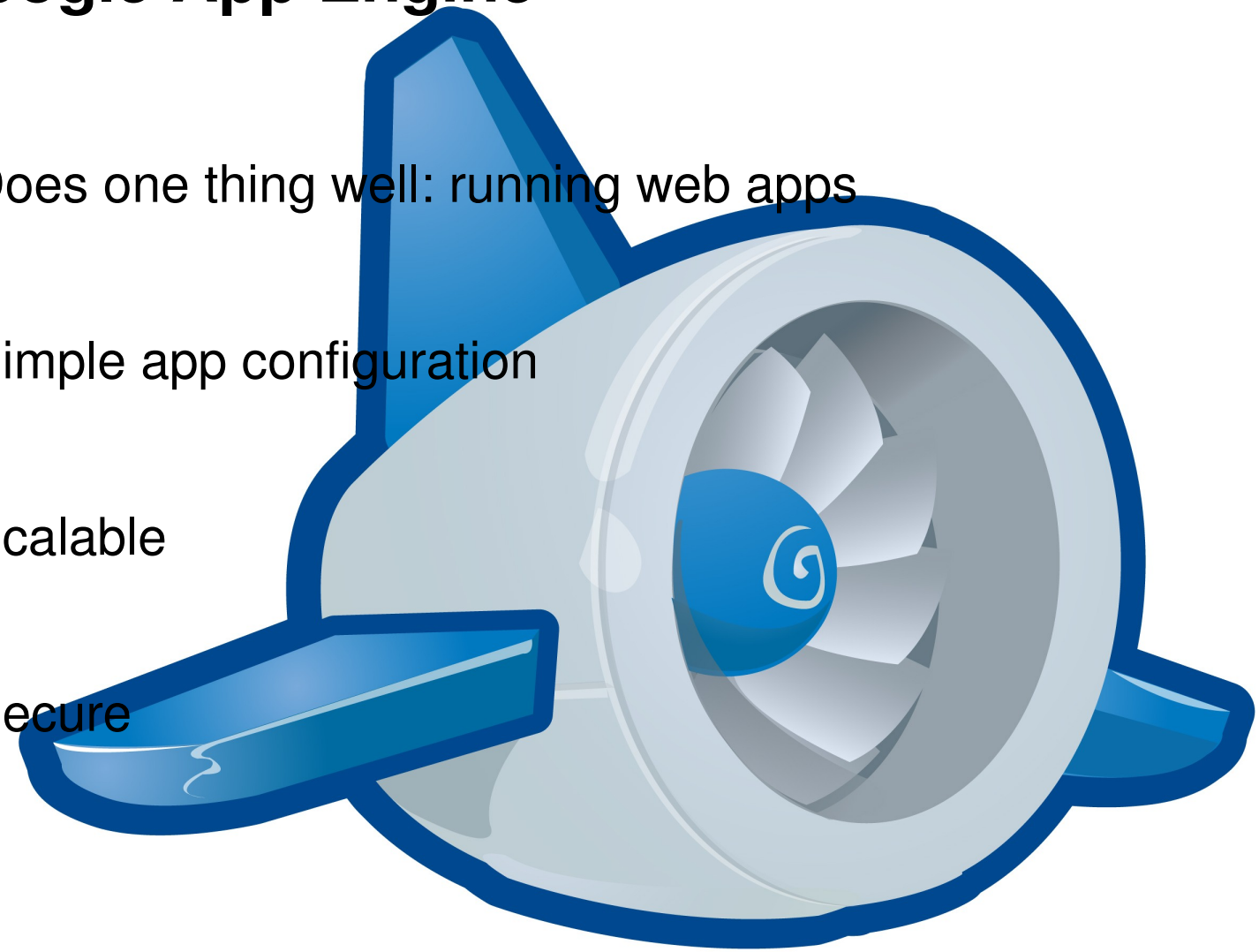
Google App Engine

Guido van Rossum

Stanford EE380 Colloquium, Nov 5, 2008

