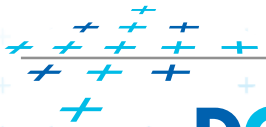
Introduction to clouds

Martin Klíma



DCGI

Cloud computing what it is

- New form of IT outsourcing
  - Replacement for server rental, webhosting, managed applications
  - Pay per use, well defined accounting
  - Self service
  - **Horizontal** and vertical scalability (vertical through virtualization)

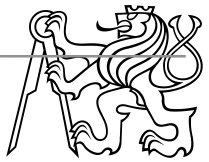
When do we want it?

- Public vs private cloud
- On premise – I have it all under my own roof
- Why not to have it on premise?
  - Licenses
  - Capacity designed for the worst scale
  - IT staff designed for 24 / 7 / 365

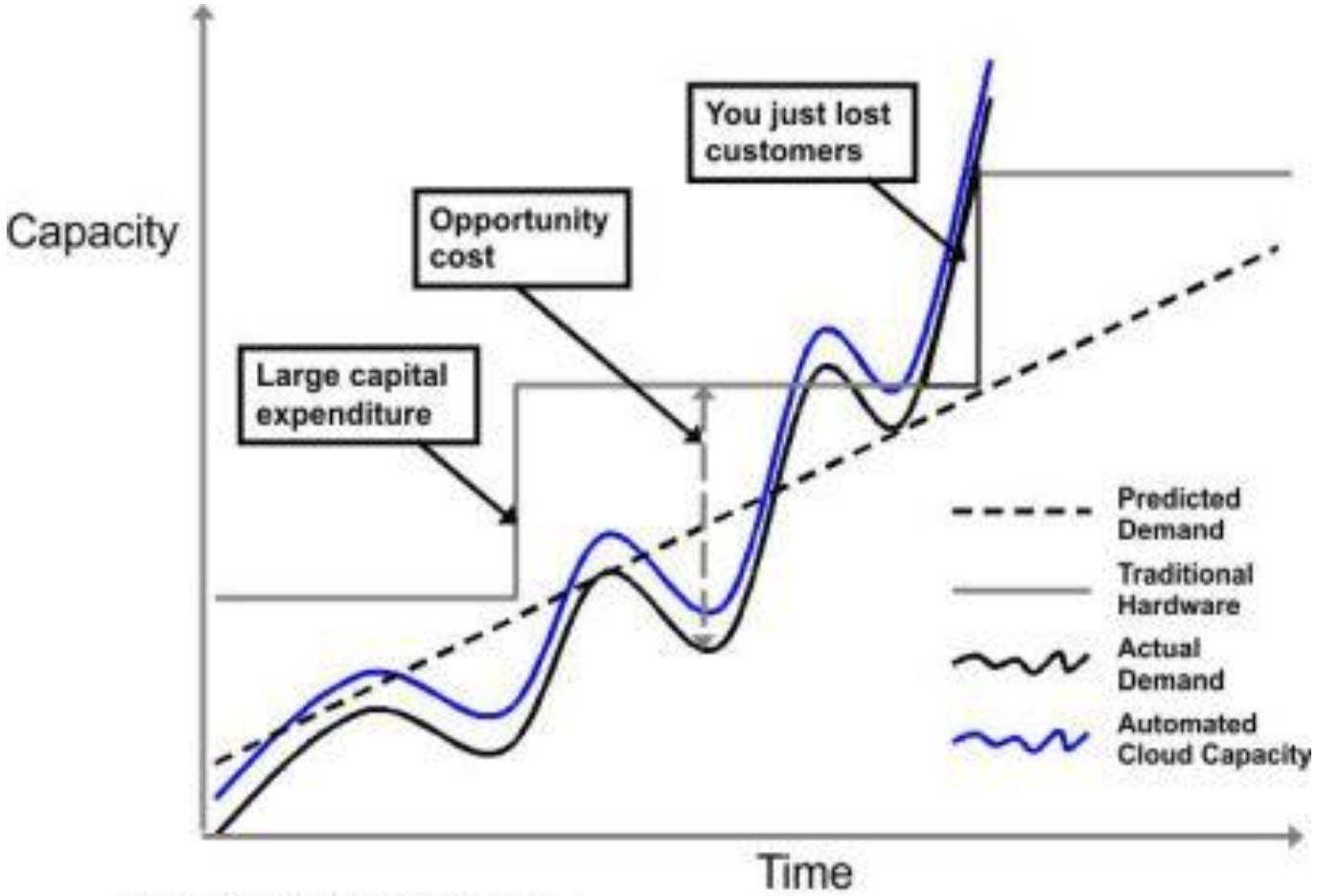
- Why not cloud
  - Data out of the company
  - Simple app may be costly

Where is the advantage?

- Try and leave
- Do not worry to scale
- Scale on demand
- Available anywhere
  - My business is getting worldwide

