1) Another issue was that ETL pipelines introduced latency — nightly jobs (the norm) meant that business intelligence was being conducted on day-old data. As the pace of business quickened, organizations began demanding fresher and fresher data to guide decision making.
2) To count tweet impressions. Furthermore, not only do we want real-time updates as users are tapping, swiping, and clicking right now, but we want historic counts dating back to the moment a tweet was posted.
3) **Issue 1:** The lambda architecture basically means that everything must be written twice: once for the batch platform and again for the real-time platform.
   Ex1: In batch processing it's easy to compute the cardinality of a set (for example, the number of tweets in an hour), and this value can be used for a variety of computations and optimizations. This isn't possible in a real-time platform if you're processing input incrementally.
   Issue 2: Aggregate values can sometimes fluctuate unpredictably.
   EX2: Let's say the Storm cluster suffered a transient load spike and 10 minutes' worth of log data got dropped. No one would notice this until the logs are processed by the batch layer sometime later. Logging pipelines typically form a different code path than the real-time processing layer and are usually more robust because persistence is an explicit design goal. In this scenario, the missing data reappear, and the aggregate values change suddenly.
4) In the kappa architecture, everything's a stream. And if everything's a stream, all you need is a stream processing engine. What the lambda architecture would call batch processing is simply streaming through historic data.
5) Apache Beam presents a rich API that explicitly recognizes the difference between event time, the time when an event actually occurred, and processing time, the time when the event is observed in the system.