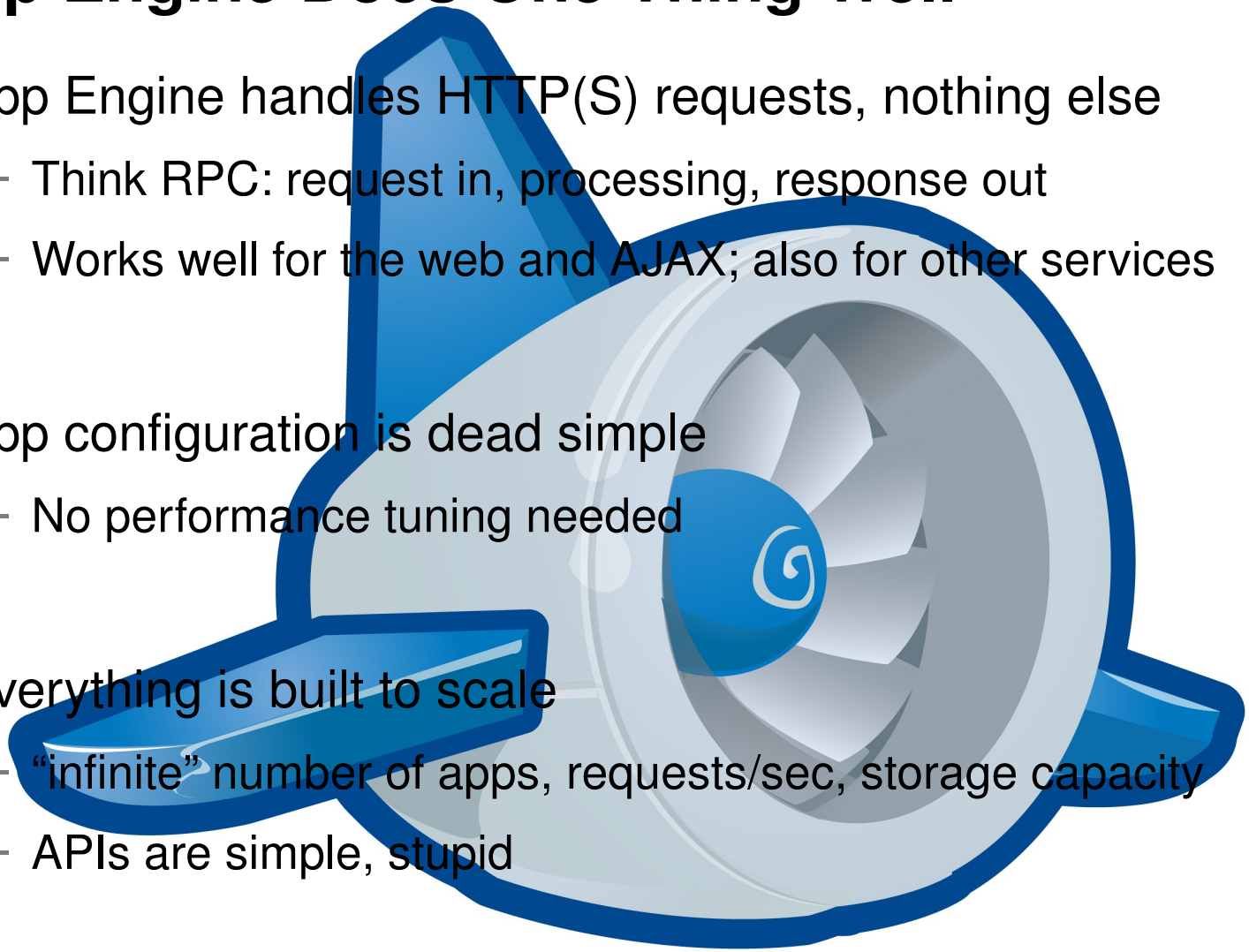
Google App Engine

- Does one thing well: running web apps
- Simple app configuration
- Scalable
- Secure