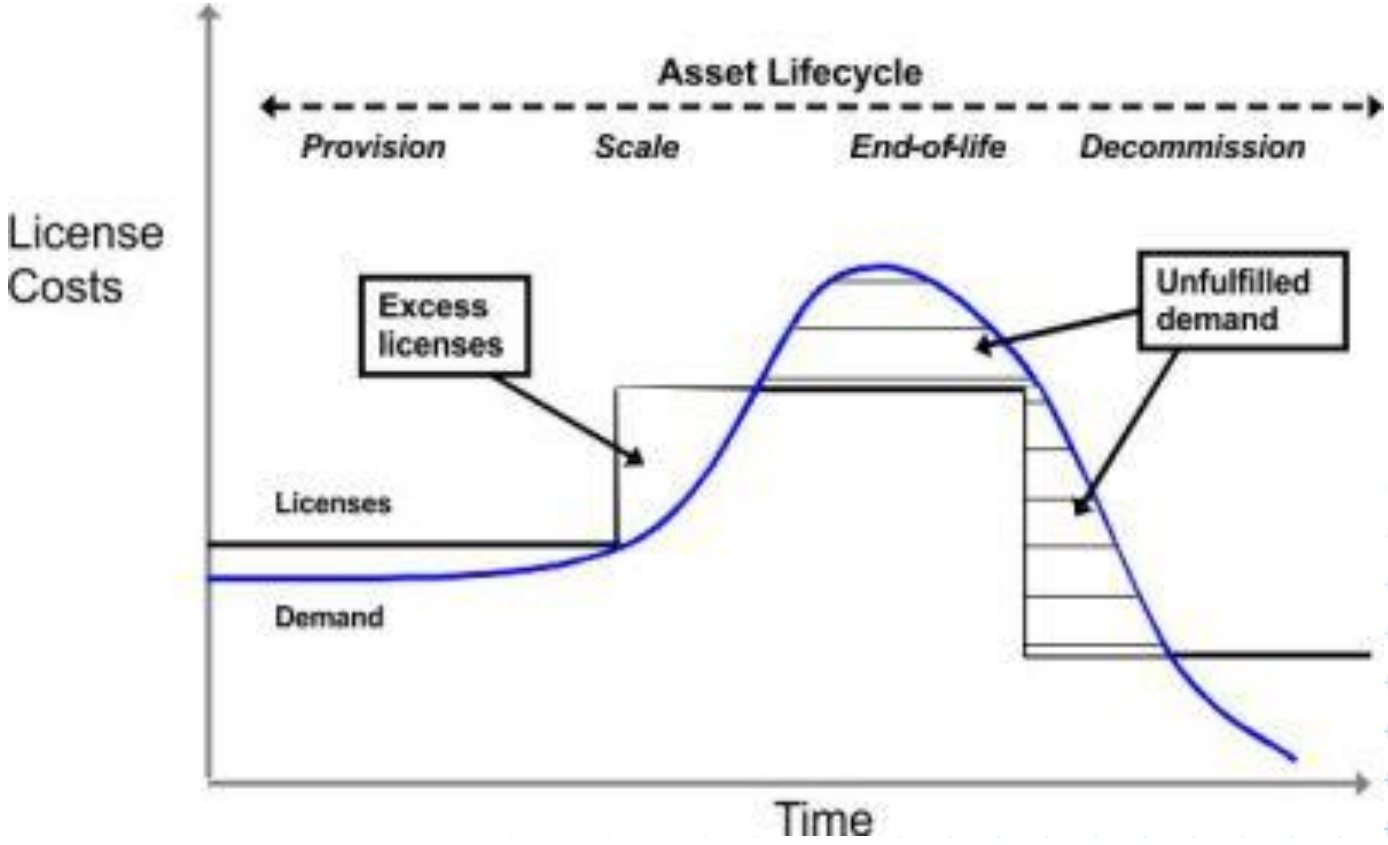


Scenarios

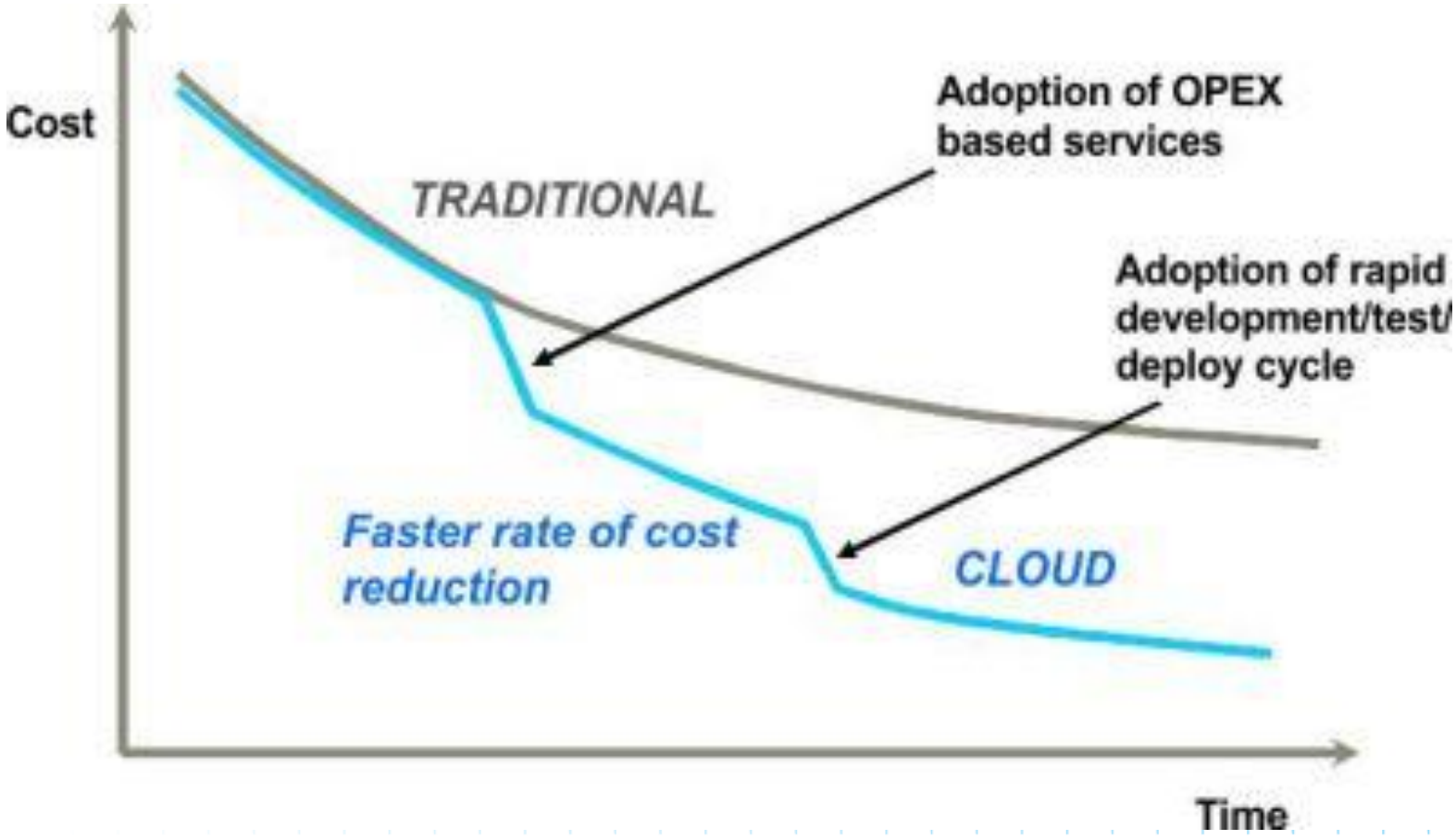


Based on illustrations by Amazon Web Services

Scenarios cont

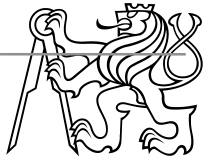


Scenarios cont



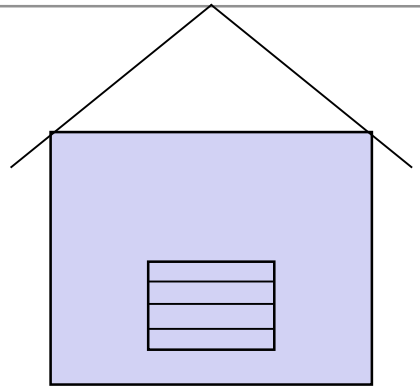
Types of cloud

- IaaS = Infrastructure as a Service
  - Rack Space
  - Amazon AWS
- PaaS = Platform as a Service
  - Google app engine
  - Microsoft Azure
- Software as a Service = SaaS

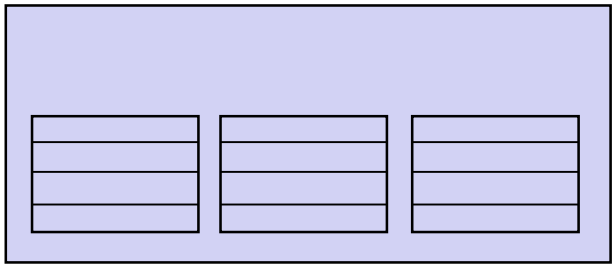


Cloud vs on premise

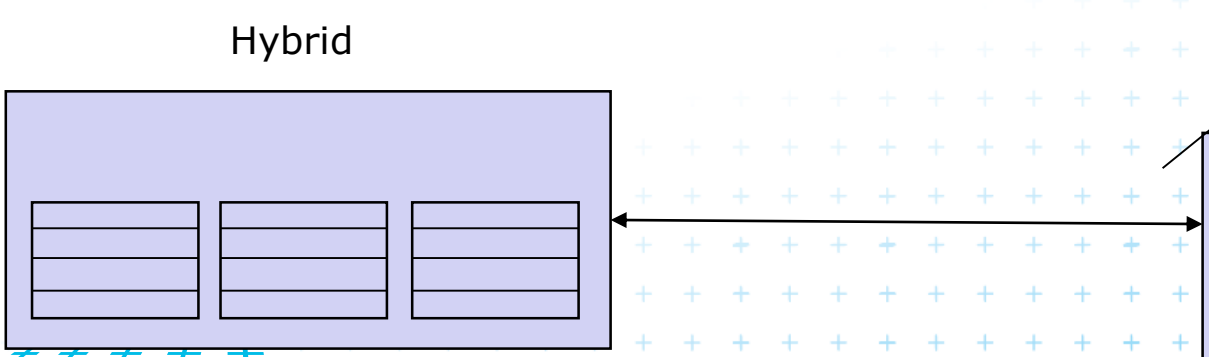
On presmise

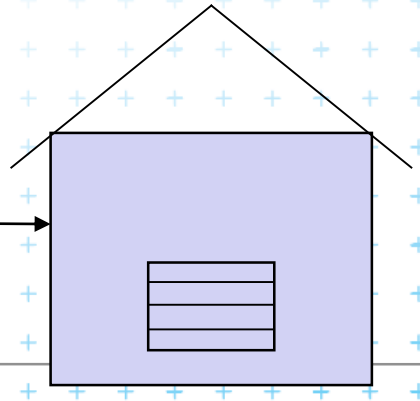


Cloud



Hybrid





Uptime and SLA

- SLA = Service Level Agreement
  - 99.9%, 99.99%, 99.999%

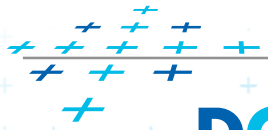
- IT Disaster recovery – 2-4% of budget
- Secondary recovery center

- RPO = Recovery Point Objective – maximum period of time for which data can be lost >>> zero
- RTO = Recovery Time Objective – maximum amount of time that my business can sustain without the data

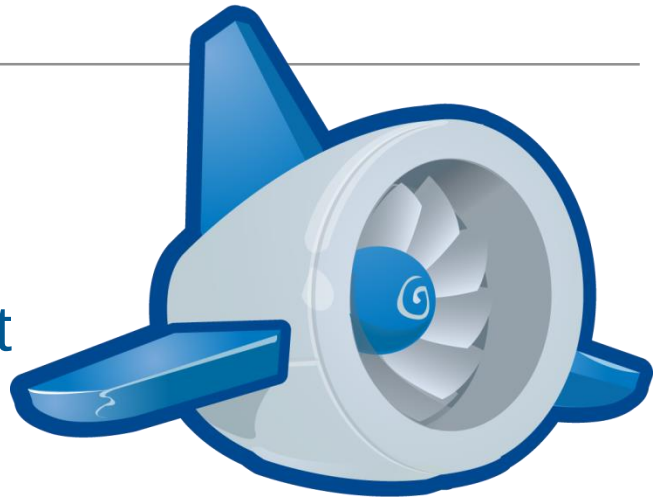
Virtualization

Virtualization is a very effective method for resource utilization.

Google App Engine - GAE



Google App Engine



- Google – well known ☺
- Large datacenters
- Sells computing power that it does not need for itself
- The initial design of the datacenter is web analysis, harvesting, indexing, searching

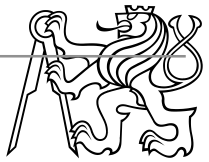
Core google functions

- Googlebot, crowlers
  - Visit a web page, download it
  - Google download a vast majority of visible web
  - Search for new resources automatically
  - A new resource can be added manually
  - Sophisticated logic to discover cheating
    - Hidden text
    - Meta data
    - Different content for bots and other users
  - Various policies for vising
    - How often to come back and download?
  - How to match the same pages on different servers?

Core Google functions

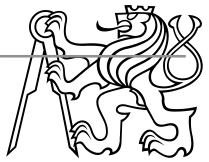
- Google Indexer
  - Builds an index of existing words and urls of pages
  - Some words are not indexed (conjunctions, prepositions, …)

- Google Query Processor
  - Uses *PageRank* of pages
    - The more URLs pointing to the page, the higher *PageRank*
    - The more trusted page points, the higher *PageRank*
    - *PageRank* considers about 100 other aspects of a page
    - Google keeps them secret
  - Use of spelling corrector
  - Prefers terms that are physically near to each other



Core Google functions

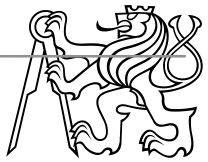
- Google Doc Servers
  - Stores the data itself
  - HUGE amount of data
  - Caching of history of internet



