Enterprise Architecture Case in PHP

2015-10-09

Ant
yftzeng@gmail.com

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

學架構的優點是，會變帥；

但缺點是，帥的不明顯。

❶ 架構先決

無視人員、流程只講技術，是要白痴

架構會影響公司文化、商業擴展；思維更要超越程式碼層次

❷ 沒有完美的架構，只有最適的架構

無視場景只講架構，是耍流氓

若無法舉出架構的缺陷，代表你還無法掌握

盲目套用別人的架構，並不會讓你變得跟他一樣好

High throughput

Low latency

千萬人同時在線

電子商務、線上媒體

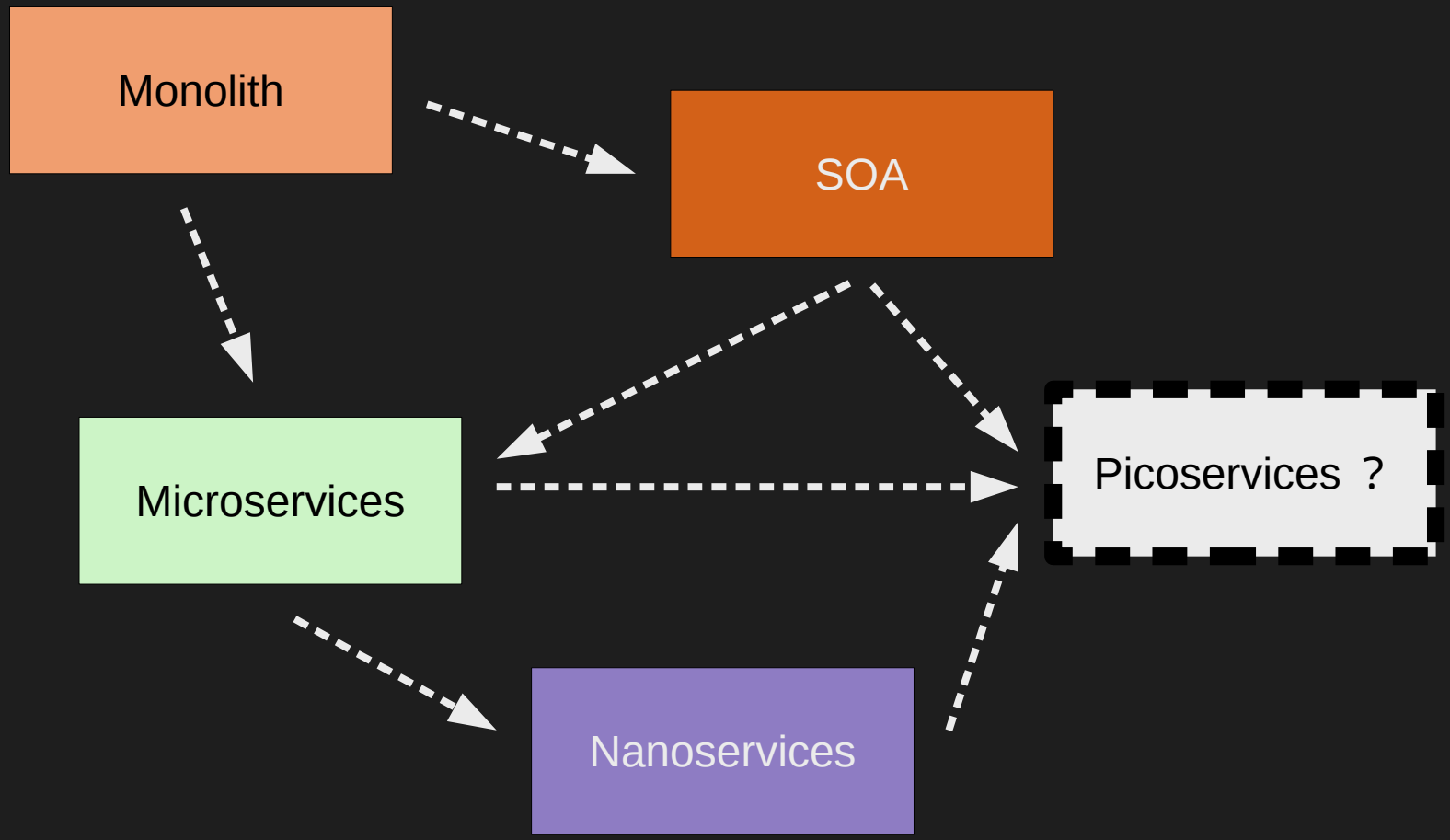
低延遲回應

廣告平台 (30ms)、高頻交易 (0.5~3ms)