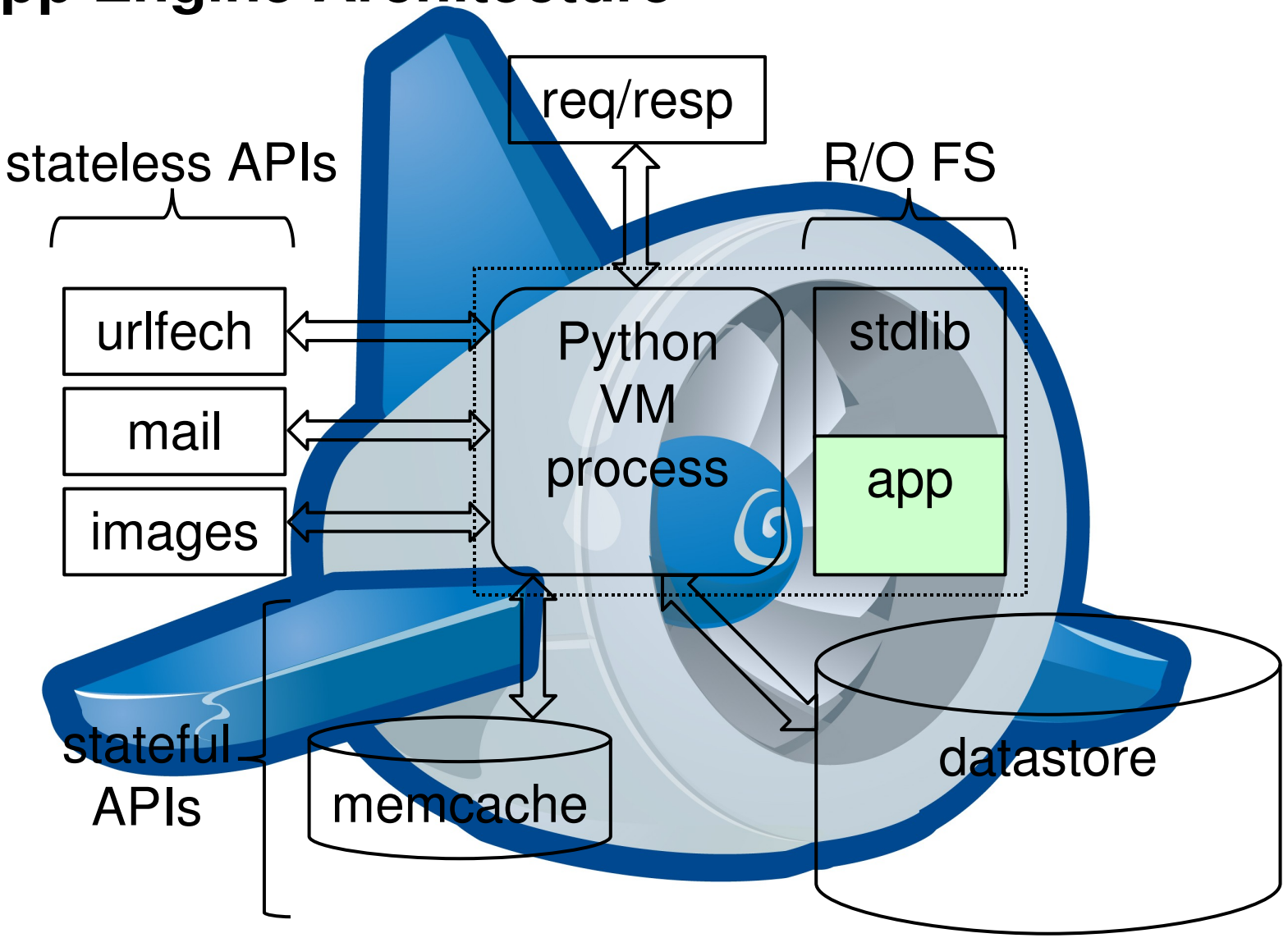


App Engine Does One Thing Well

- App Engine handles HTTP(S) requests, nothing else
  - Think RPC: request in, processing, response out
  - Works well for the web and AJAX; also for other services
- App configuration is dead simple
  - No performance tuning needed
- Everything is built to scale
  - “infinite” number of apps, requests/sec, storage capacity
  - APIs are simple, stupid