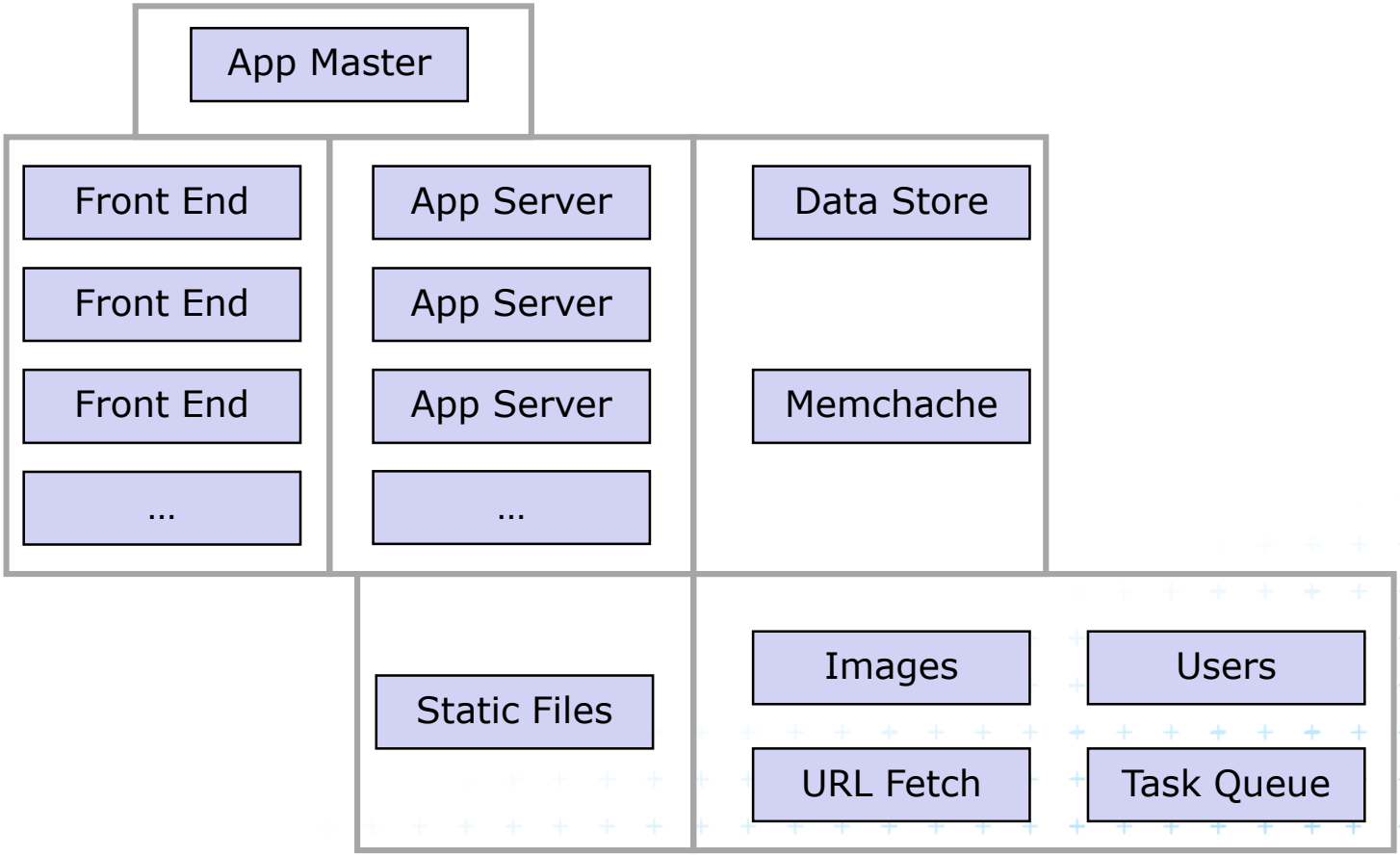
Google App Engine

Offers the basic infrastructure

- Scalability
  - To scale up/down resources when needed
- Reliablity
  - Capability to survive failure, recovery



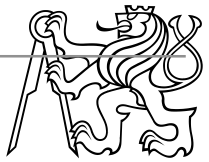
Infrastructure



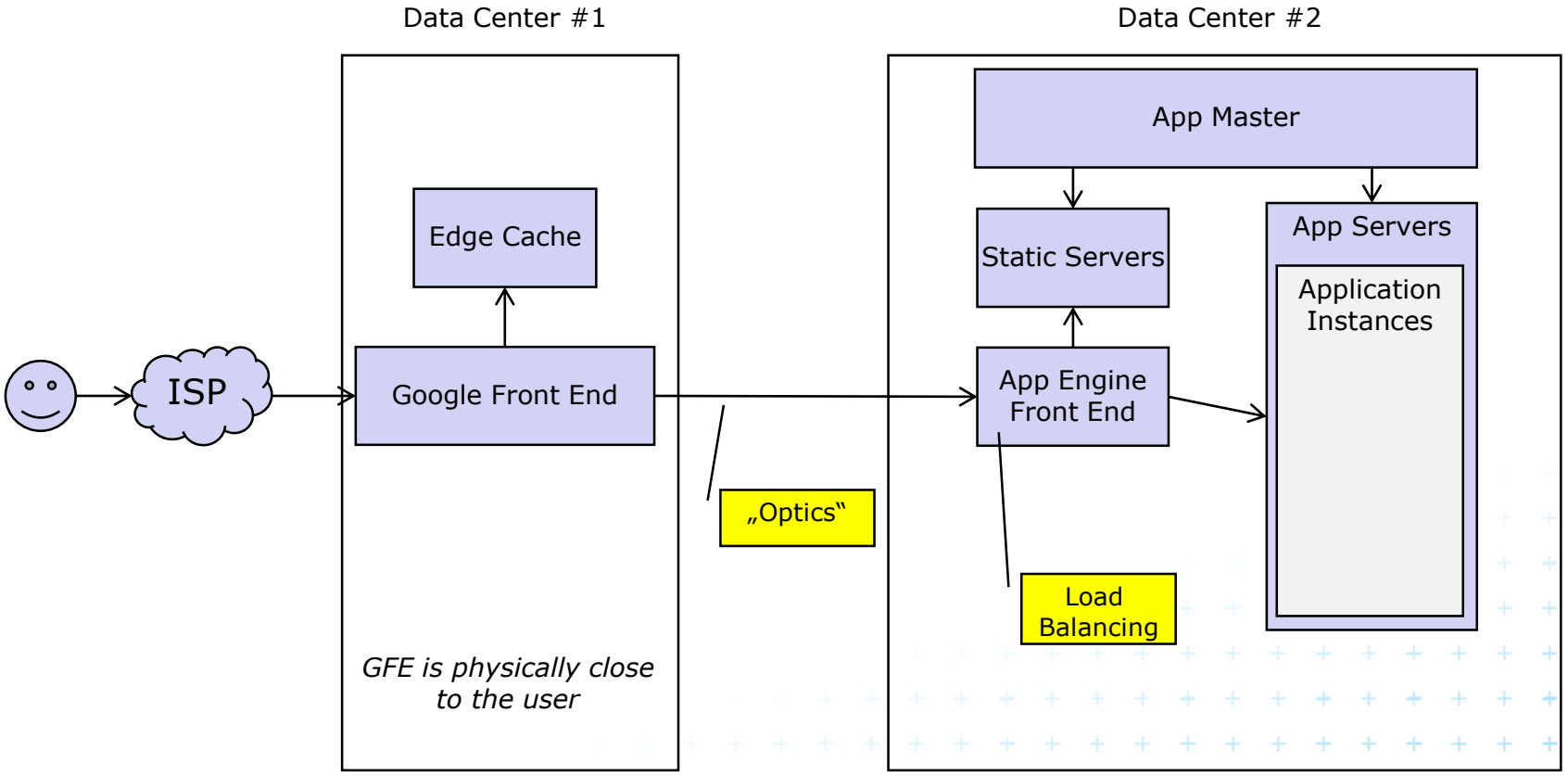
Best practice

- Non-Relational DB BigTable

- App design
  - Fast request handling
  - Low resource utilization
  - Fyzical HW independence



Google Front End



Using Edge Cache

1. HTTP headers

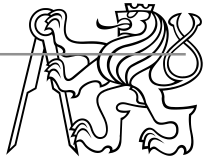
   …it is in fact a HTTP cache ☺

```
class MyHandler (webapp.RequestHandler):
        def get(self):
            self.response.headers.add_header(
                'cache-control',
                'public, max-age='7200') #2 hodiny
```

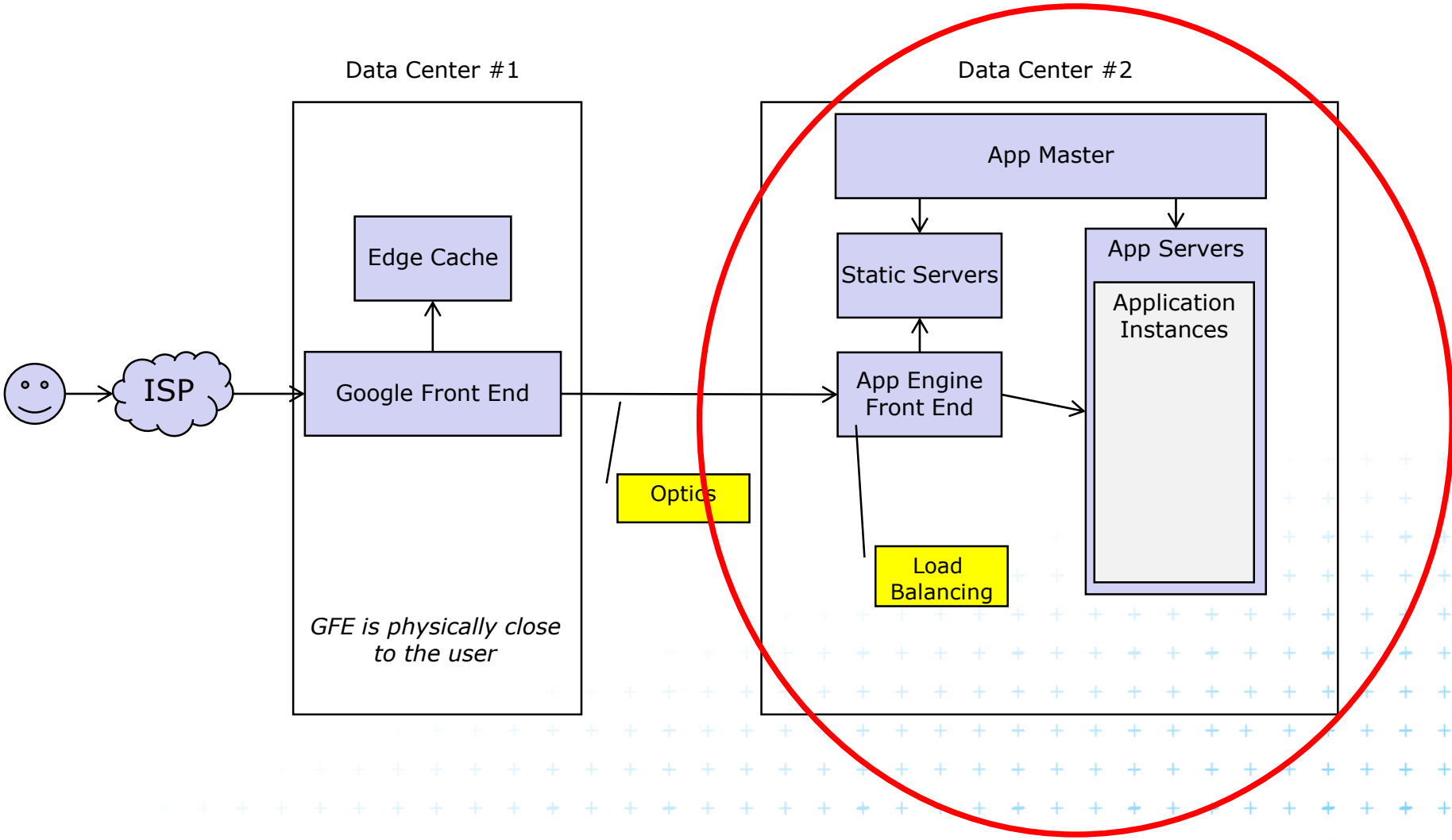
2. Define content as static

appegnine-web.xml

```
…
<static>
<include path="/**.png" />
<exclude path="/data/**.png" />
</static>
```

