# GAE Feedback

Distributed Systems 2015-2016

DistriNet
IMinds
KU LEUVEN

# Key Criteria

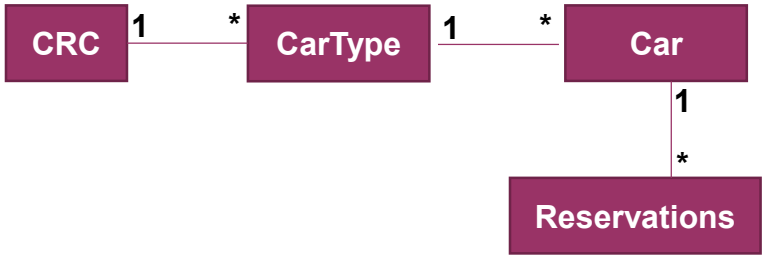
- 1. Persistence: GAE Datastore using JPA
  - Structure of entities in Entity Groups
  - Use of JPQL
- 2. Indirect Communication
  - Adapt from Direct Communication
  - Use of Task Queues, reason about data flow
- 3. Potential State Inconsistencies (theoretical exercise)
  - Why? How to avoid?
  - How to minimize impact on performance?

# General Impression

- Observation:
  - ✔ Persistence at GAE
  - ❗ Indirect communication
  - ❗ State inconsistency appearing in worker-based schema
- Hotspots for erros
  - JPA: attached vs. detached objects
  - Indirection == Time Decoupling
  - TaskQueue: Worker threads vs. processes
- Written answers
  - Some are quite good (short + complete), even providing alternative reasons
  - Others focus too much on the application and less on concepts and technology (usually the (too) long ones)

# 1. Persistence

- **Entity Group** (EG): entity relations as hierarchical **tree-structure**
  - one-to-many == parent-child relationship
  - every entity has at max one parent entity (tree)
- Easiest solution in car rental ageny
  - All entities in a single Entity Group
  - Use EntitiyManager instance per modification of EG
    - E.g. Multiple CRCs == multiple EntityManagers
  - No unmanaged relations between those entities
    - E.g. Manual lookup by UID (Primary Key)



# 1. Persistence (cont.)

- Transactions
  - Each EntityManager session == atomic commit
  - Don‘t use CrossGroup transactions (XGT)
- Google App Engine: JPQL
  - No JOINs supported
  - Use only to query on one entity

# 1. Persistence (cont.)

- JPA: Attached vs. Detached entities
  - Change is not persisted:

```
EntityManager em = EMF.get().createEntityManager();
List<Quote> q =  em.createQuery(..).getResultList();
em.close();
[…]
q.get(0).setStatus(Status.FAILED);
```

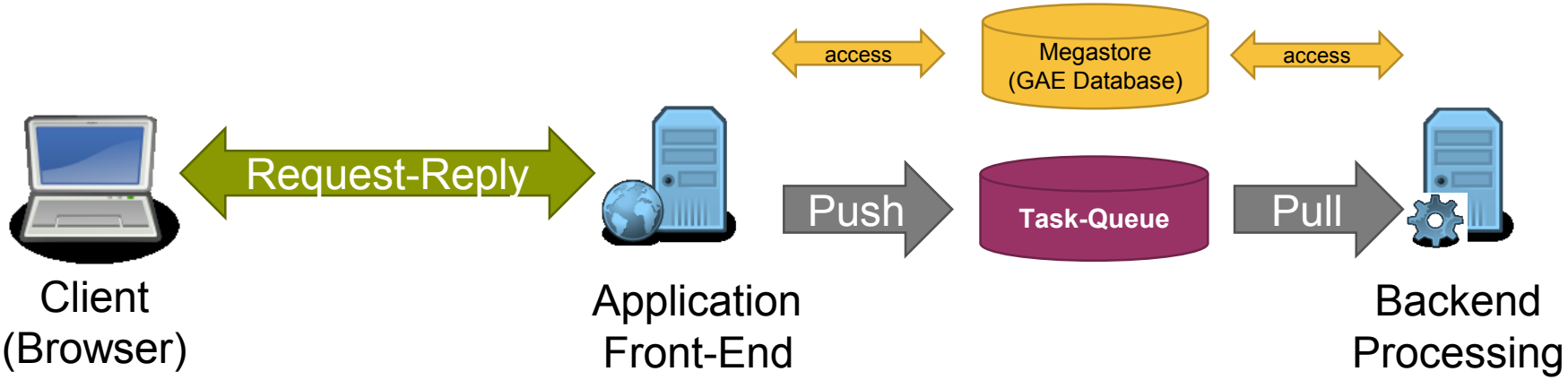
  - Change is persisted:

```
EntityManager em = …
List<Quote> q =  …
q.get(0).setStatus(Status.FAILED);
em.close();
```

```
em.close();
q.get(0).setStatus(Status.FAILED);
em=EMF.get()…
em.merge(q);
```