App Engine Architecture

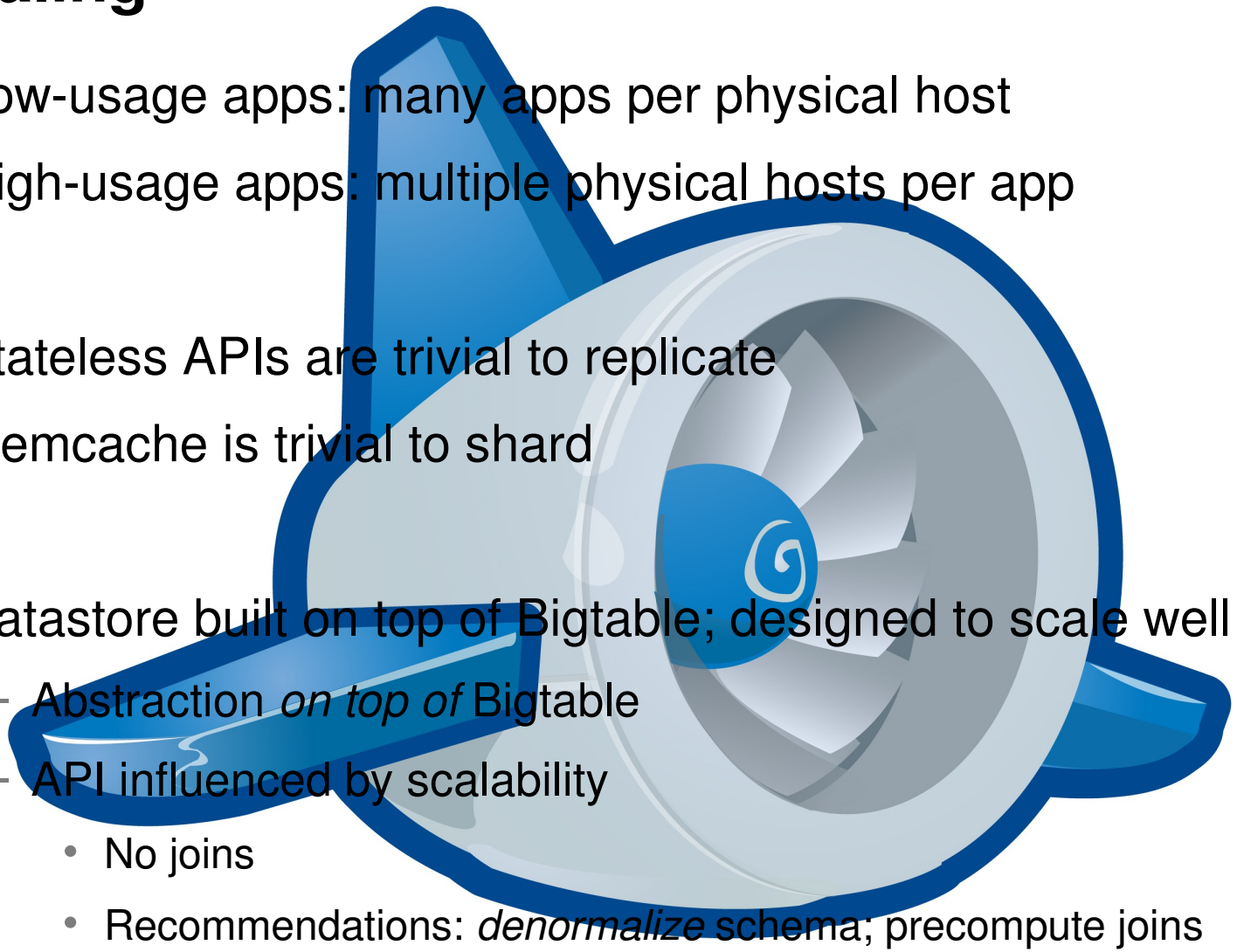


Scaling

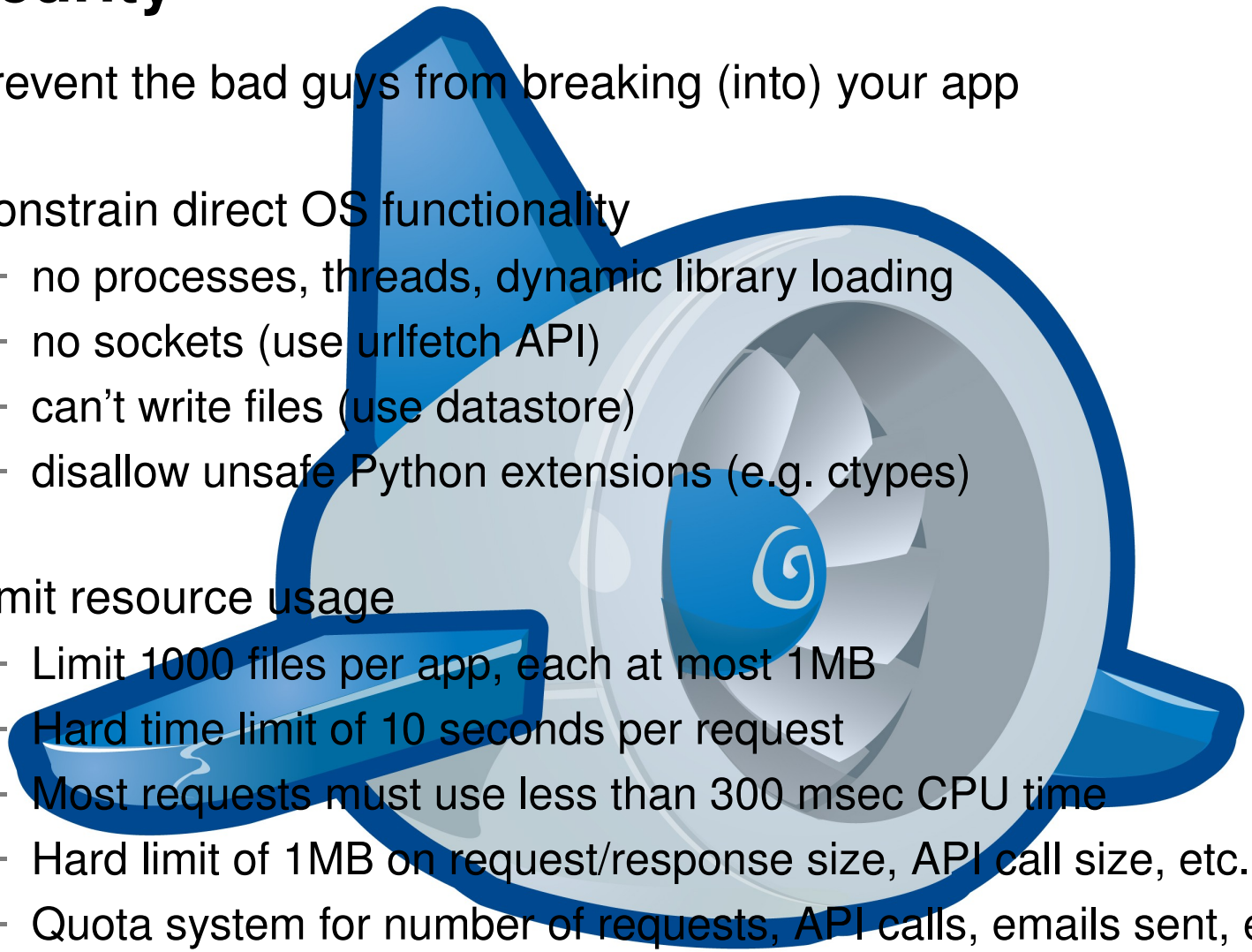
- Low-usage apps: many apps per physical host
- High-usage apps: multiple physical hosts per app

- Stateless APIs are trivial to replicate
- Memcache is trivial to shard

- Datastore built on top of Bigtable; designed to scale well
  - Abstraction *on top of* Bigtable
  - API influenced by scalability
    - No joins
    - Recommendations: *denormalize* schema; precompute joins



Security



- Prevent the bad guys from breaking (into) your app

- Constrain direct OS functionality
  - no processes, threads, dynamic library loading
  - no sockets (use urlfetch API)
  - can't write files (use datastore)
  - disallow unsafe Python extensions (e.g. ctypes)