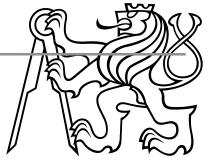



App Server


Data Center #1
Data Center #2
App Master
Edge Cache
Static Servers
App Servers
Application Instances
ISP
Google Front End
App Engine Front End
Optics
Load Balancing
GFE is physically close to the user

App Server

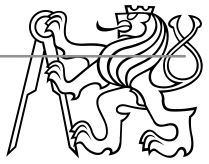
- Application running environment
  - Sandbox
  - Dedicated memory
  - Container controls the app live cycle
  - App does not see out of container
- App master
  - Monitors how apps run
  - Scales them up/down
  - Makes instance visible to each other



App Server – types of instances

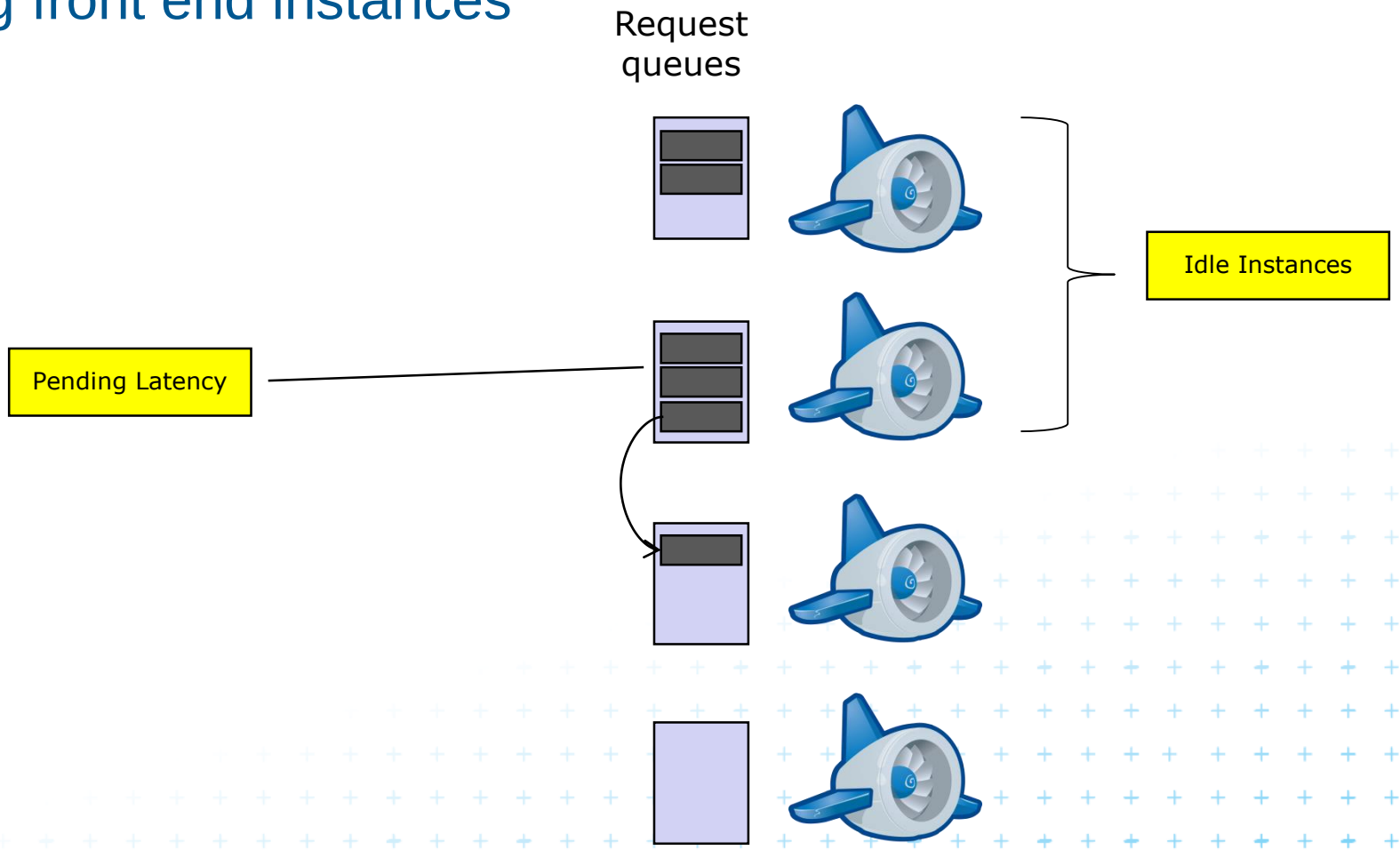
- Front End instance
  - Good for fast request serving
  - Max lifetime 60 sec
  - Typical for web application
  - Stateless, easy to scale
- Back End instance
  - Good for larger computing tasks
  - Stateful
  - Batch data processing, not time limitted
  - Costly, difficult to scale
- Communication
  - Queues

Careful with big libraries



App Server – Scaling

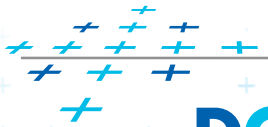
- Scaling front end instances



App Engine parameters

- Pending Latancy
  - Time a request spends in a queue
  - Over a given threshold a new front end instance is created
  - The longer PL, the worse the response time
  - The shorter PL, the more expensive ☹

- Iddle Instances
  - How many instance do we keep when there is no utilization
  - Stand-by readiness



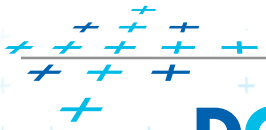
DCGI

WA 2

Google High Replication Data Store

Martin Klíma

Zdroj: http://www.youtube.com/watch?v=xO015C3R6dw

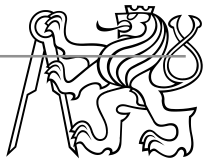


DCGI

The main data storages in GAE

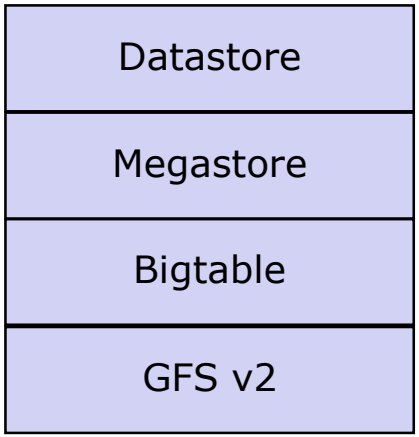
- Master/Slave
  - One master node
  - Number of slaves replicating the content o master

- High replication
  - No central/control node
  - All instances are equal



Datastore Software Stack

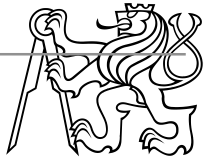
- App Engine Datastore
  - Schema-less storate
  - Advanced query engine
- Megastore
  - Multi-row transactions
    - Across multiple machines
    - Entity Groups
  - Simple indexes/queries
  - Strict schema
- Bigtable
  - Distributed key/value store
- Next generation distributed file system



Entitity Group

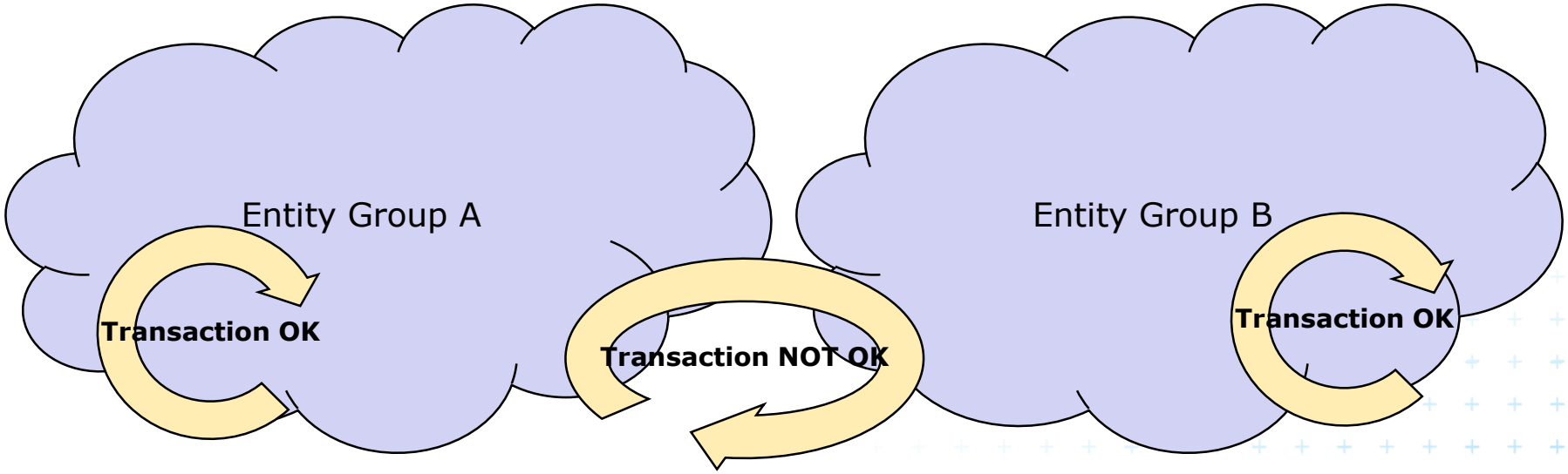
- Logical grouping of entities
  - Parent/child key relationship
- Unit of Transactionality
  - Transactions can only read/write entities in a single grup
- Unit of consistency
  - Strong serial consistency