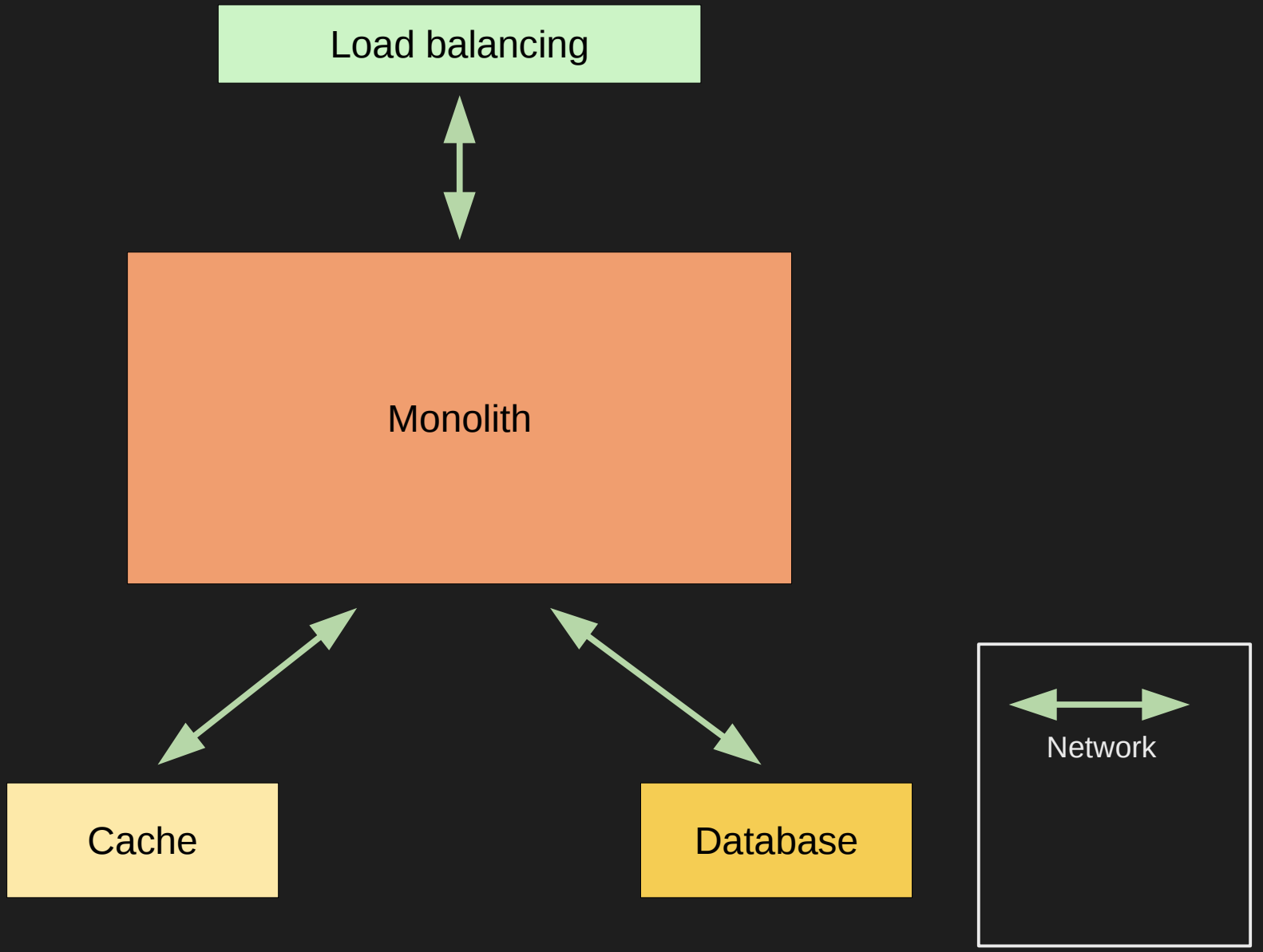
❸ 架構是演進的，預想但不過早調優

無視未來只求現有，是耍自閉

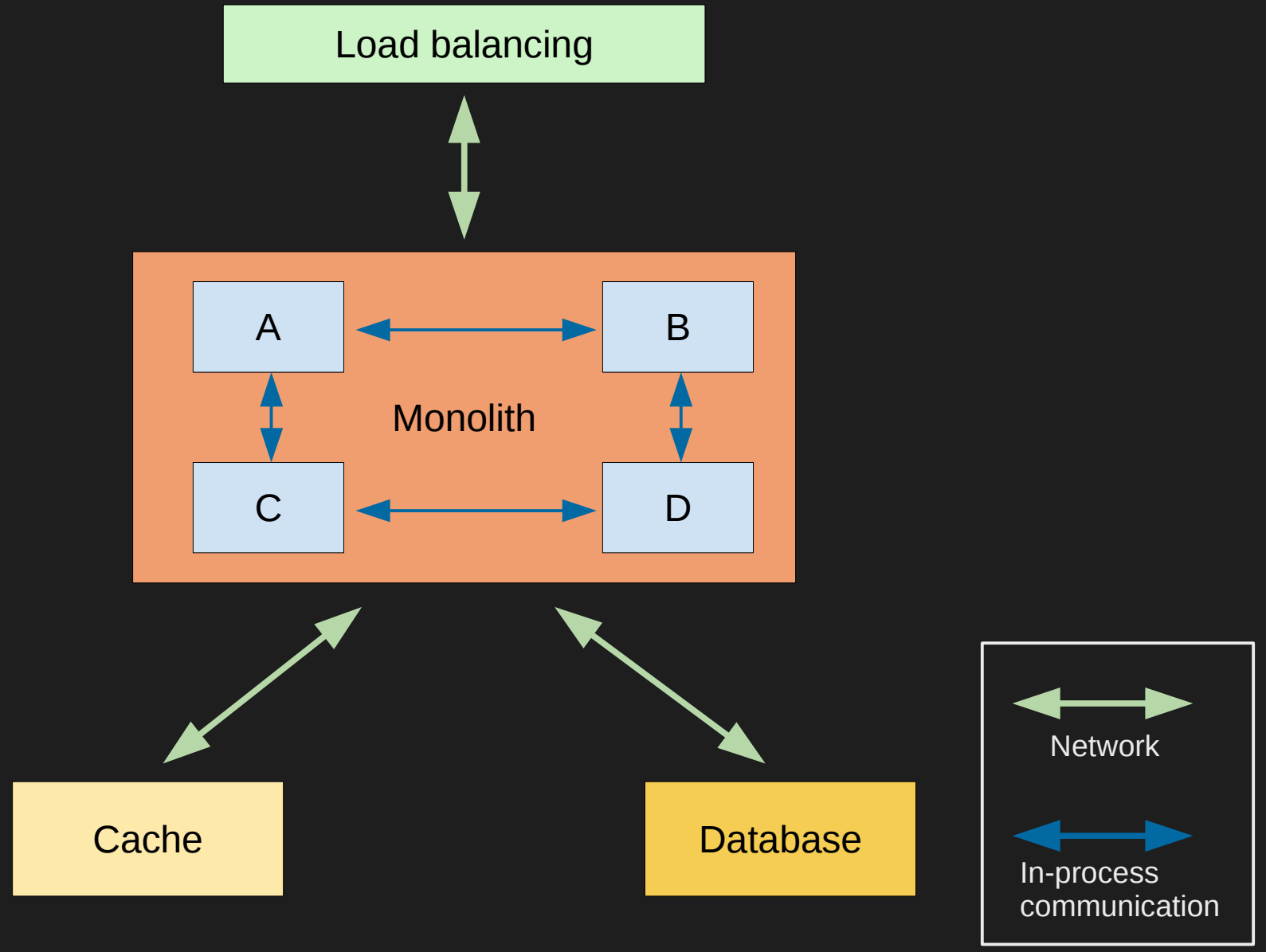
兵馬未動，糧草先行，預想下一步，下下一步，甚至下下下一步 ...