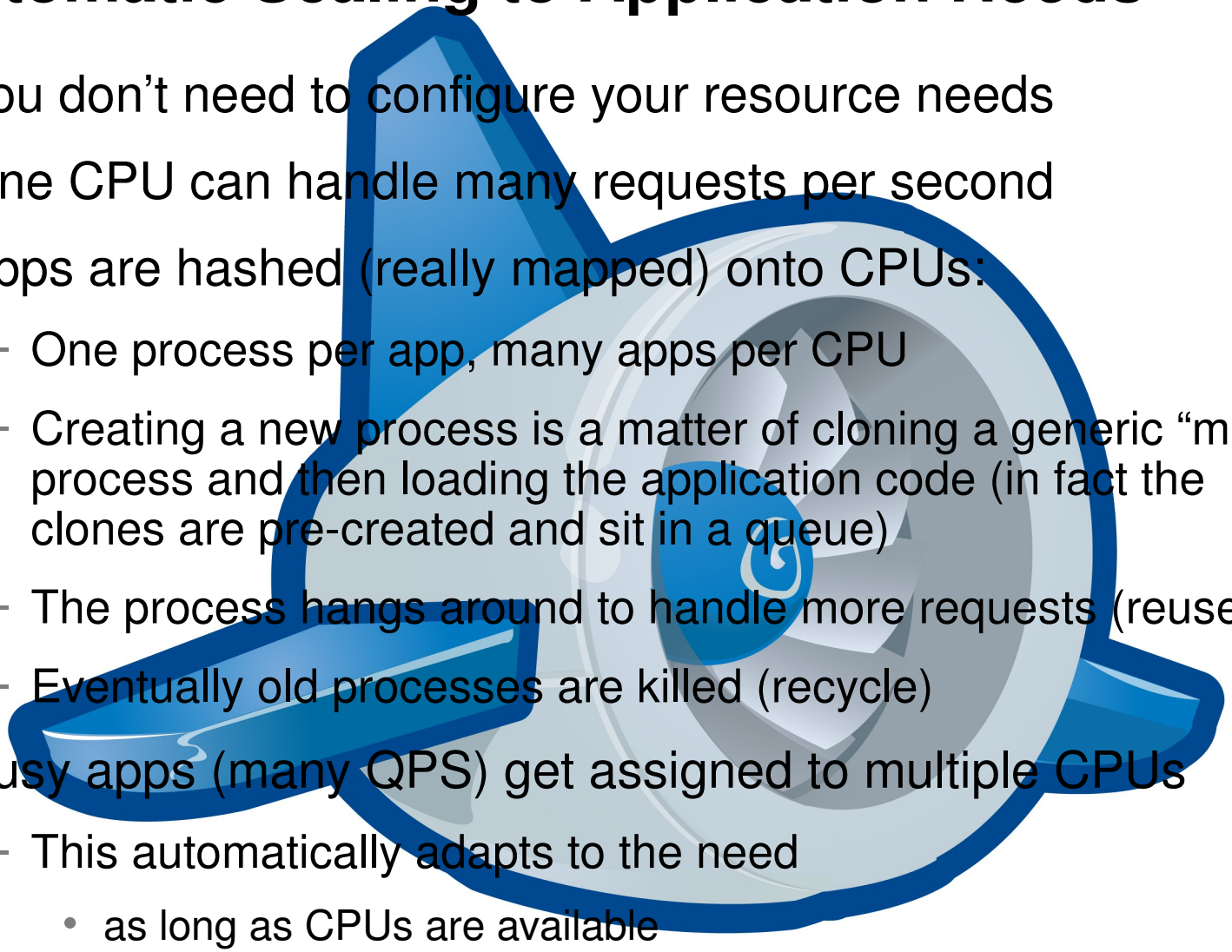
- Limit resource usage
  - Limit 1000 files per app, each at most 1MB
  - Hard time limit of 10 seconds per request
  - Most requests must use less than 300 msec CPU time
  - Hard limit of 1MB on request/response size, API call size, etc.
  - Quota system for number of requests, API calls, emails sent, etc

Why Not LAMP?

- Linux, Apache, MySQL/PostgreSQL, Python/Perl/PHP/Ruby
- LAMP is the industry standard
- But management is a hassle:
  - Configuration, tuning
  - Backup and recovery, disk space management
  - Hardware failures, system crashes
  - Software updates, security patches
  - Log rotation, cron jobs, and much more
  - Redesign needed once your database exceeds one box

- "We carry pagers so you don't have to"