What you write you will always get.
No partial success of transaction

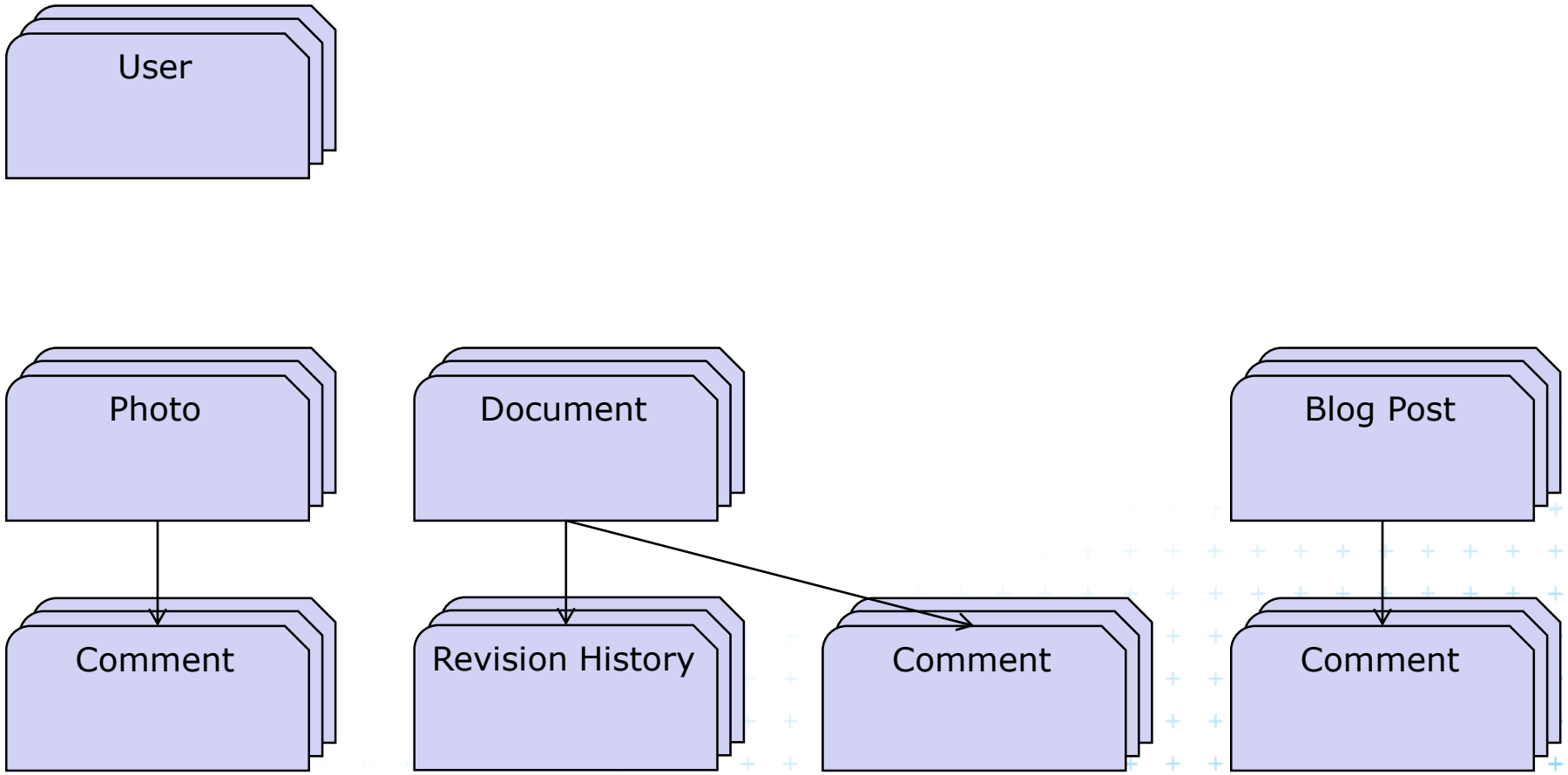


Entity groups

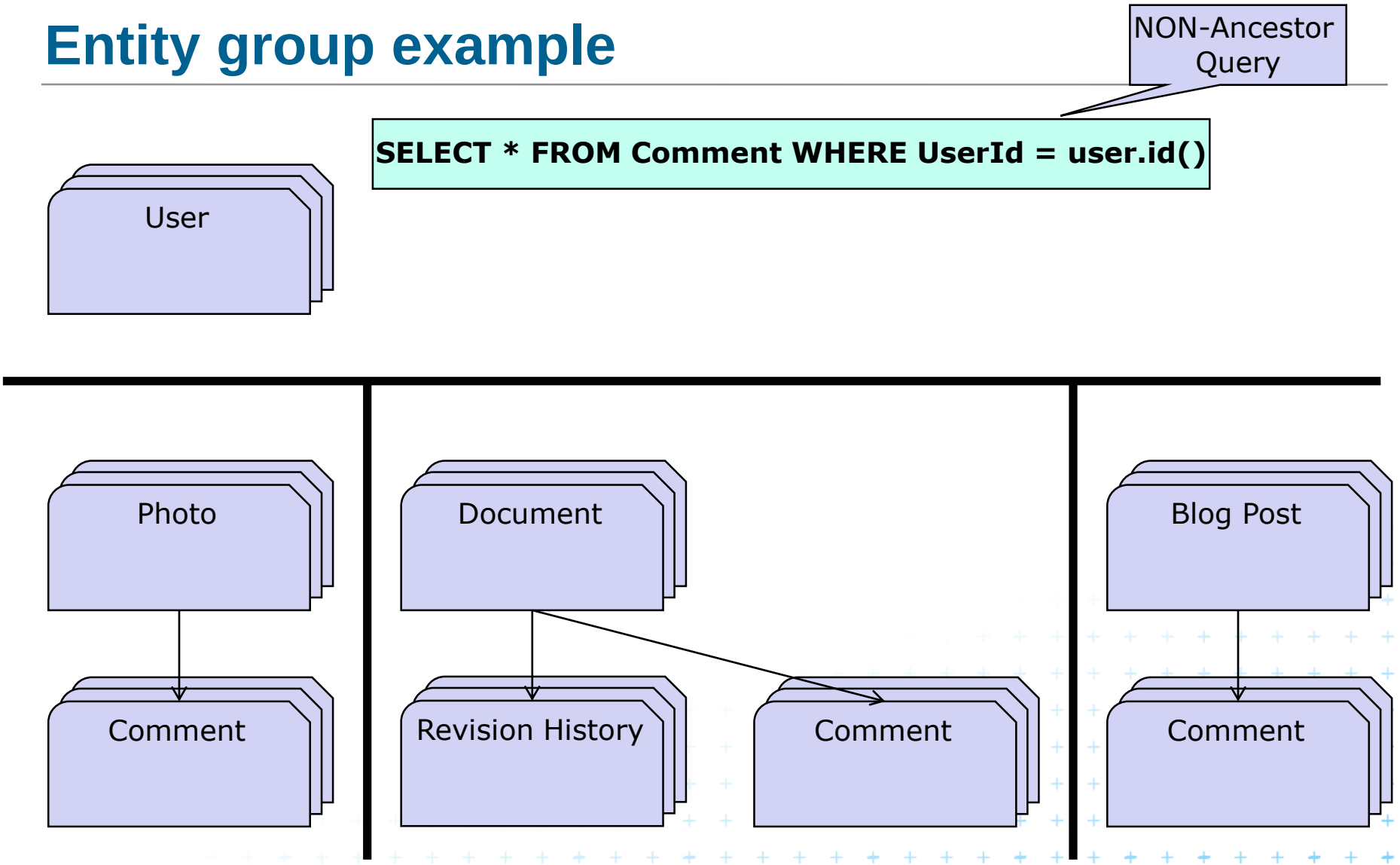
- Transaction on one entity group are guaranteed
- Transaction across entity groups are not quaranteed



