalize and share our findings via an internal RFC
- ➢ Hold meetings with stakeholders to clarify finer points

Is this Agile? Waterfall? You decide.

[1] https://ai.google/research/pubs/pub47224

Planning delivery

It’s all well and good to build momentum around a project, but the trick is always how to get it out the door.

Constraints: Fixed head count and lots of existing infrastructure to support

Methodology: Focus. Align. Negotiate. Reuse.

Closing thoughts

Summary findings

We need to be careful in extrapolating one team’s work in a specific domain to the rest of our industry.

Rabbit holes, and all.

Nevertheless…

criteo.

Lesson 1

Listen!

Lesson 2

Be wary of state!

Lesson 3

Don’t over aggregate!

Thank you!

We're hiring and have an amazing rooftop.

Come see it at our free conference, Nov 19

NABD CONFERENCE

https://nabdconf.com/