Automatic Scaling to Application Needs

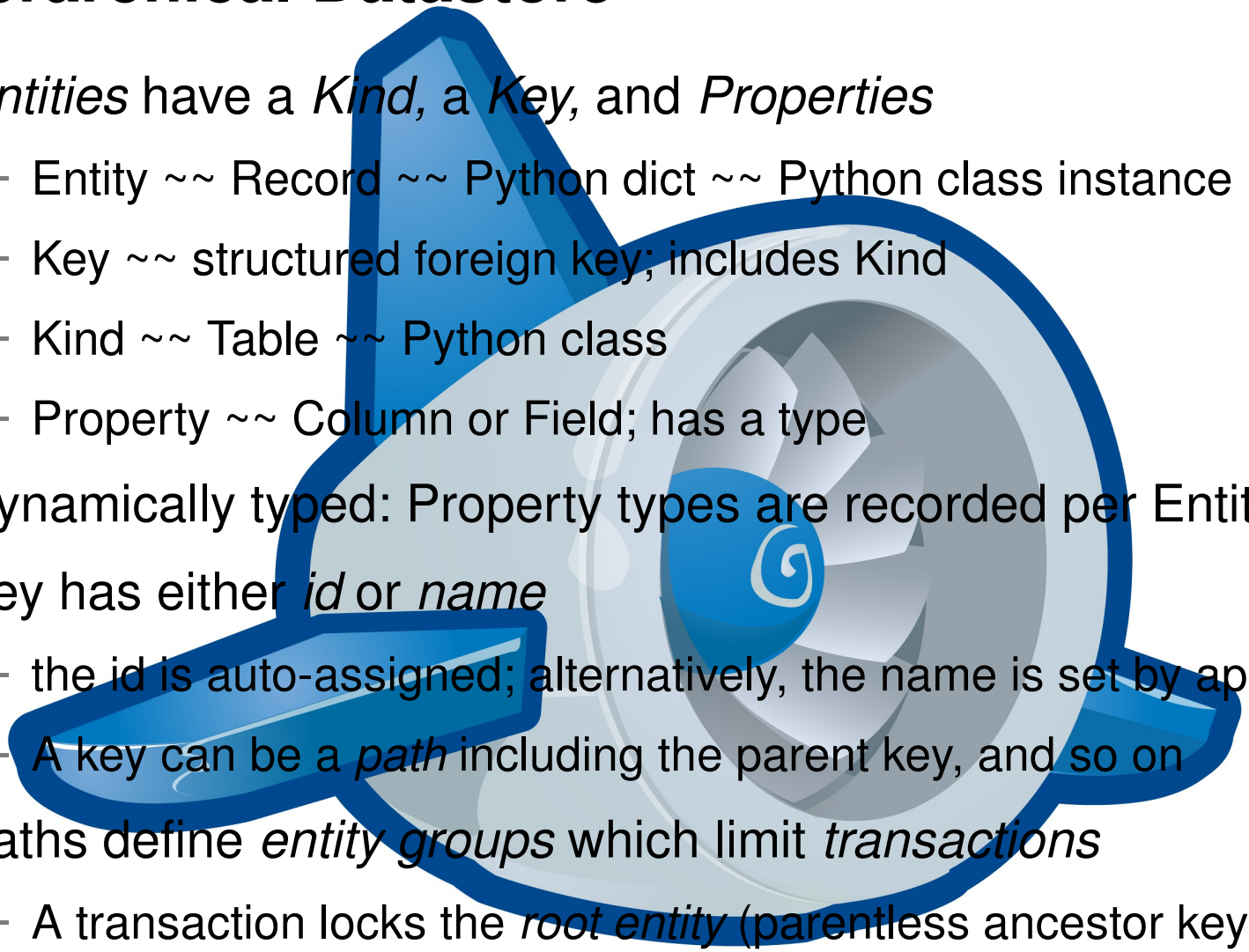


- You don't need to configure your resource needs
- One CPU can handle many requests per second
- Apps are hashed (really mapped) onto CPUs:
  - One process per app, many apps per CPU
  - Creating a new process is a matter of cloning a generic "model" process and then loading the application code (in fact the clones are pre-created and sit in a queue)
  - The process hangs around to handle more requests (reuse)
  - Eventually old processes are killed (recycle)
- Busy apps (many QPS) get assigned to multiple CPUs
  - This automatically adapts to the need
    - as long as CPUs are available