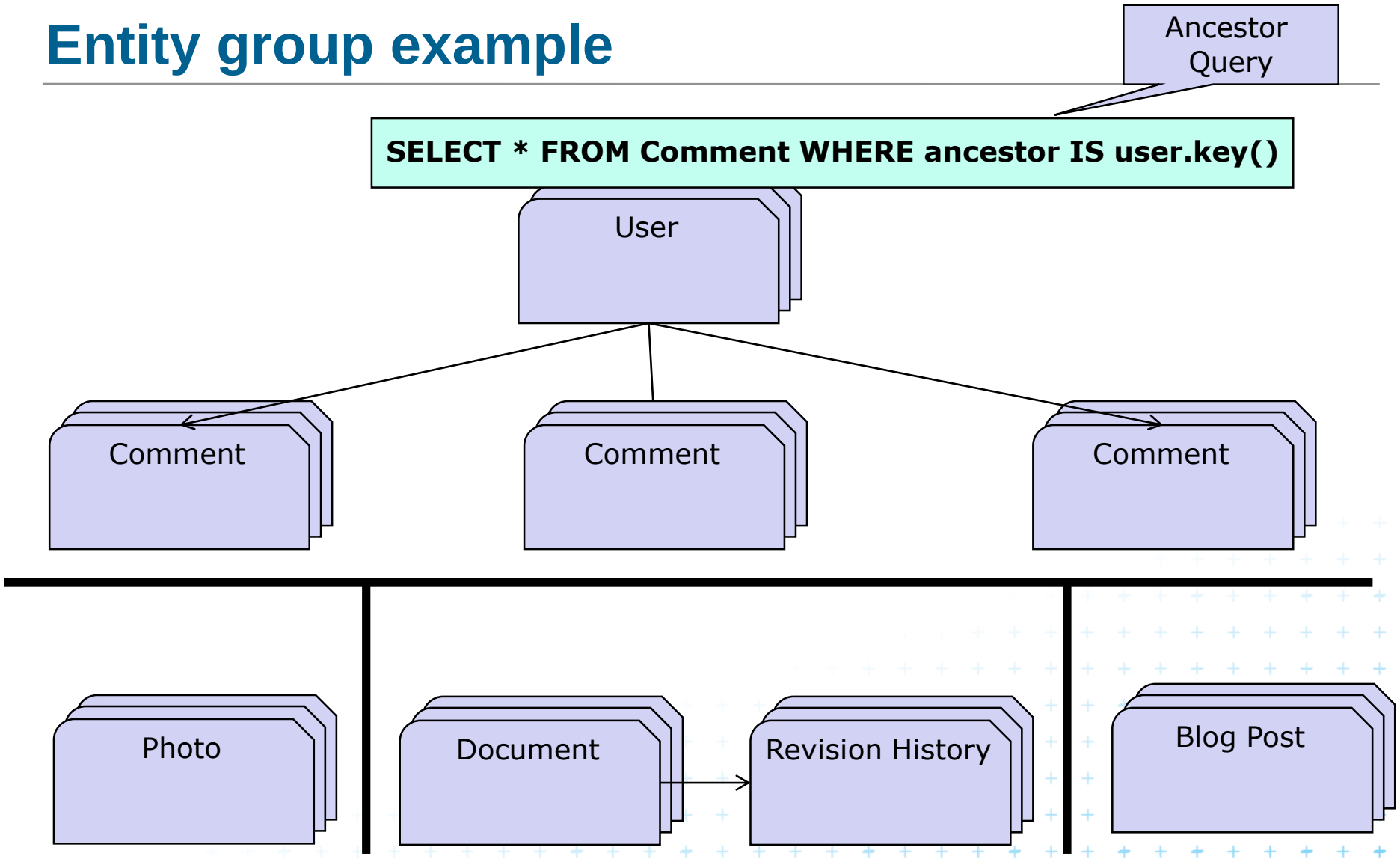
Entity group example



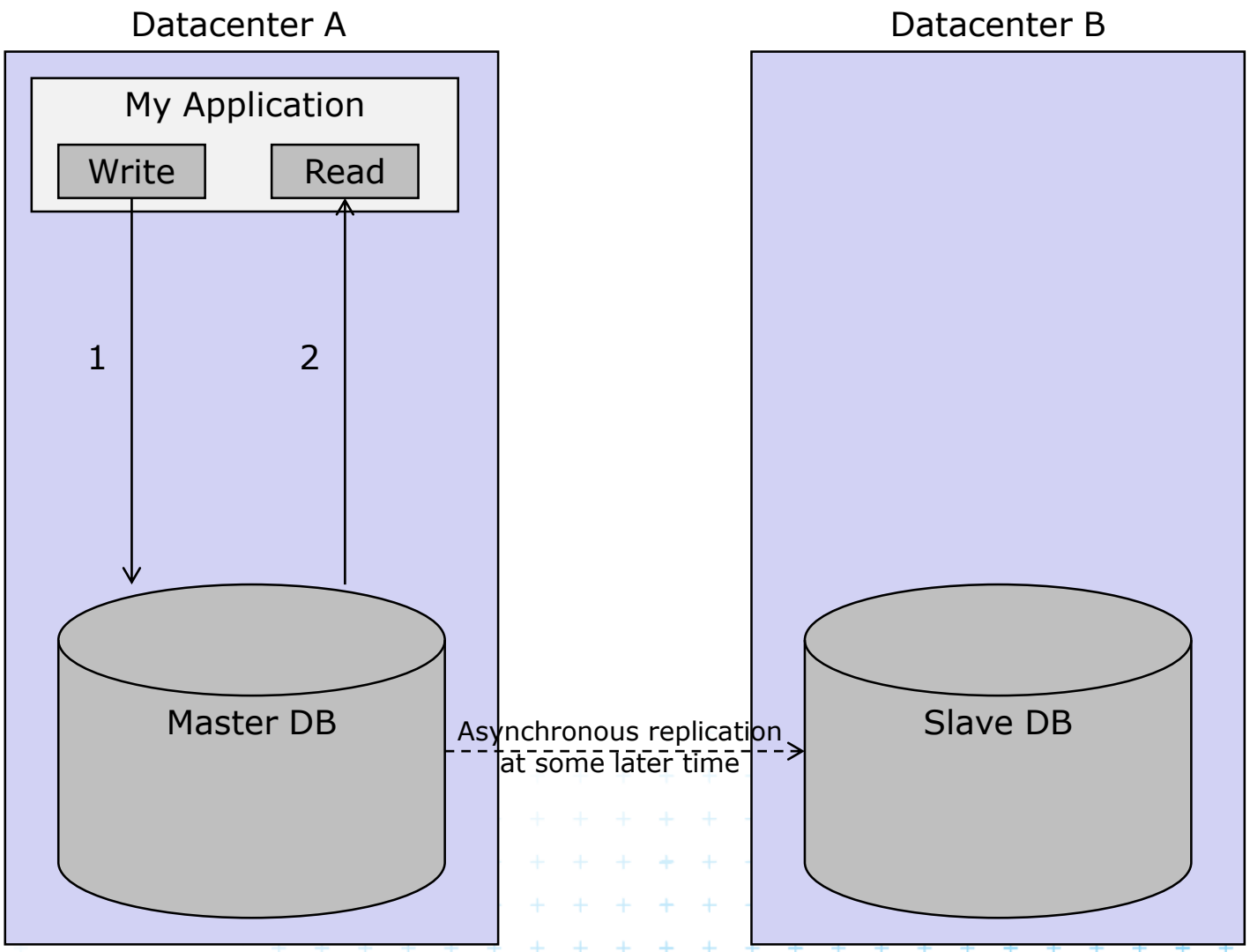
Entity group example



Entity group example



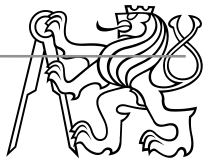
Master/Slave write/read model



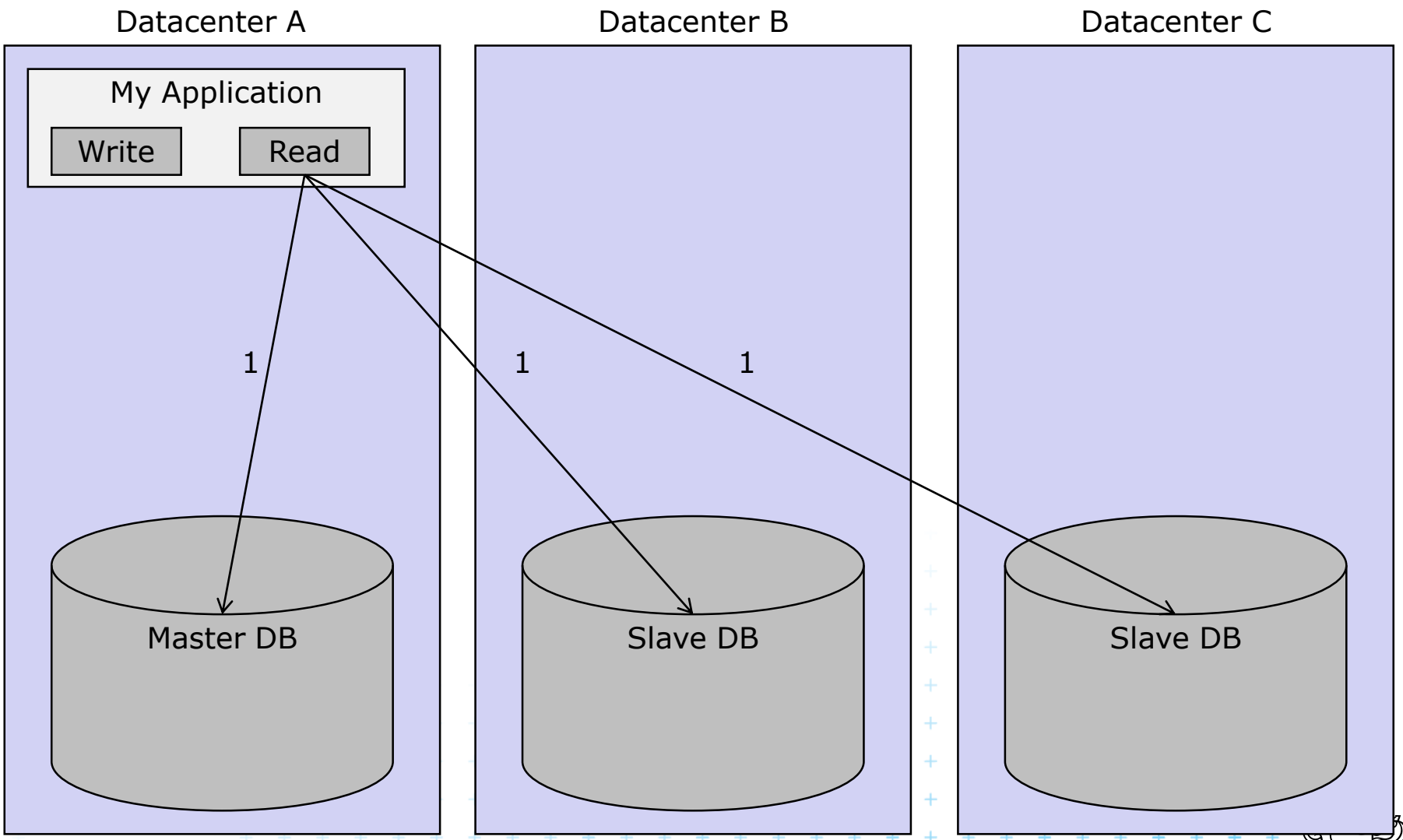
My Application can see all its writes because they were made to its Master DB