Monolith
SOA
Microservices
Picoservices ?
Nanoservices

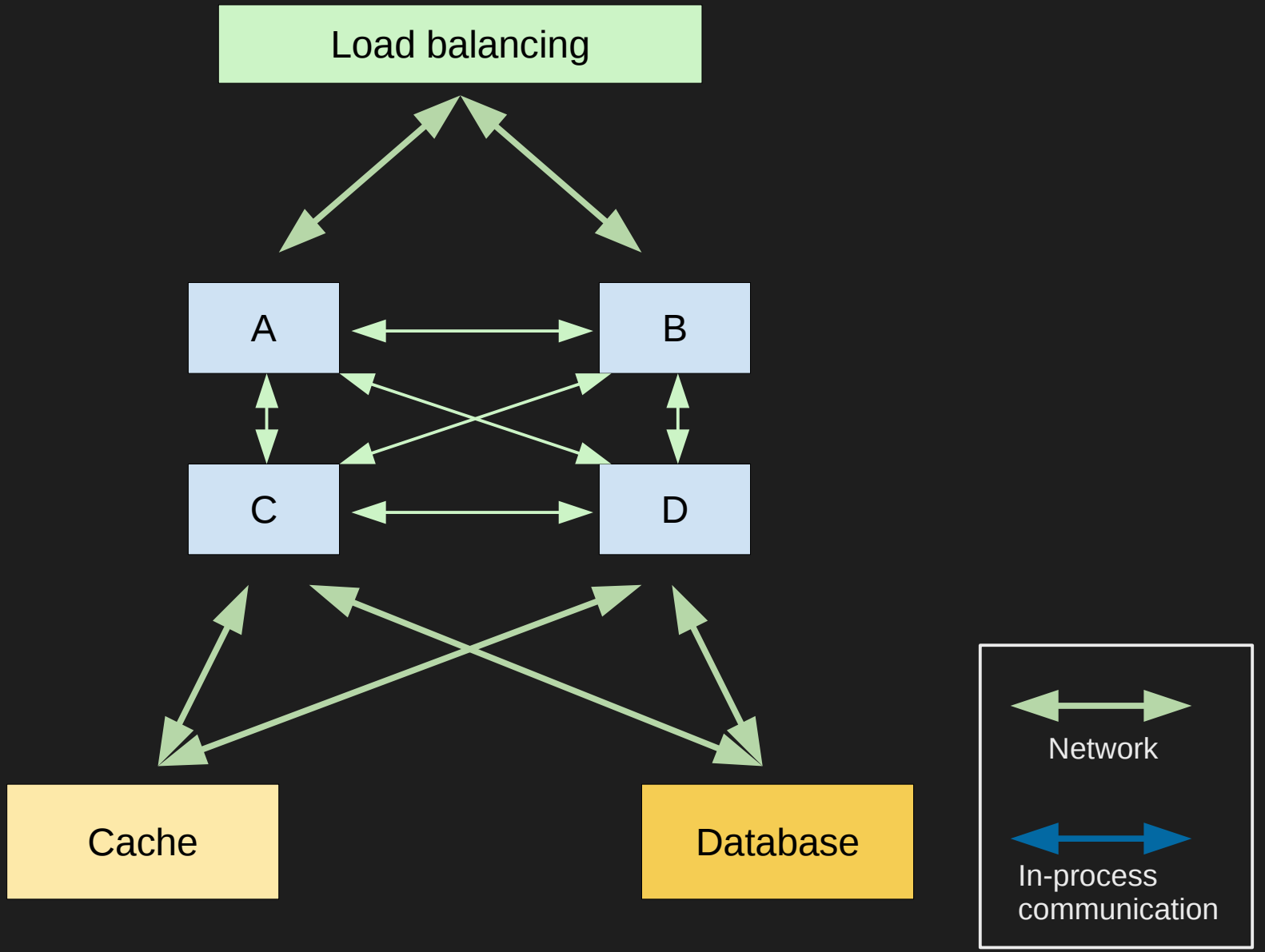
Monolith → Microservices



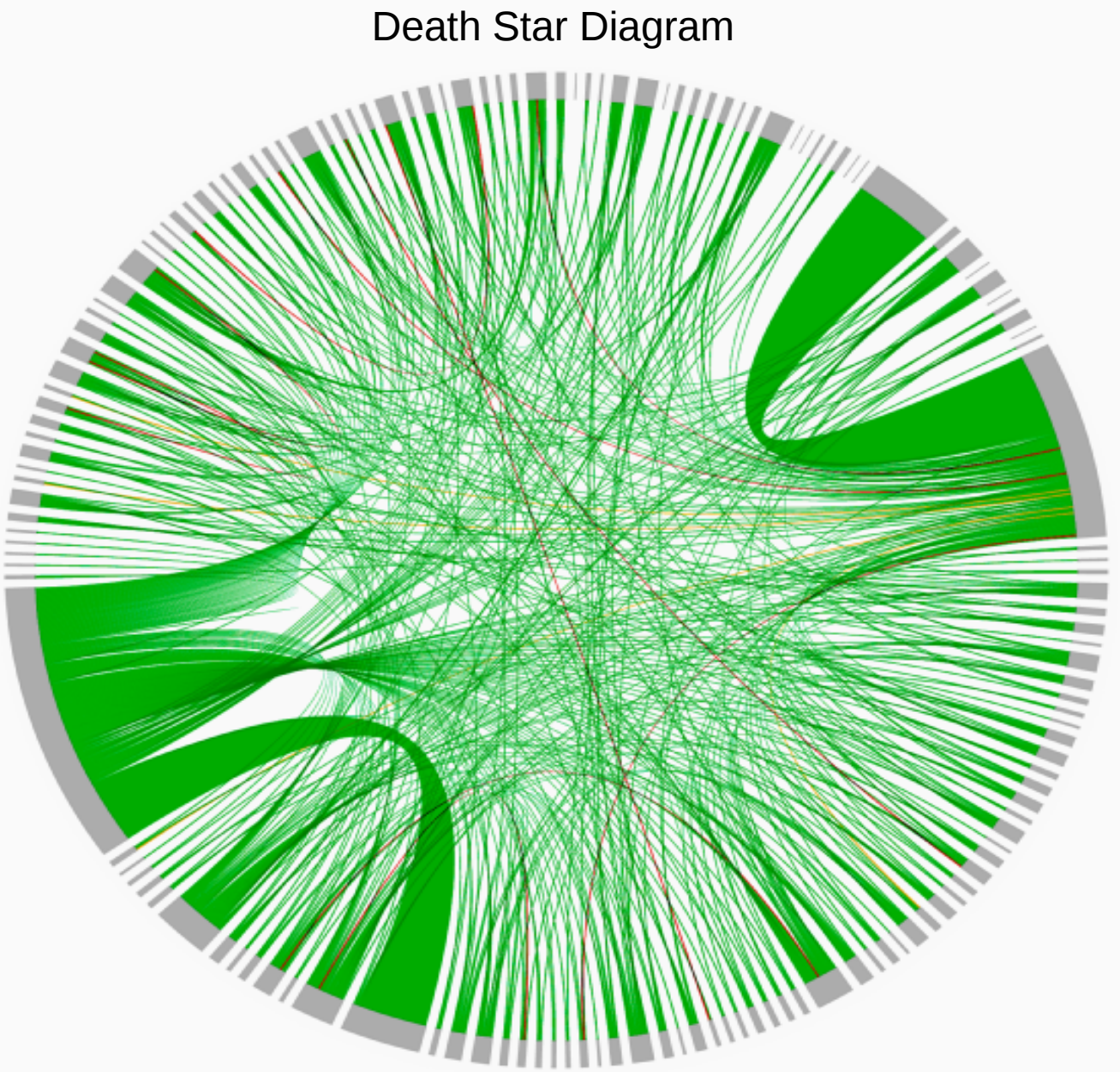
Monolith → Microservices



Monolith → Microservices

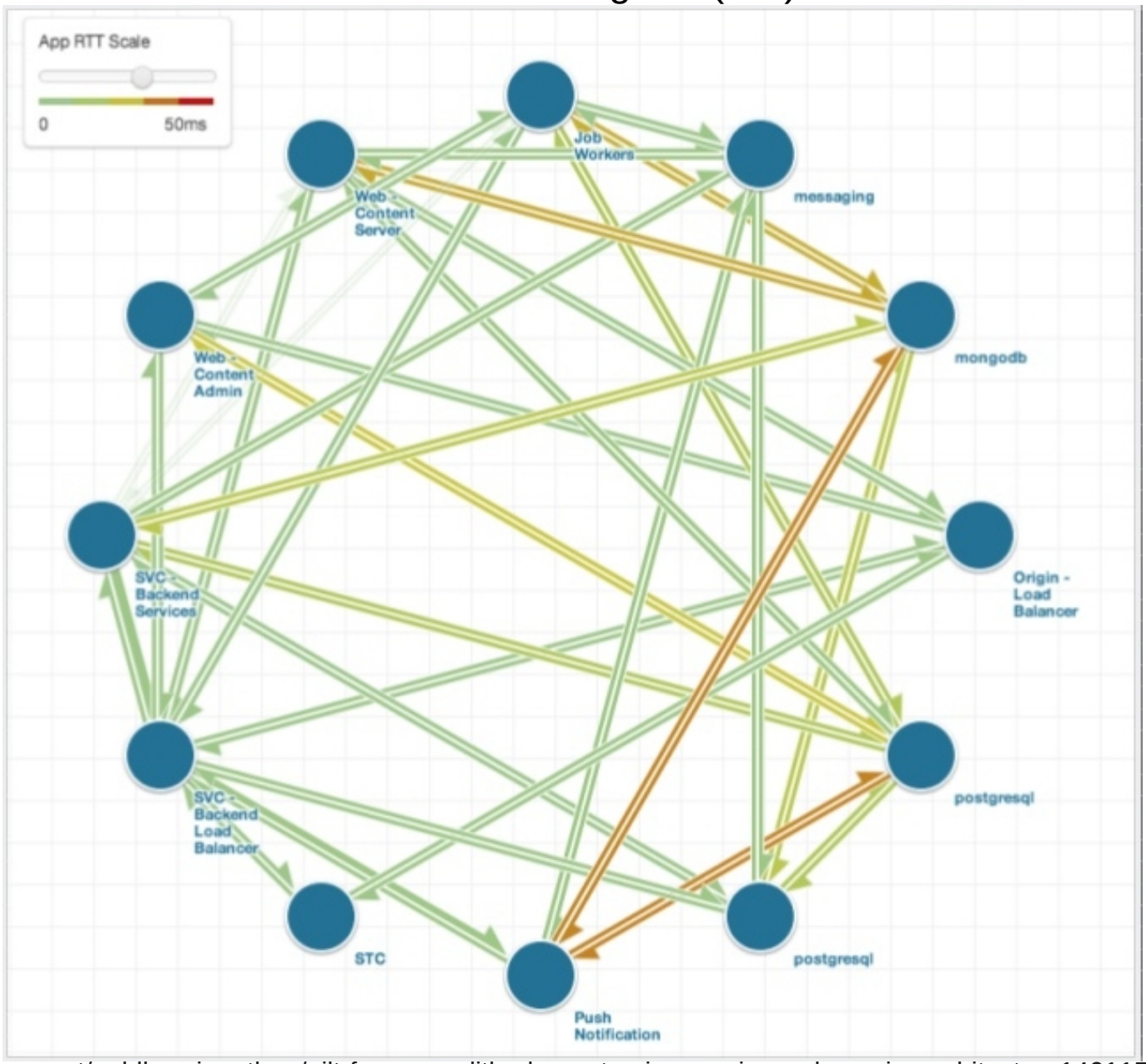


Monolith → Microservices



Monolith → Microservices

Death Star Diagram (Gilt)



Ref: http://www.slideshare.net/goldbergjonathan/gilt-frommonolithrubyapptomicroservicescalaservicearchitecture140115150436phpapp01 (s45)

Monolith → Microservices

Death Star Diagram (Twitter)

Ref: https://twitter.com/adrianco/status/441883572618948608

Monolith → Microservices

Death Star Diagram (Netflix)

Ref: http://www.slideshare.net/BruceWong3/the-case-for-chaos (s12)

In-process communication

Fast

Network (HTTP)

Slow and unreliable

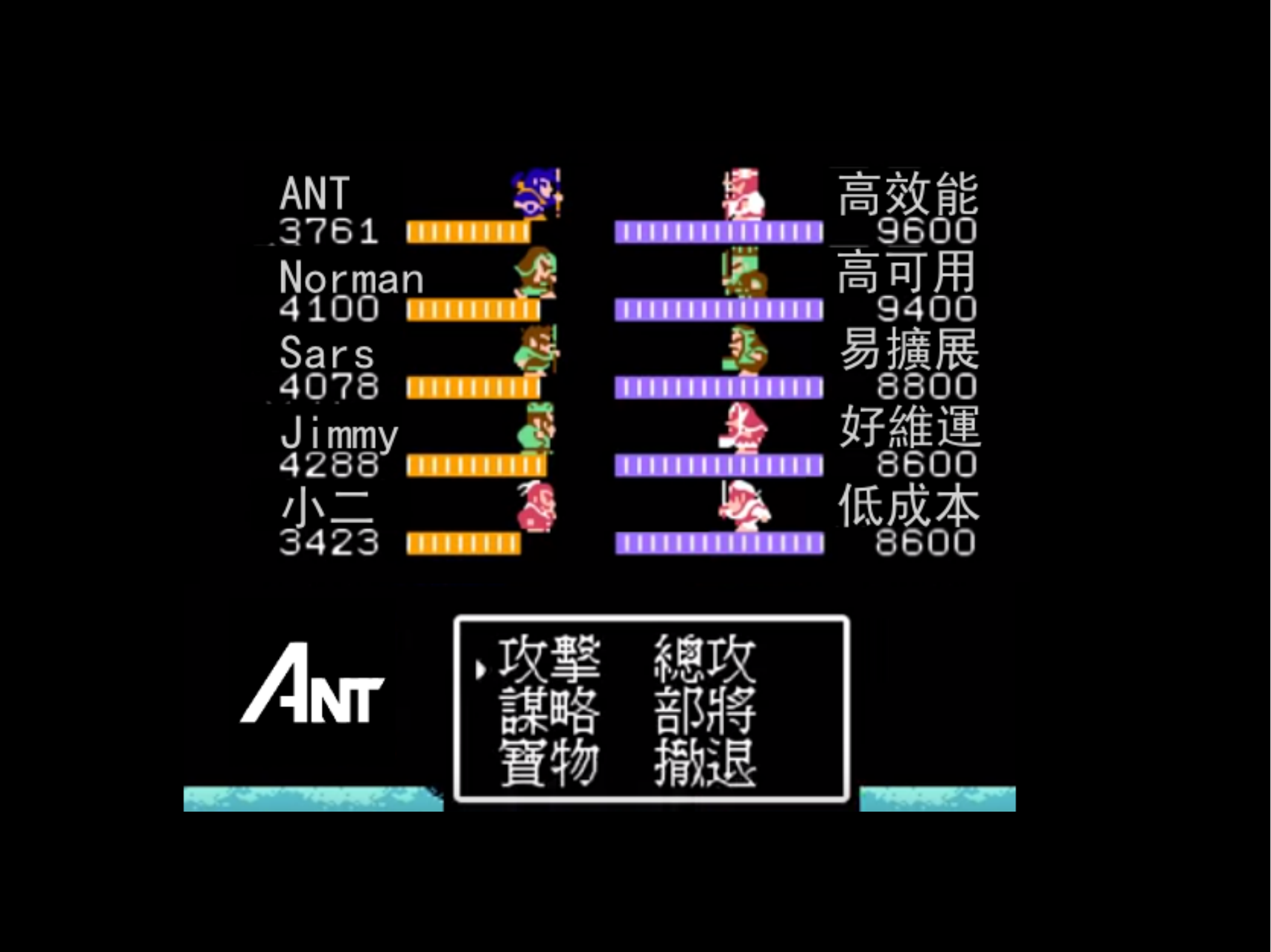
More worse,
overlay networks is slow ...

Name	TCP BW (MB/s)	TCP Lat (µs)	BW %	Lat %
Native	116	91.8	100.00	100.00
Weave	6.91	372	5.96	405.23
Flannel UDP	23	164	19.83	178.65
Flannel VXLan	112	129	96.55	140.52

Ref: http://www.generictestdomain.net/docker/weave/networking/stupidity/2015/04/05/weave-is-kinda-slow/

Azure VM type	network interface	native qperf bandwidth	native qperf latency	weave qperf bandwidth	weave qperf latency	weave bandwith compared to native
A0	5 Mbps	590 KB/sec	289 us	557 KB/sec	419 us	-16%
A1	100 Mbps	12.5 MB/sec	483 us	9.11 MB/sec	483 us	-27%
A2	200 Mbps	24.8 MB/sec	426 us	10.3 MB/sec	557 us	-58%
A3	400 Mbps	48 MB/sec	465 us	12 MB/sec	483 us	-75%
A8	not documented	310 MB/sec	94.2 us	51 MB/sec	188 us	-83%
D1	not documented	56.3 MB/sec	276 us	17 MB/sec	456 us	-69%
D2	not documented	117 MB/sec	178 us	24.6 MB/sec	446 us	- 79%

Ref: https://arjanschaaf.github.io/is-the-network-the-limit/ (2015-06-29)


ANT
3761
Norman
4100
Sars
4078
Jimmy
4288
小二
3423
高效能
9600
高可用
9400
易擴展
8800
好維運
8600
低成本
8600
ANT
攻擊 總攻
謀略 部將
寶物 撤退