Preserving Fairness Through Quotas

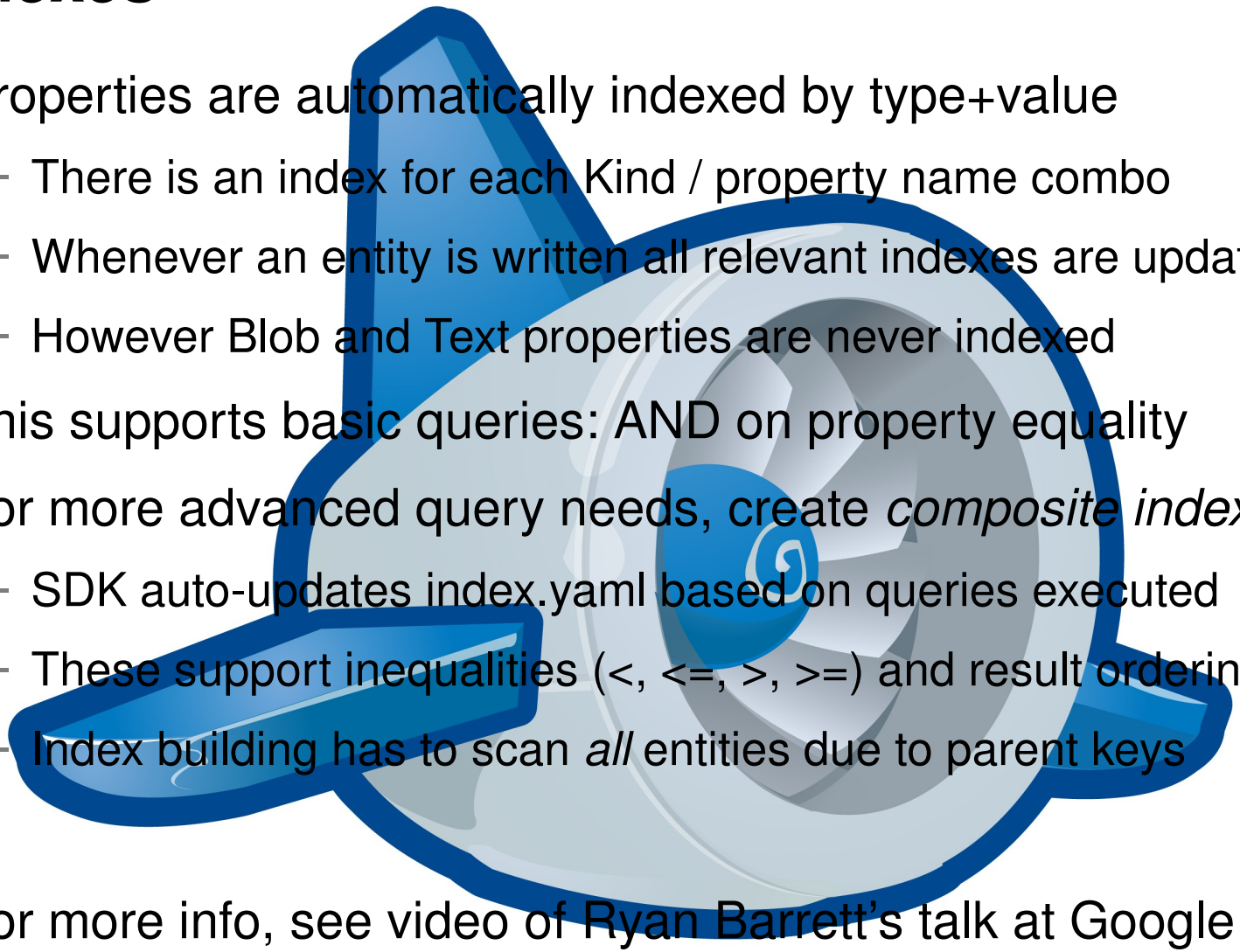
- Everything an app does is limited by quotas, for example:
  - request count, bandwidth used, CPU usage, datastore call count, disk space used, emails sent, even errors!
- If you run out of quota that particular operation is blocked (raising an exception) for a while (~10 min) until replenished
- Free quotas are tuned so that a well-written app (light CPU/datastore use) can survive a moderate “slashdotting”
- The point of quotas is to be able to support a very large number of small apps (analogy: baggage limit in air travel)
- Large apps need raised quotas
  - currently this is a manual process (search FAQ for “quota”)
  - in the future you can buy more resources

Hierarchical Datastore



- *Entities* have a *Kind*, a *Key*, and *Properties*
  - Entity ~~ Record ~~ Python dict ~~ Python class instance
  - Key ~~ structured foreign key; includes Kind
  - Kind ~~ Table ~~ Python class
  - Property ~~ Column or Field; has a type
- Dynamically typed: Property types are recorded per Entity
- Key has either *id* or *name*
  - the id is auto-assigned; alternatively, the name is set by app
  - A key can be a *path* including the parent key, and so on
- Paths define *entity groups* which limit *transactions*
  - A transaction locks the *root entity* (parentless ancestor key)

Indexes



- Properties are automatically indexed by type+value
  - There is an index for each Kind / property name combo
  - Whenever an entity is written all relevant indexes are updated
  - However Blob and Text properties are never indexed
- This supports basic queries: AND on property equality
- For more advanced query needs, create *composite indexes*
  - SDK auto-updates index.yaml based on queries executed
  - These support inequalities (<, <=, >, >=) and result ordering
  - Index building has to scan *all* entities due to parent keys
- For more info, see video of Ryan Barrett's talk at Google I/O

The Future

- Big things we're working on:
  - Large file uploads and downloads
  - Datastore import and export for large volumes
  - Pay-as-you-go billing (for resource usage over free quota)
  - More languages (no I'm not telling…)
  - Uptime monitoring site
- No published timeline – agile development process

Switch to Live Demo Now

Q & A

- Anything goes