High Replication engine

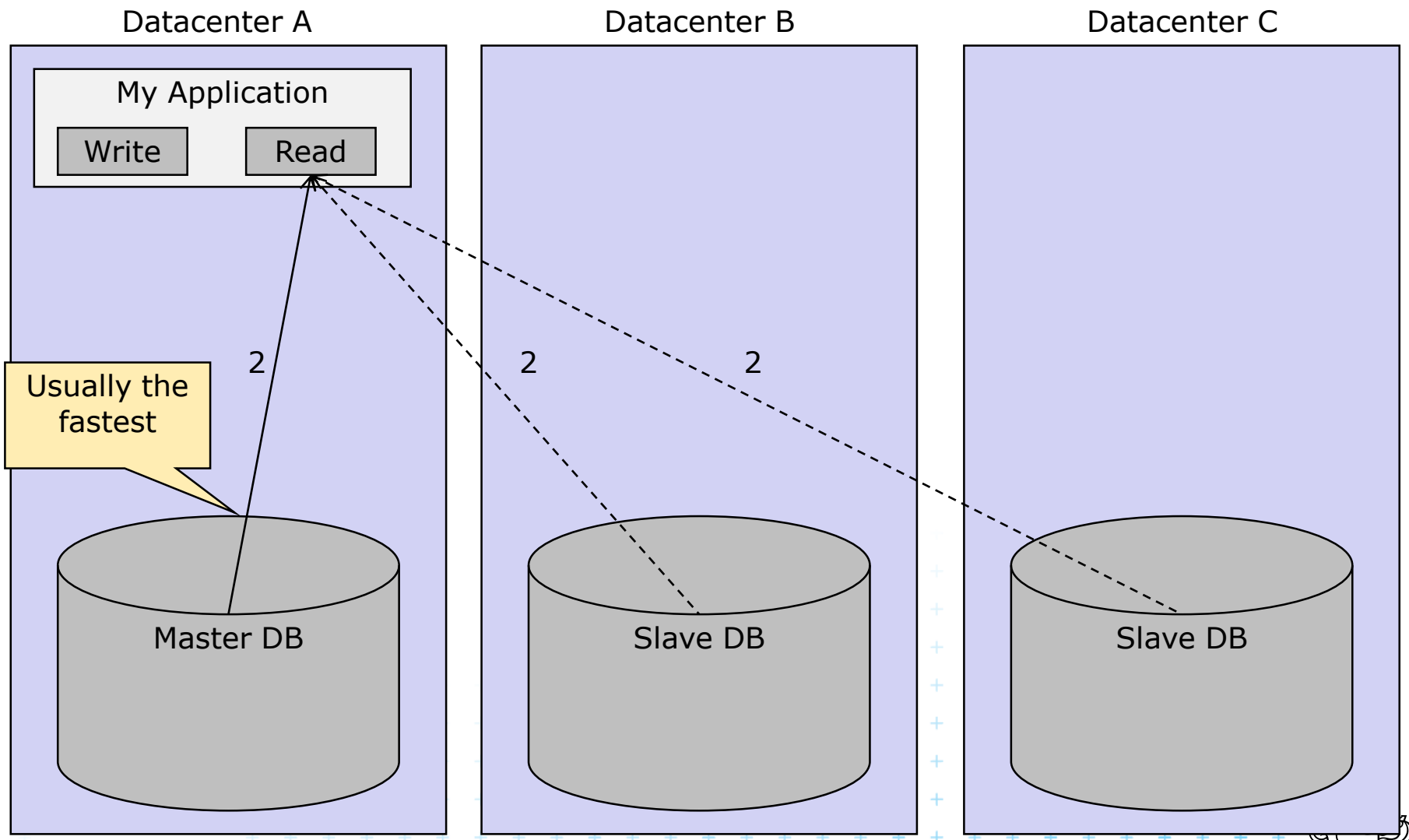
- Write
  - Write to at least majority of nodes
  - Minority may not get writes synchronously
  - Asynchronous replication
  - On demand replication
- Read
  - Read from fastest (mostly local)
  - Catch up on demand