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

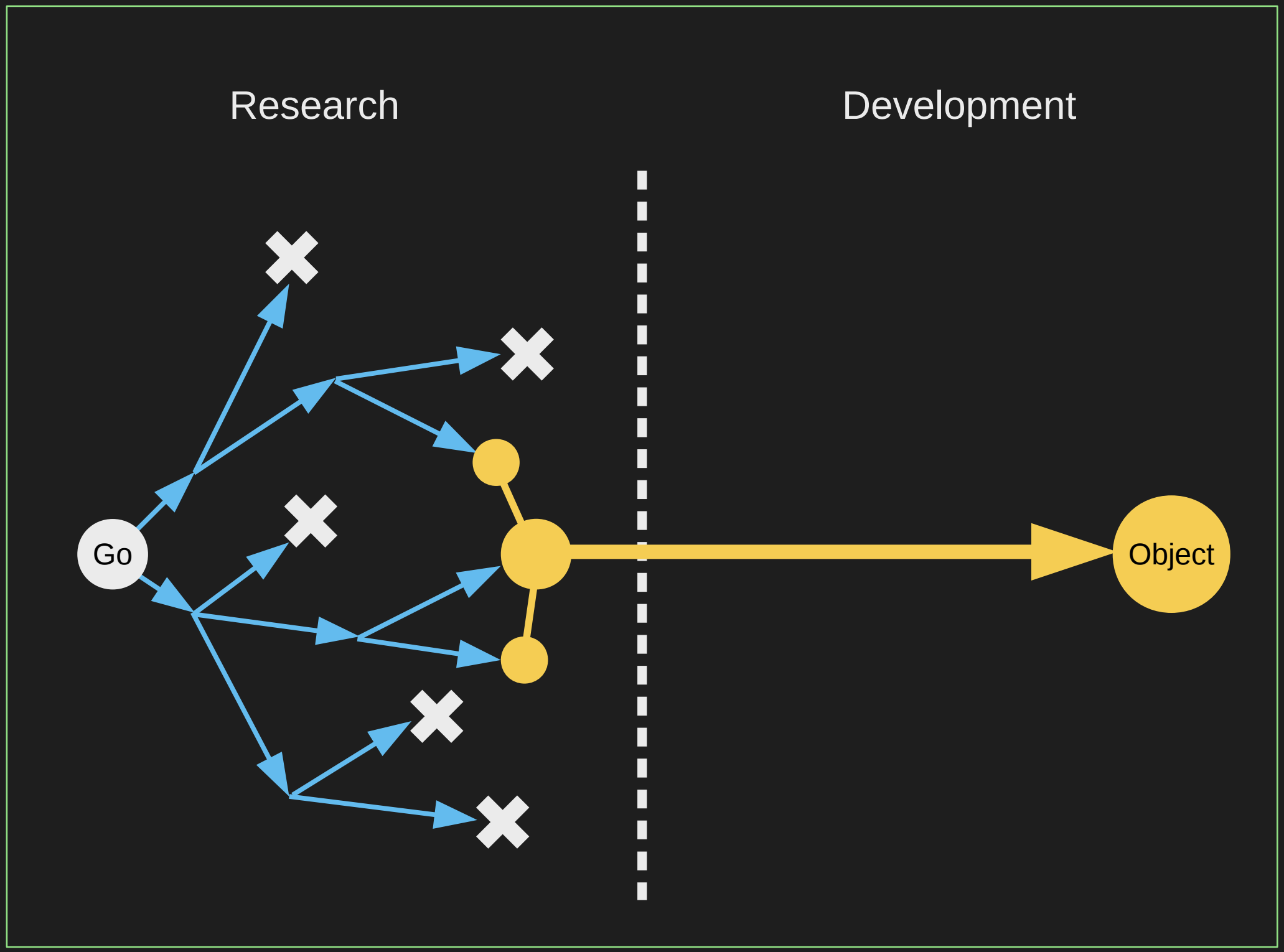
- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

Enterprise Architecture Case in PHP

https://www.muzik-online.com/

https://muzikair.com/

We ❤ PHP

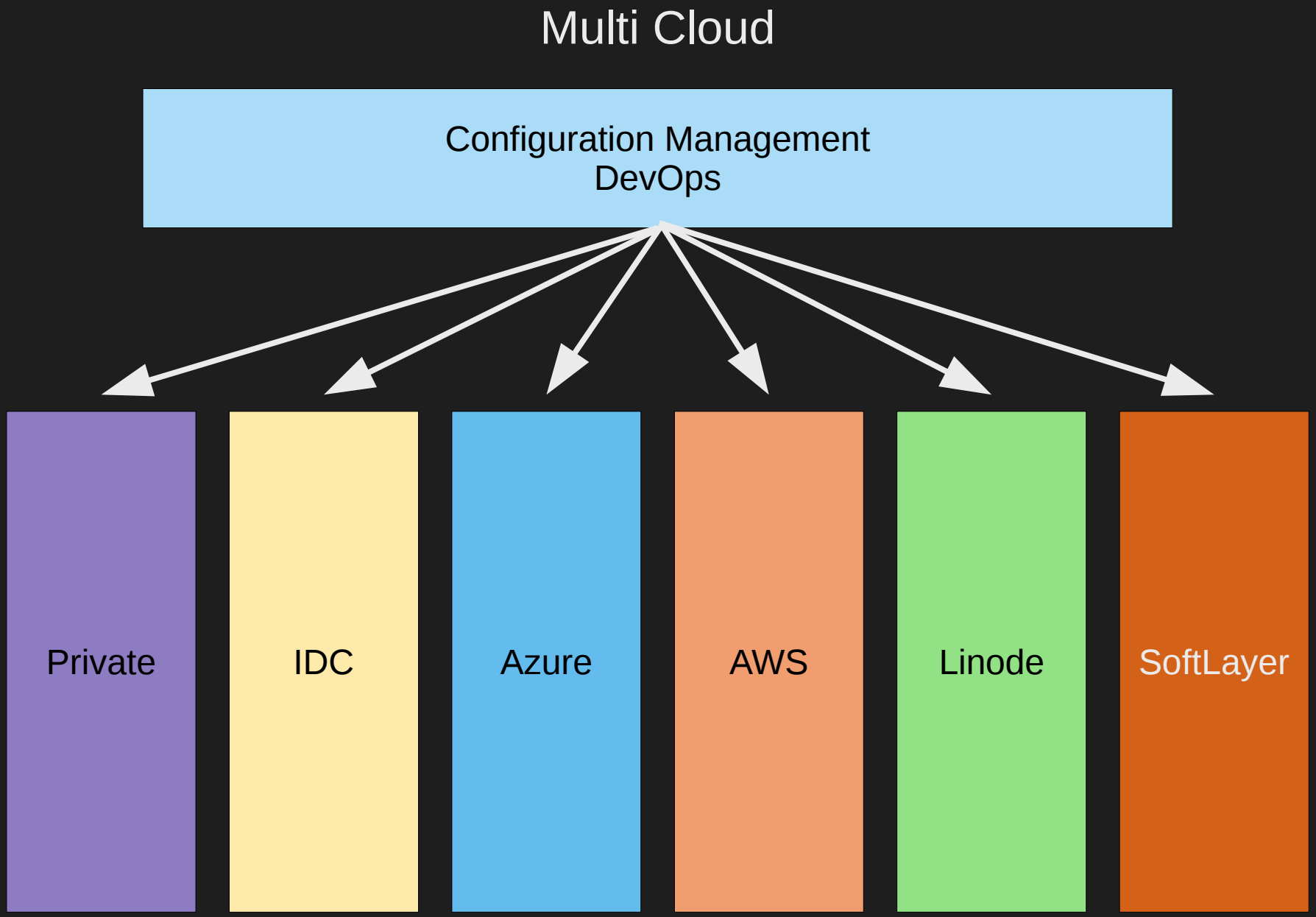

Research
Development
Go
Object

https://www.muzik-online.com/

https://muzikair.com/

對技術研究高度投入，專注節省工程師開發時間

投入時間分析研究 MySQL / PostgreSQL / PHP / Cassandra / Redis ... 等源碼


Multi Cloud
Configuration Management
DevOps
Private
IDC
Azure
AWS
Linode
SoftLayer

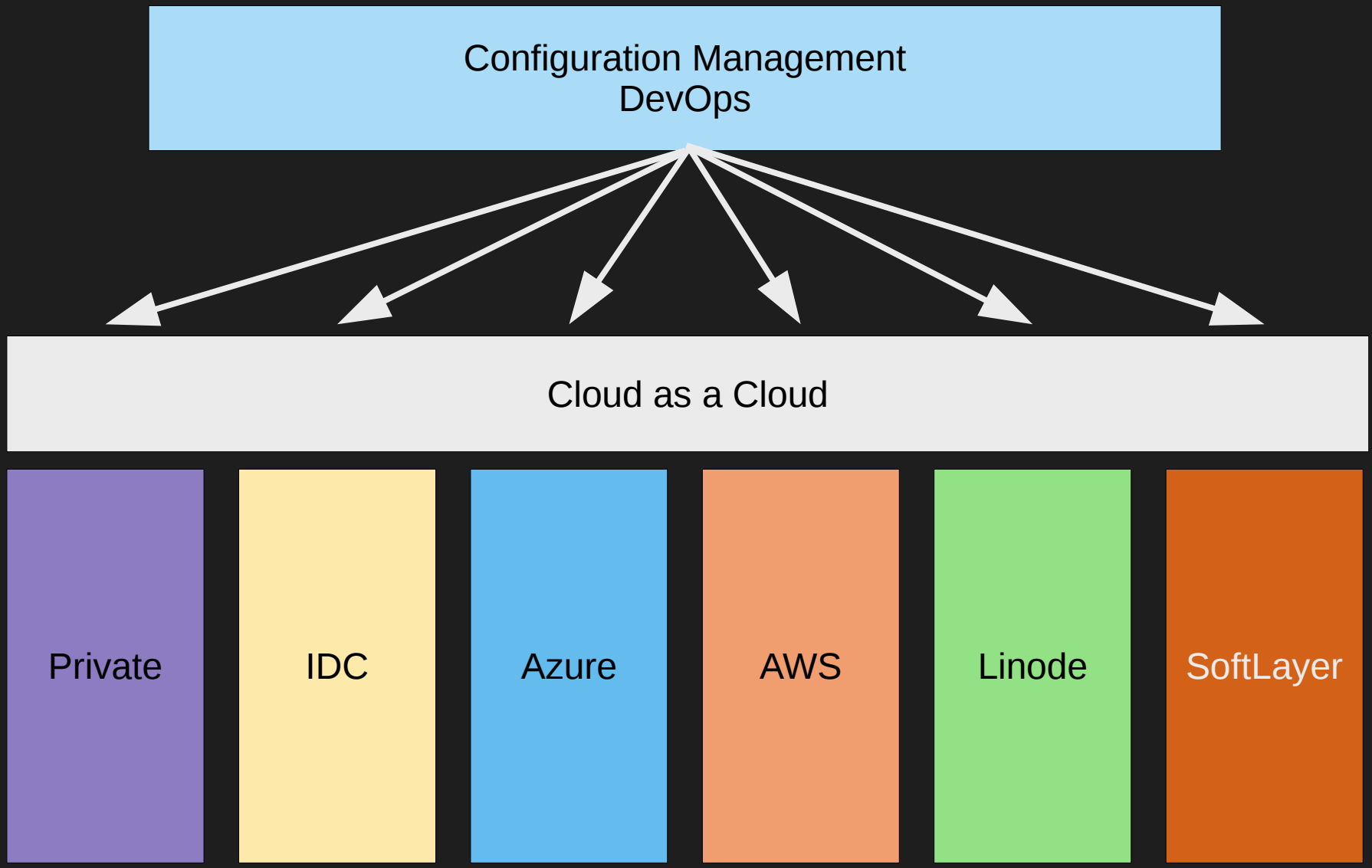
https://www.muzik-online.com/

https://muzikair.com/

追求 Cloud as a Cloud

Leverage w/o vendor lock-in
Build cloud on top of cloud

Multi Cloud


Configuration Management
DevOps
Cloud as a Cloud
Private
IDC
Azure
AWS
Linode
SoftLayer