# 2. Indirect Communication

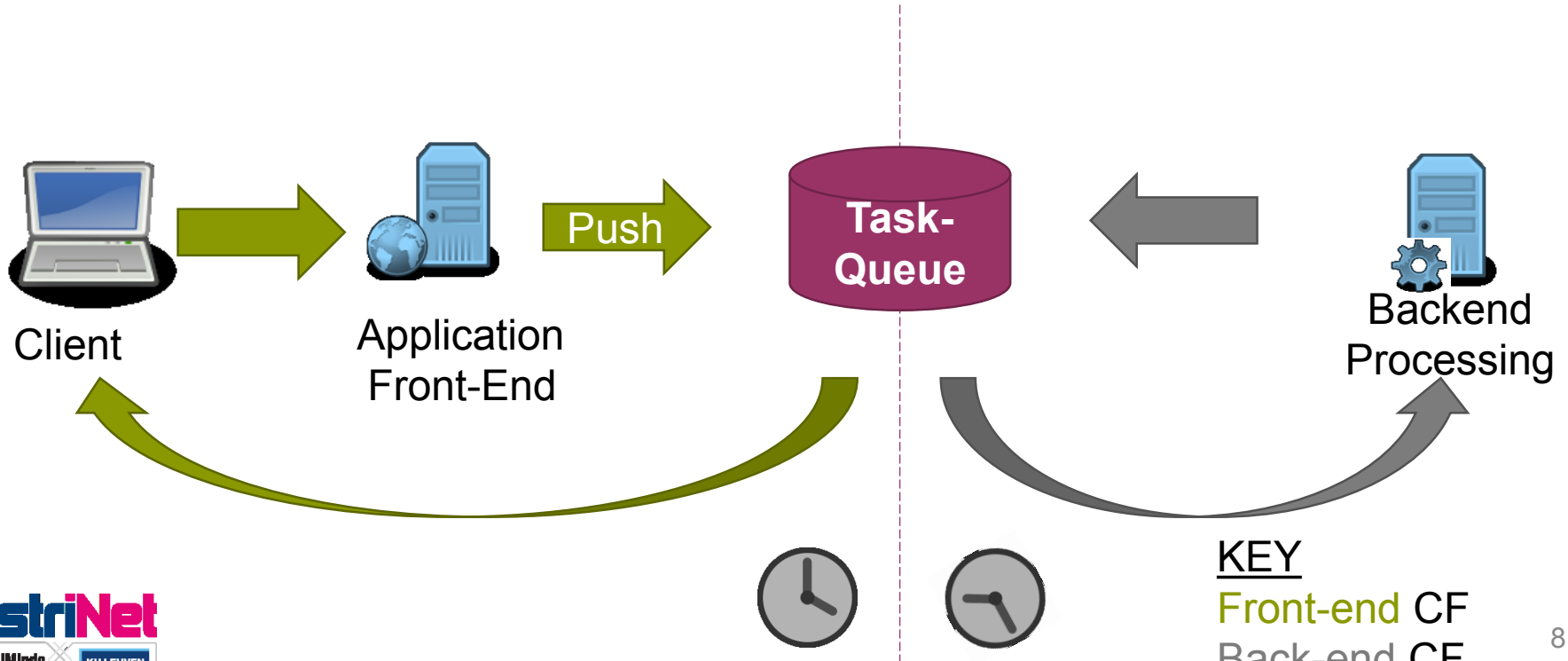
- Principle of Message Queues understood
- Confusion with the **3 roles** involved
  - Data flow:
    - Client (Browser) ← direct → Font-end Service
    - Font-end Service – via Queue → Back-end Service (Worker)




KEY
Direct Communication
Indirect Communication

# 2. Indirect Communication

- …
  - Control flow (CF):
    - Front-end Service triggered by Client
    - Back-end Service triggered by GAE infrastructure
    - (1) and (2) are **independent in time**



# 2. Indirect Communication

- Do not
  - Send msgs from back-end to front-end
    - client may be gone at time of message
  - wait with front-end until back-end completes
- Examples
  - no shared memory (shared „global“ variables )
  - no Channel API
- Backend workers use separate maschines
  - in own process (⇔ own thread)
  - no shared memory
    - shared „global“ static variables > won‘t work

# 2. Indirect Communication

- **Feedback channel** from Worker to Client
  - report about success AND failure
  - otherwise indistinguishable : not-yet-processed or failed

- Client may have **multiple** unconfirmed **quotes waiting**, or may have sent a rental order twice.
  - feedback should uniquely refer to quotes (e.g. using **orderID**)

# 3. Potential State Inconsistencies

- **State changes** (creation of reservations) happen only at the Worker role and **not at the front-end**.
  - Keyword ‚**synchronized**‘ for quote-enqueue function **no effect** on potential state inconsistencies.
- Multiple workers process tasks queue **in parallel** (default)
  - This is where potential inconsistencies are rooted
- Each worker is a separate process (separate JVM)
  - thread-based monitors (i.e. ‚**synchronized**‘) **cannot help**

# 3. Potential State Inconsistencies

- One solution
  - Set #worker per queue = 1
  - To increase parallelism
    - Maintain one queue per company
      (**Note**: only if client will book from single CRC)

- NOT
  - Create a queue per CarType or per Quote
    - → loss of all-or-nothing semanics
  - Create a queue for every task
    - japodizes state consistency even more (parallelism →∞)

# Written Answers

- Always: reason about your answer (unless it‘s certainly obvious)
  - Why did you choose tech a or option b?
  - Can you imagine drawbacks?
- Be to the point and don‘t loose the target
  - Asked: Is there a scenario for an inconsistent state.. ?
  - Expected: Yes, (the scenario is) when a does b.
  - Found: The application receives two requests […] Suppose a sequence like [..] now suppose another sequence […] As a result, [..] return OK, but […] is the updated value.
  - Result:
    - after 1 page text
    - not sure whether you mean yes or no?

# Written Answers

- Another DO-NOT example: deferred answer
  - If there are multiple threads running to confirm quotes. There is a possibility that two consecutive tasks that are in conflict, when multiple threads are running they can confirm the quotes at the same time.
  - Read: if you believe that multiple threads are in use, than yes, otherwise no

# Questions?