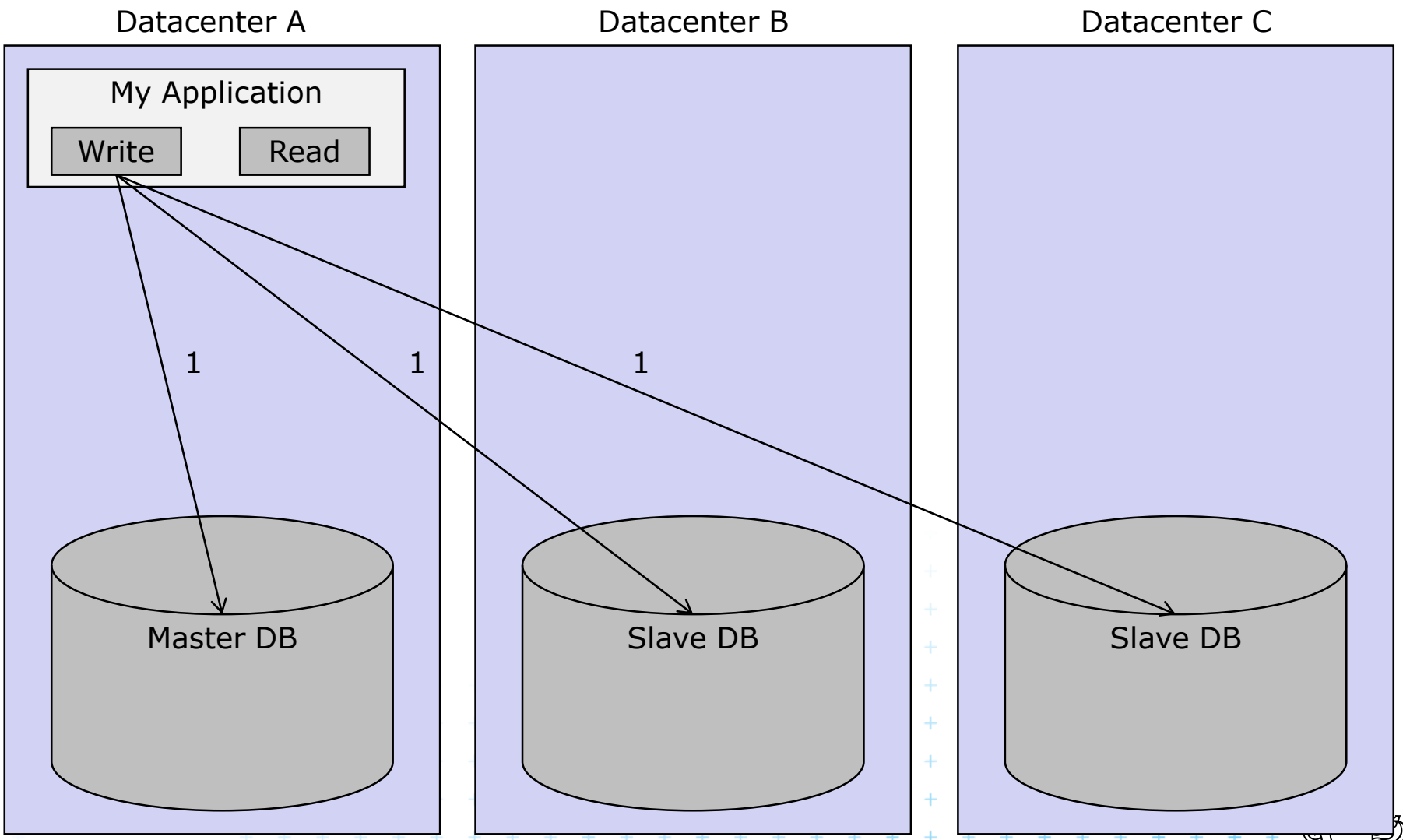
High replication read model



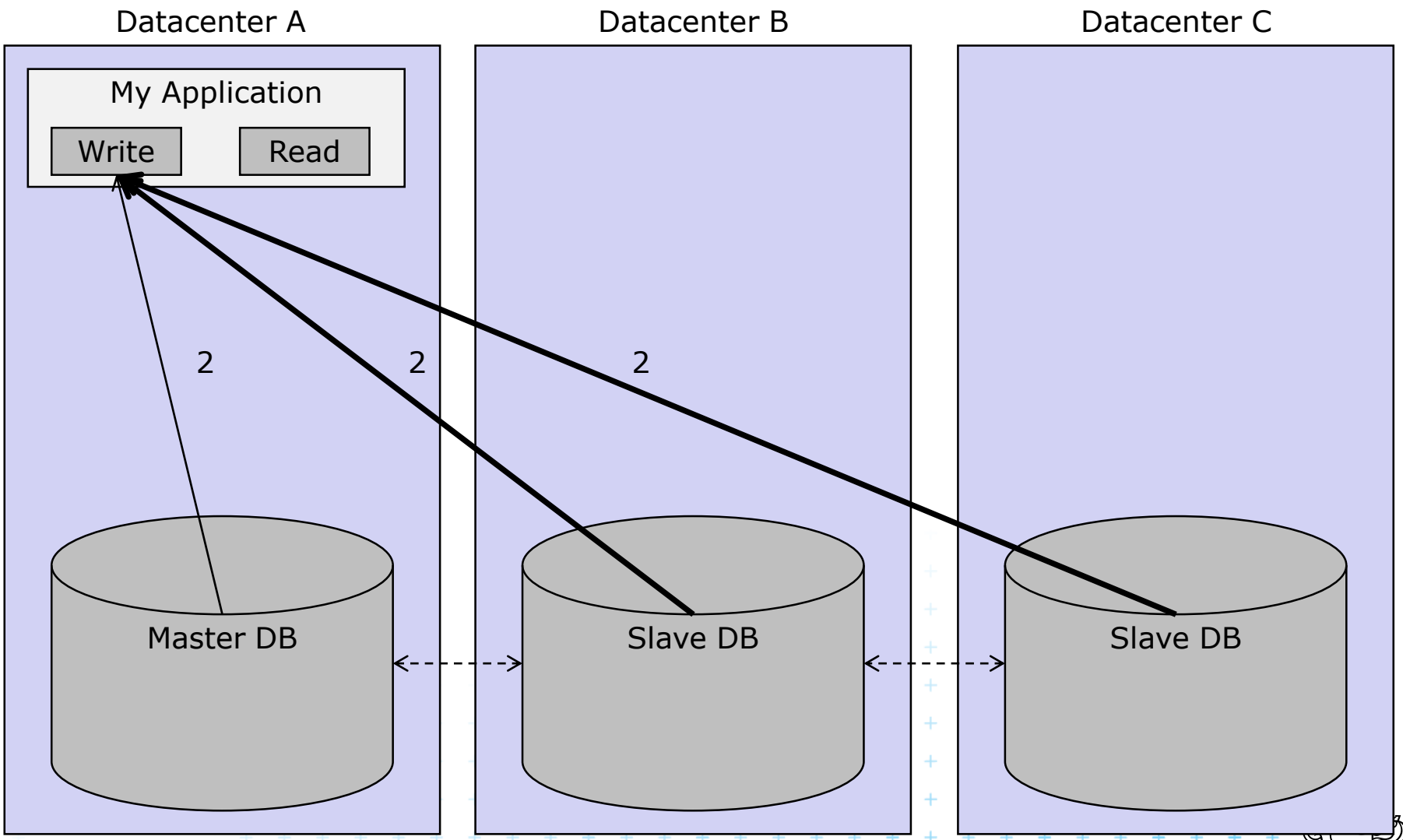
High replication read model



High replication write model

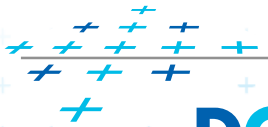


High replication write model



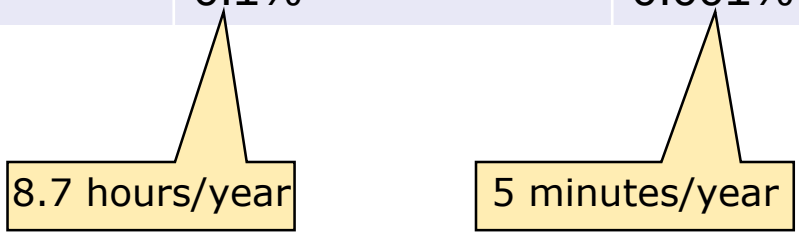
High replication

- There is no guarantee that a given database has all the written data at a given time.



Datastore performance

		Master/Slave	High Replication
Average Latency	Read	15 ms	15 ms
	Write	20 ms	45 ms
Average Error Rate	Read	0.1%	0.001%
	Write	0.1%	0.001%

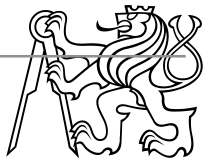


SLA !!!

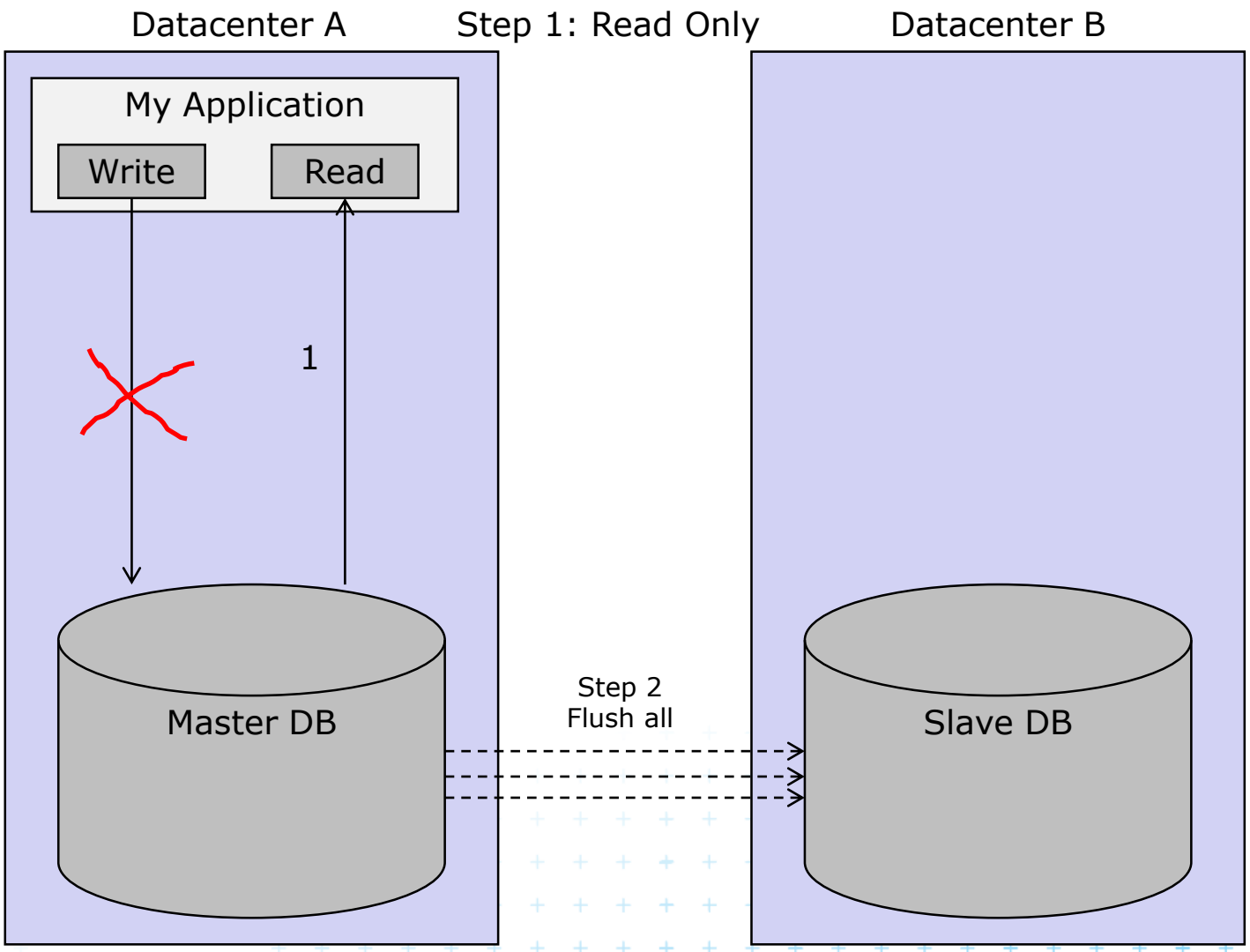
Maintenance

- Master/Slave
  - Switch master
    - One hour of read-only datastore

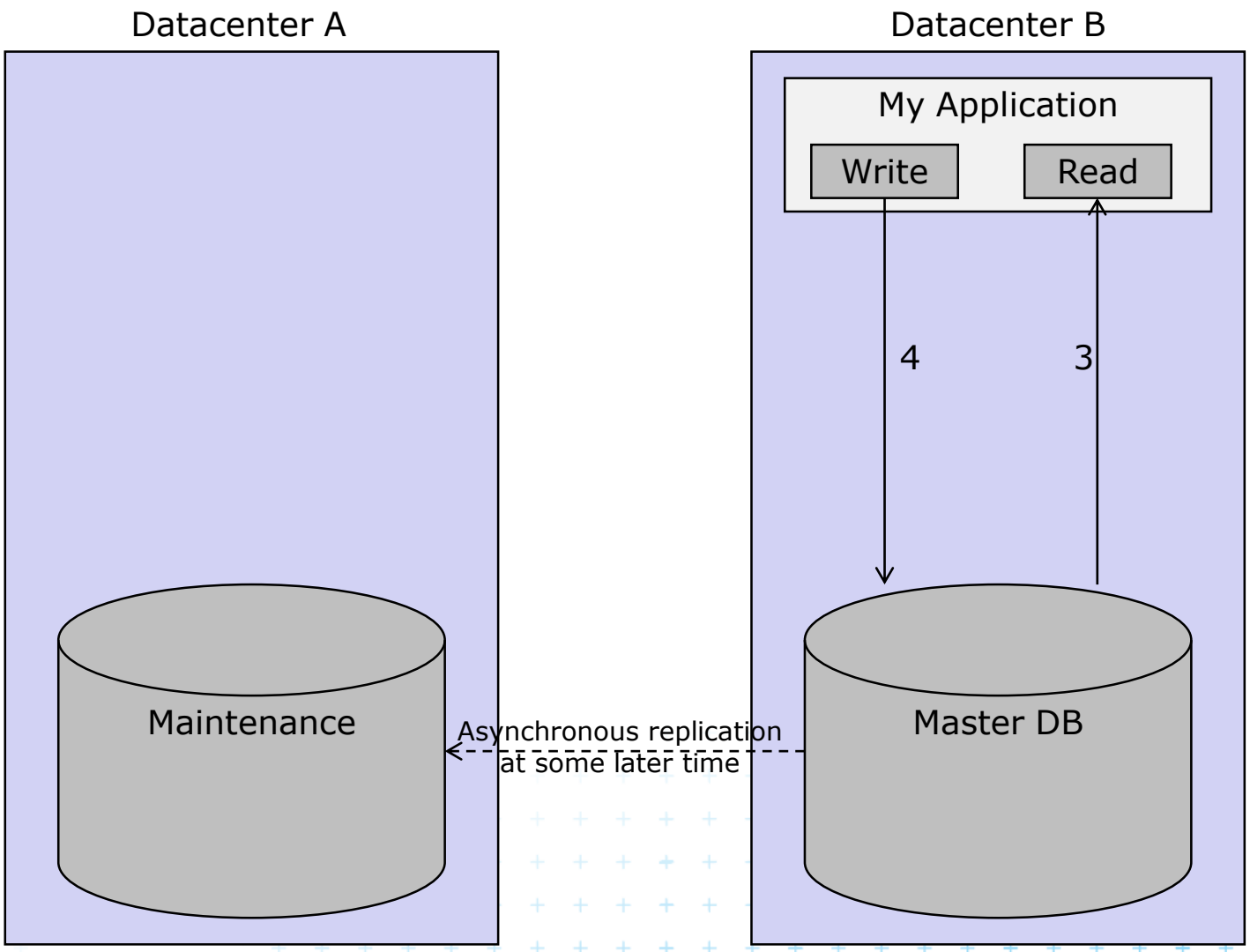
DCGI



Master/Slave maintenance



Master/Slave maintenance



High replication

- Almost not affected by maintenance time
- Memcache flush (1 minute)