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

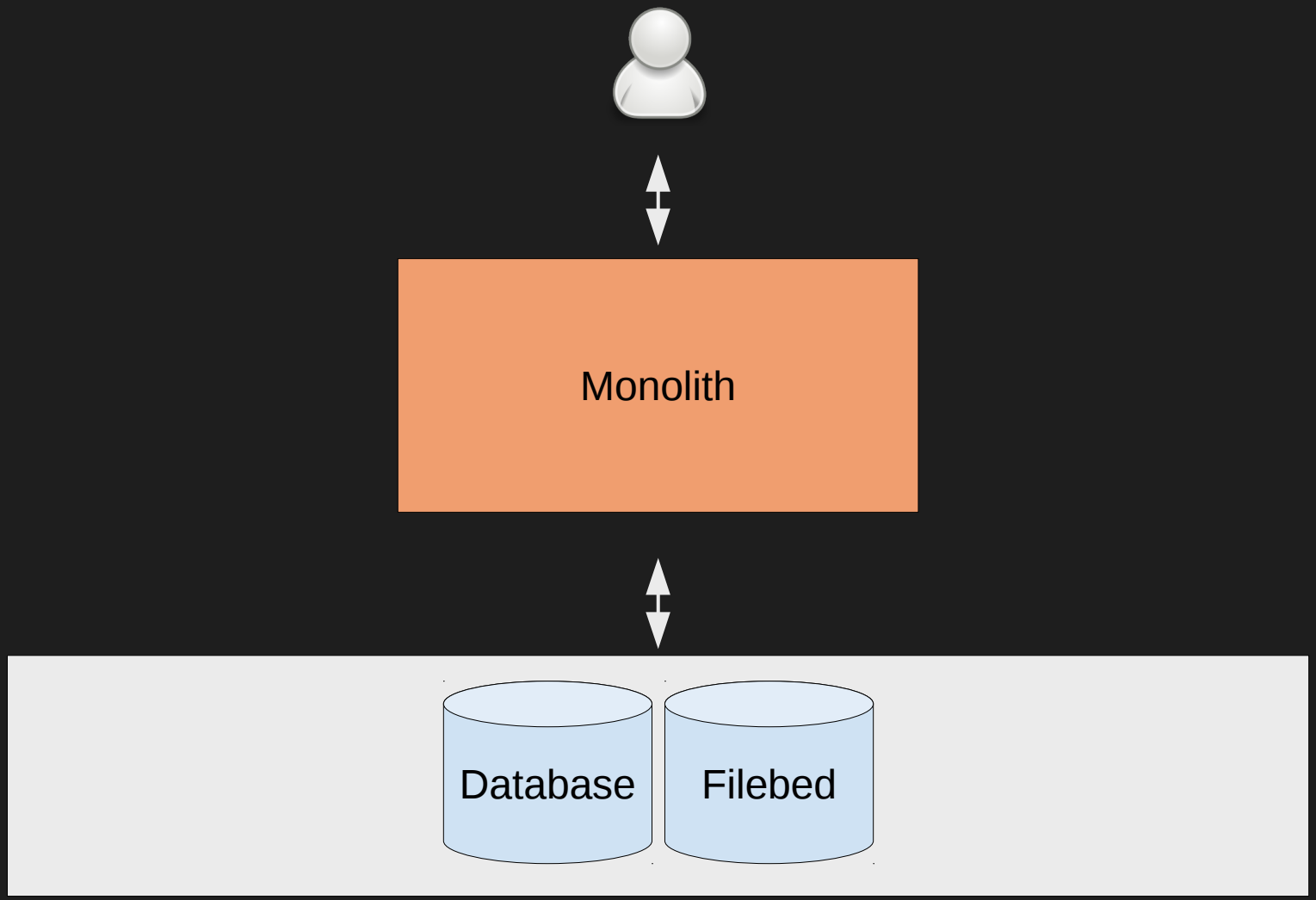
議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

乾坤之初，混沌玄黃

上古卷軸，大型冒險

History > MUZIK Zero



Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

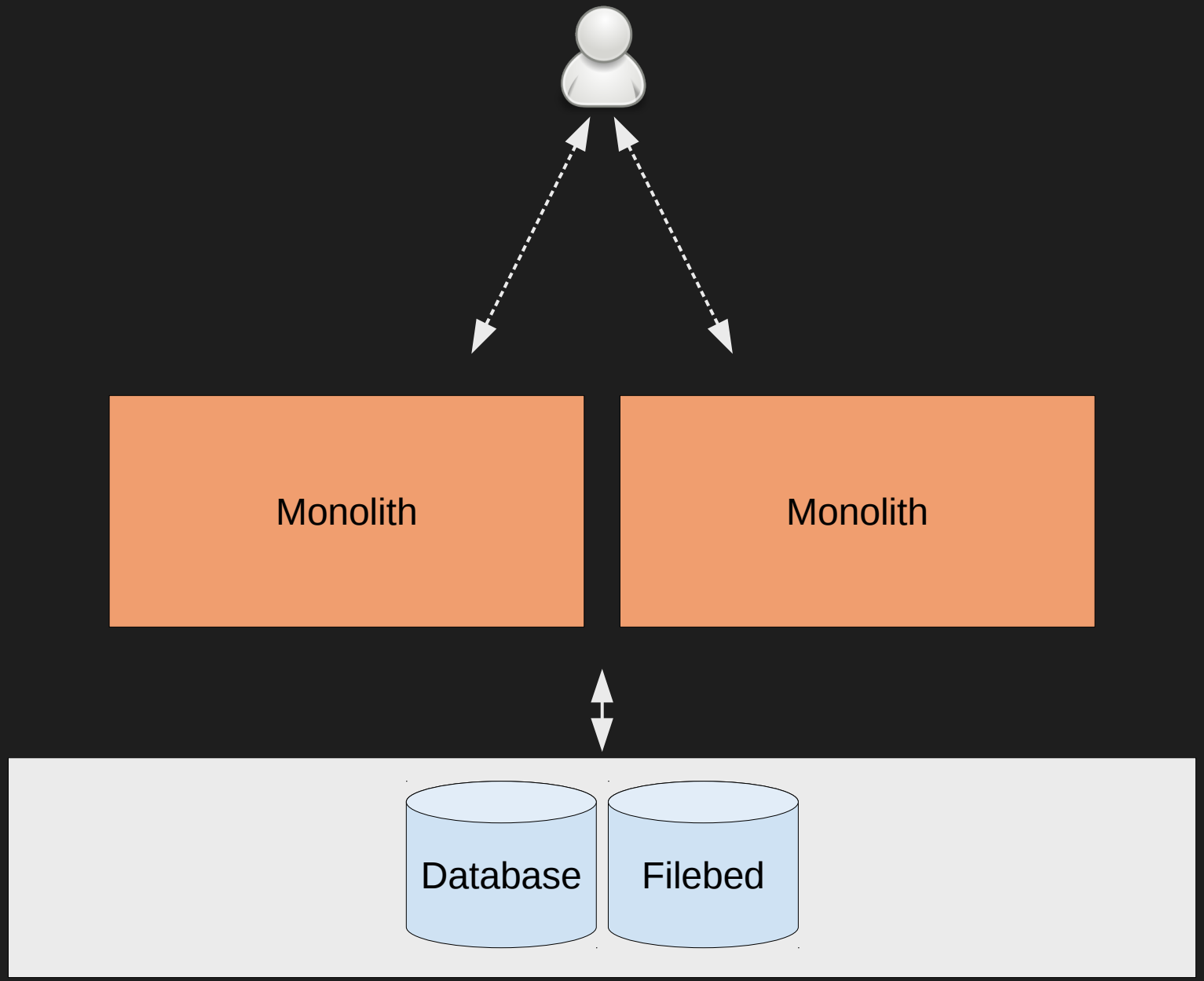
巴赫 (Bach)

古典樂之父

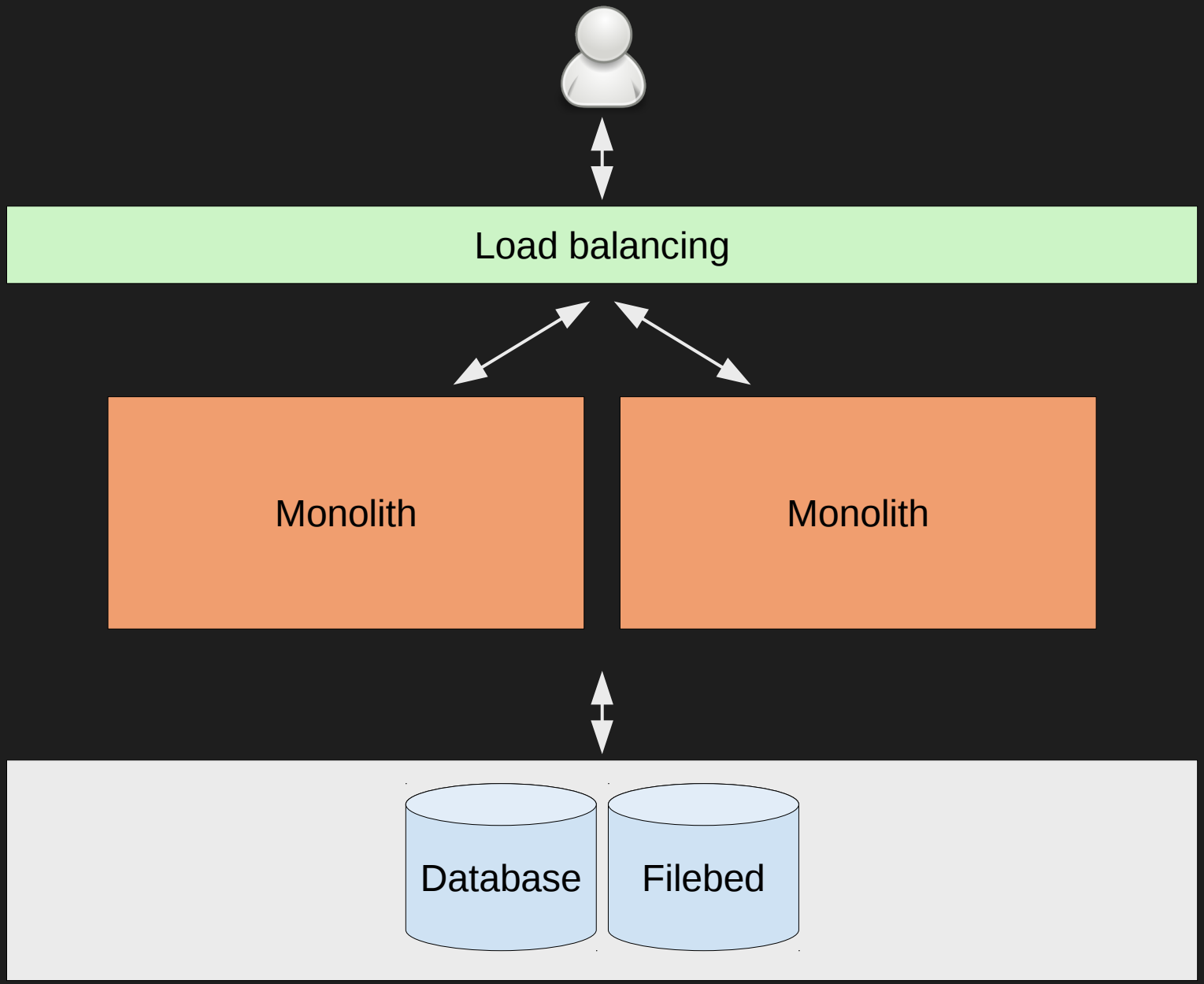
架構之始，動而至靜

披荊斬棘，開國立基

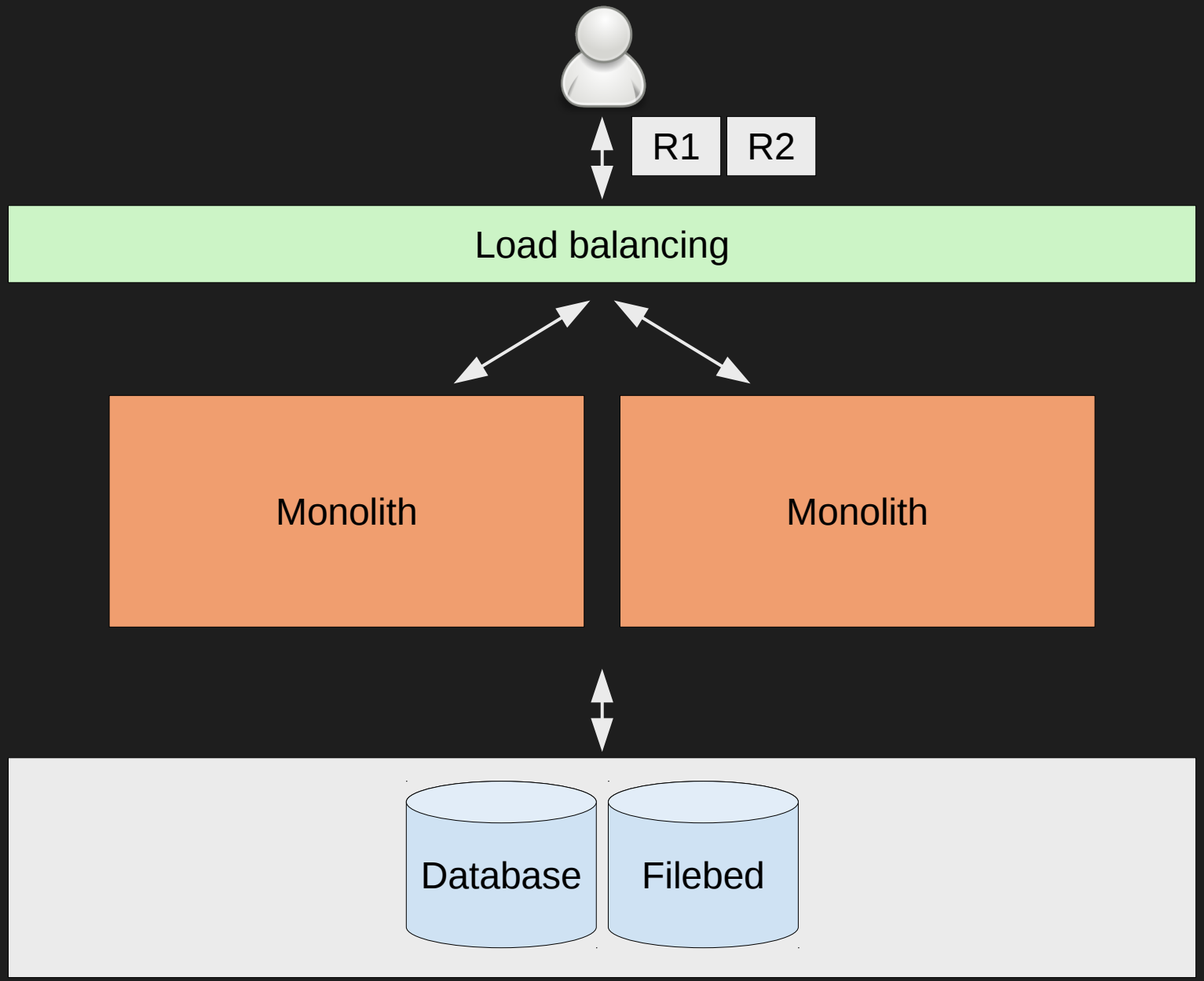
History > MUZIK 1.0 (Bach)



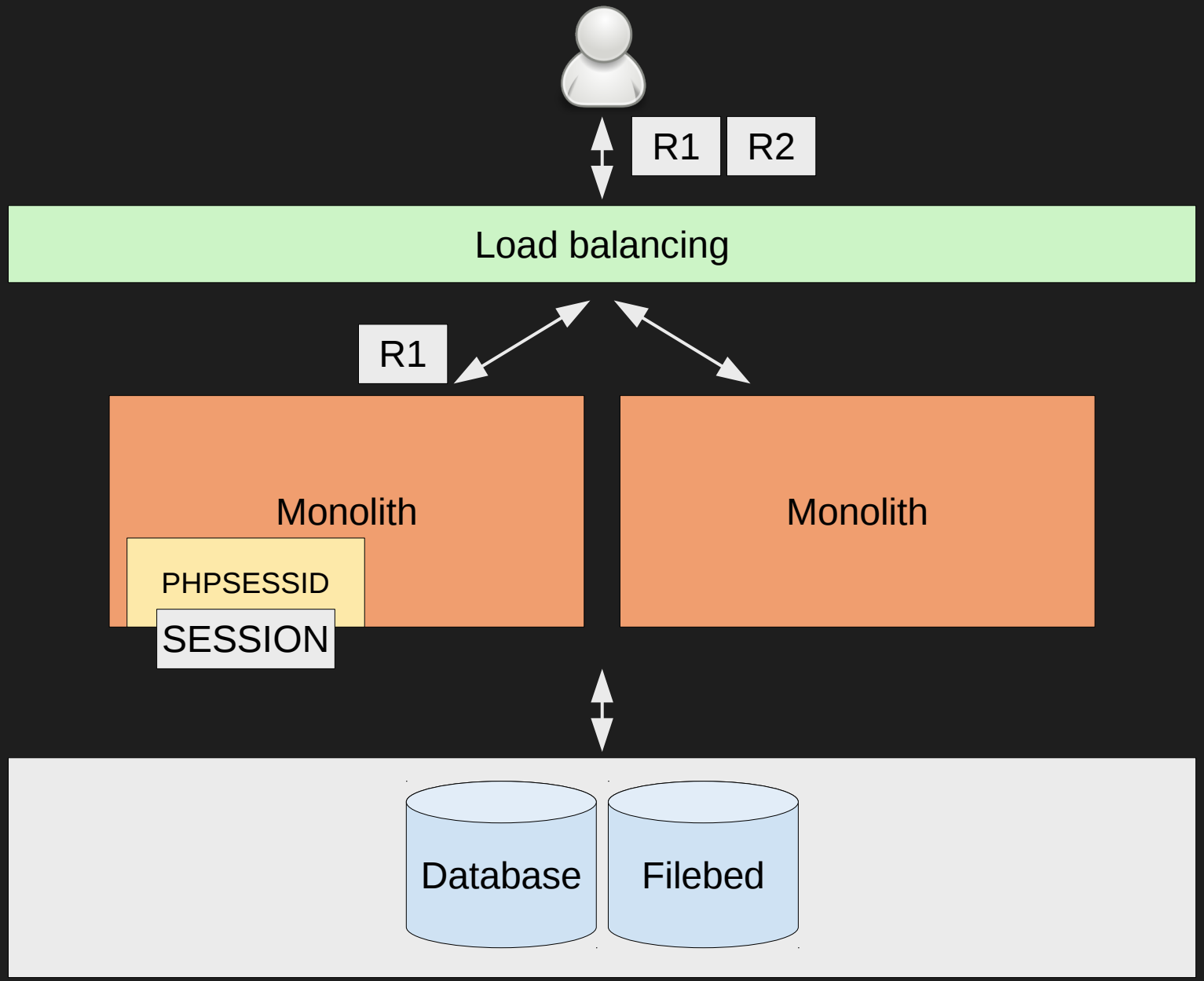
History > MUZIK 1.0 (Bach)



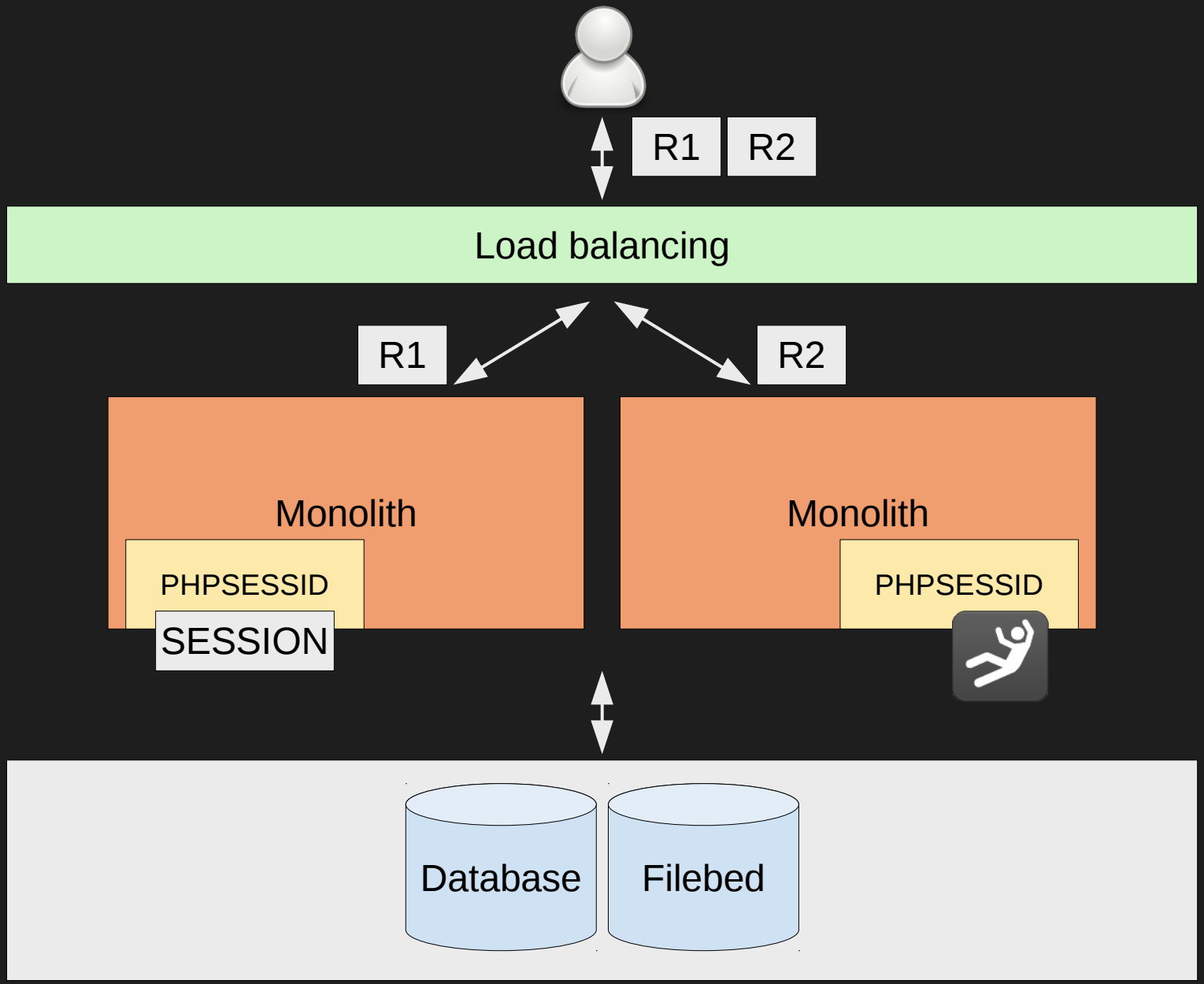
History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)



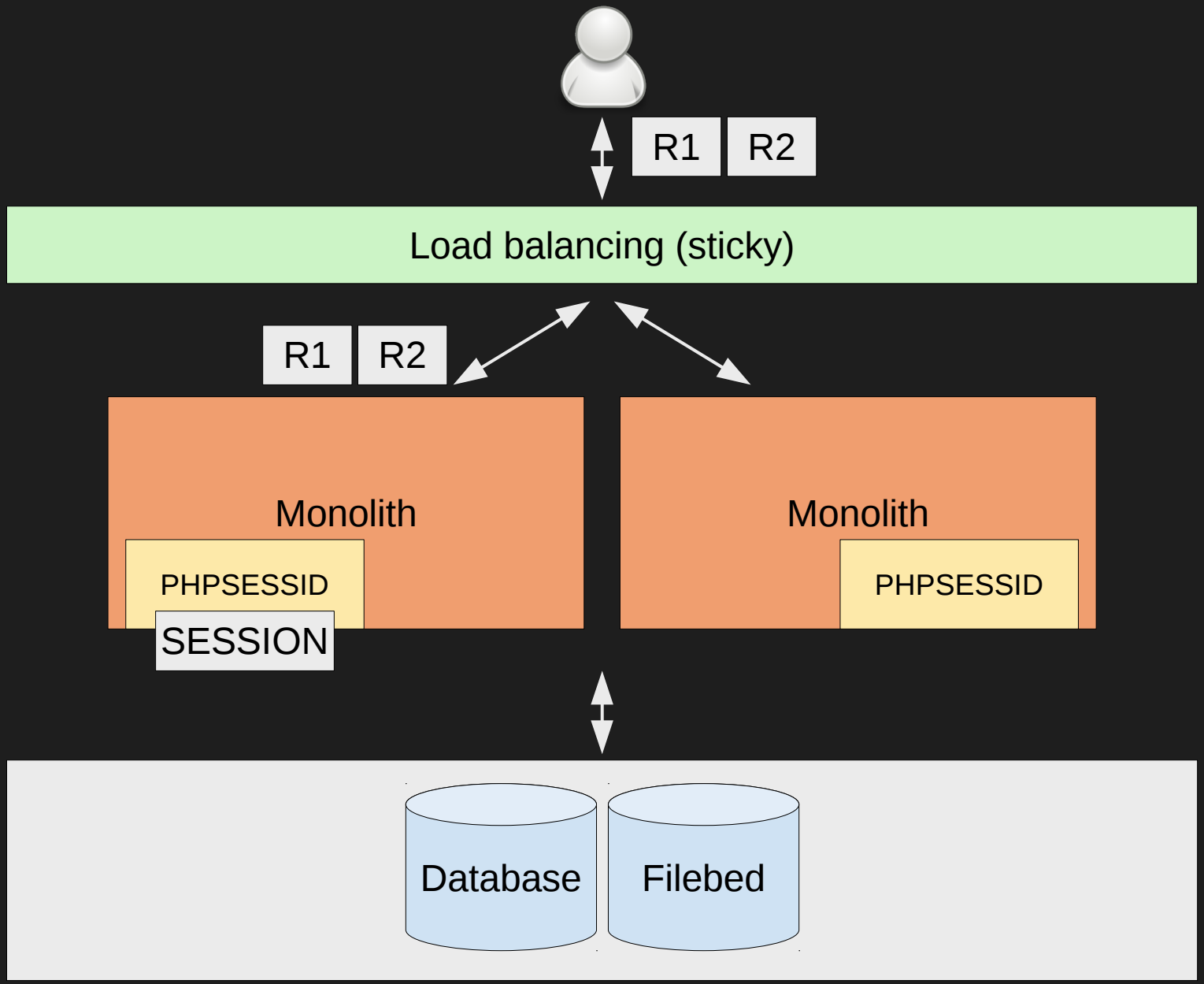
History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

"Sticky" balancing / cookies

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Load balancing do more works (lower performance)
  - ✔ Unbalanced loads
  - ✔ Supporting
    - ✔ Some old hardware / software do not support 'sticky'
    - ✔ Amazon ELB support sticky sessions on April 2010
  - ✔ Hard to control and scale-out (Non-SDN, No APIs)

History > MUZIK 1.0 (Bach)

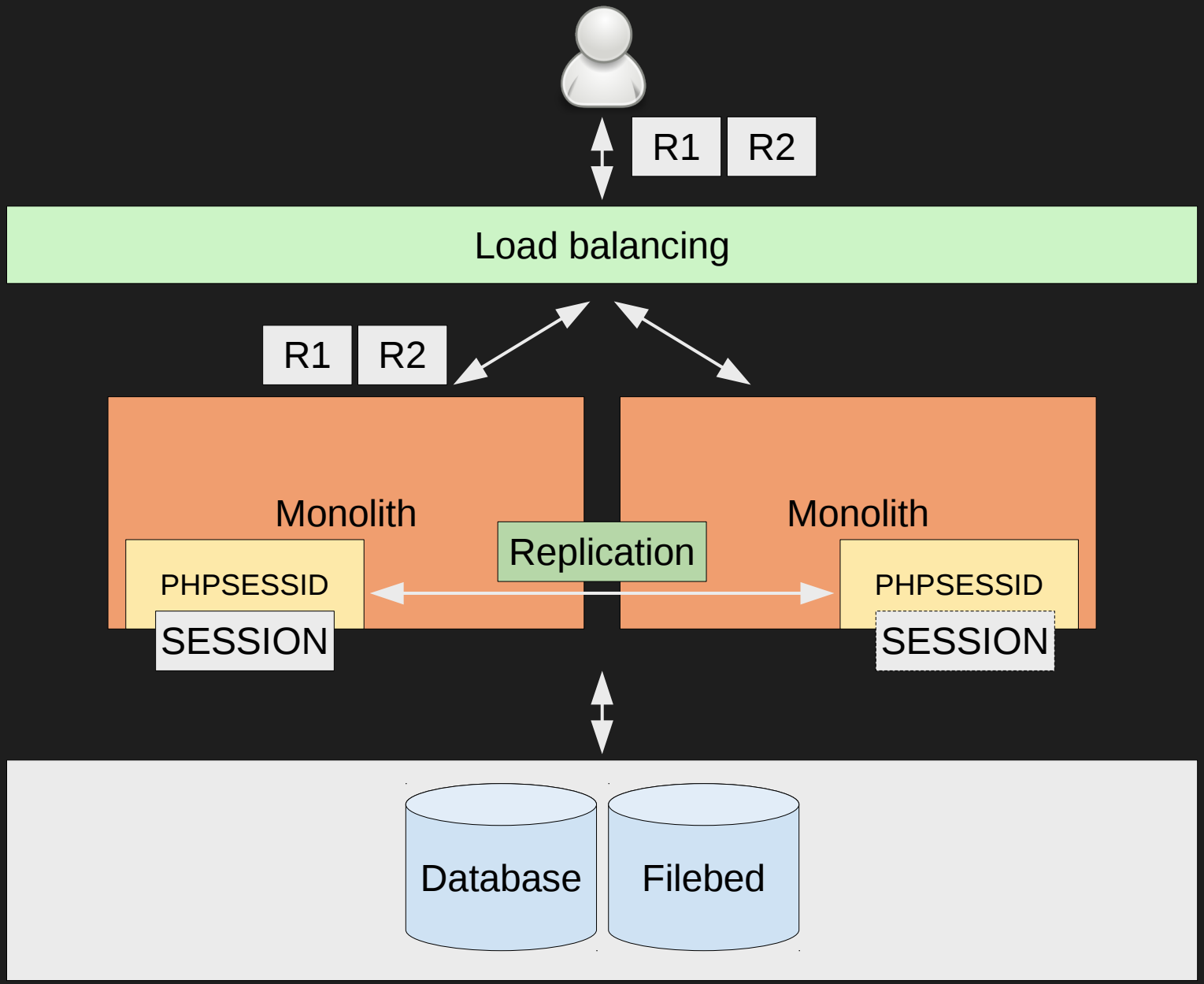
- ✔ 缺點
  - ✔ 負載平衡器多了一些工作 (效能耗損)
  - ✔ 負載不平衡 (重度用戶會偏重使用某些機器)
  - ✔ 支援性
    - ✔ 有些舊的硬體 / 軟體不支援 'sticky'
    - ✔ 2010 年 4 月 Amazon ELB 才支援 sticky sessions
  - ✔ 難以控制或橫向擴展 (硬體不支援 SDN ，沒有 API)

History > MUZIK 1.0 (Bach)

Application server session replication

(Tomcat / JBoss / WAS)

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ More application servers, more performance lost
  - ✔ Broadcast storm
  - ✔ Session serialization / unserialization (lower performance)
  - ✔ Supporting
    - ✔ Some application server do not supporting

History > MUZIK 1.0 (Bach)

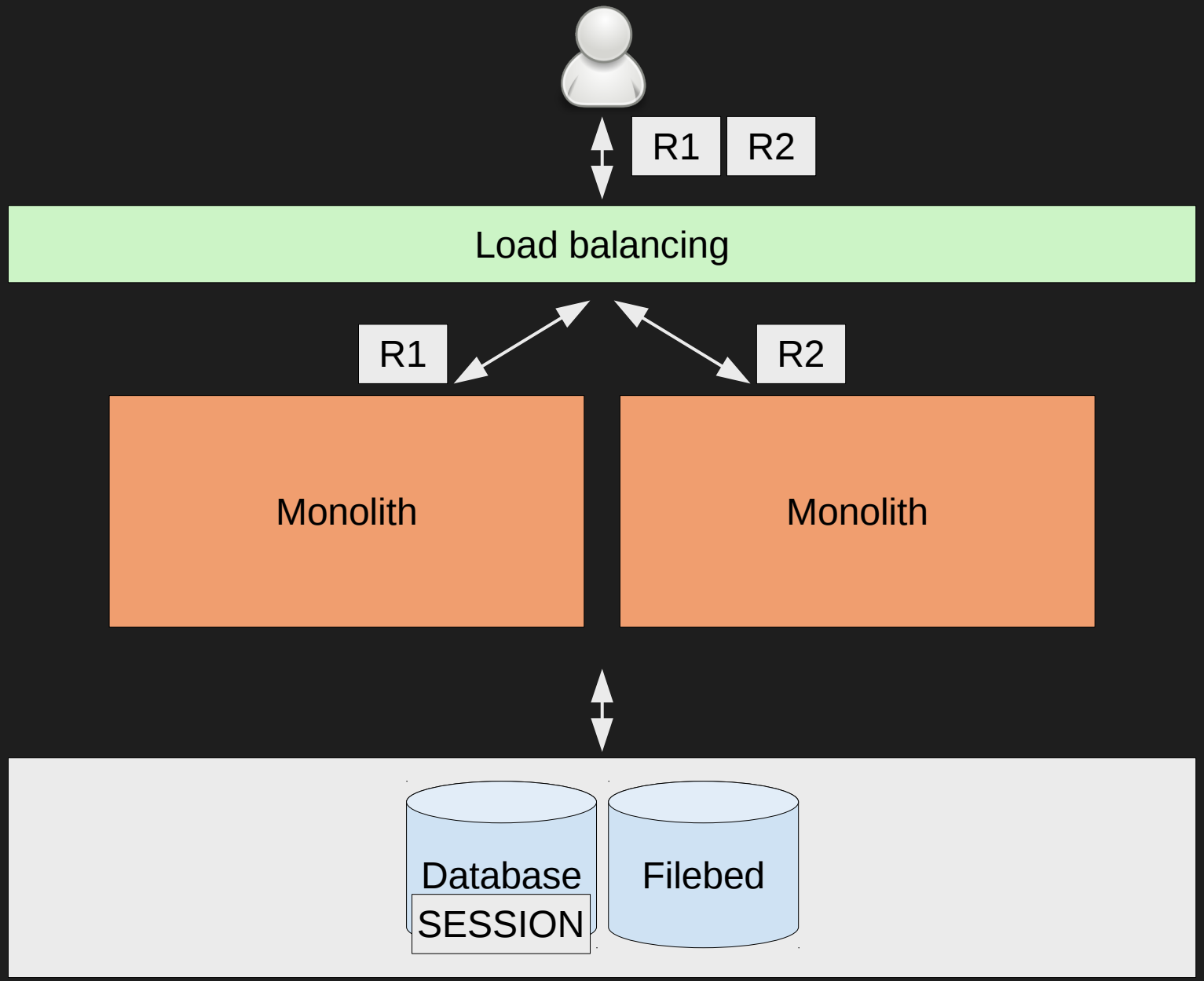
- ✔ 缺點
  - ✔ 應用伺服器愈多，同步效能愈差
  - ✔ 廣播風暴
  - ✔ Session 需要序列化 / 反序列化，影響效能
  - ✔ 支援性
    - ✔ 不是所有應用伺服器都支援 Session 複製

History > MUZIK 1.0 (Bach)

Save sessions in database

(shared-nothing architecture)

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Need to modify application code (easy)
  - ✔ Database do more works (lower performance)
  - ✔ Database is more harder scale-out than application server

History > MUZIK 1.0 (Bach)

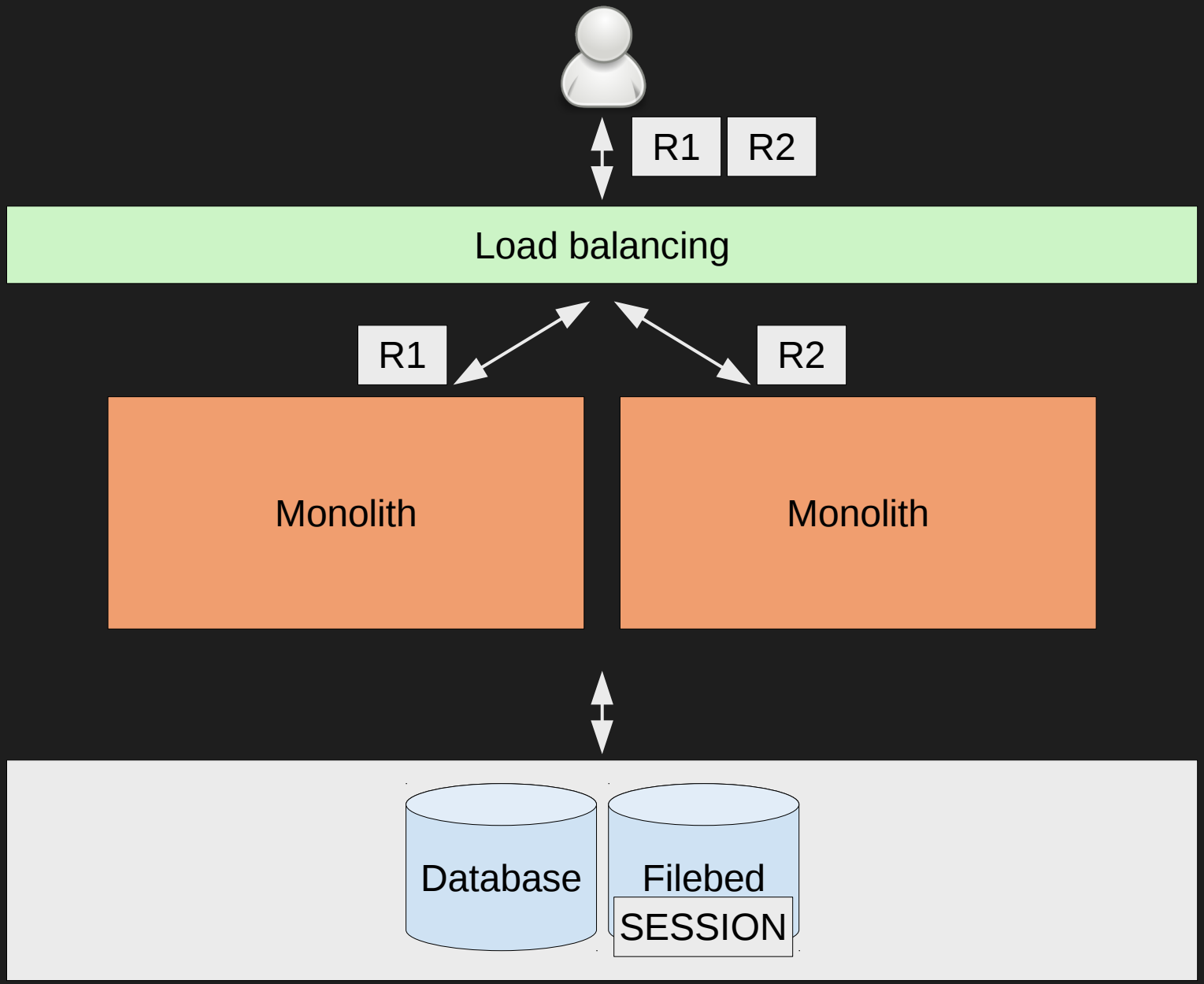
- ✔ 缺點
  - ✔ 需要修改應用程式 (容易)
  - ✔ 資料庫多了一些工作 (效能耗損)
  - ✔ 資料庫相較應用伺服器更難以橫向擴展

History > MUZIK 1.0 (Bach)

Save sessions in shared disks

(shared-nothing architecture)

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Need to modify application code (easy)
  - ✔ Shared disk do more works (lower performance)
  - ✔ Shared disk is hard to scale-out
  - ✔ Session serialization / unserialization (lower performance)

History > MUZIK 1.0 (Bach)

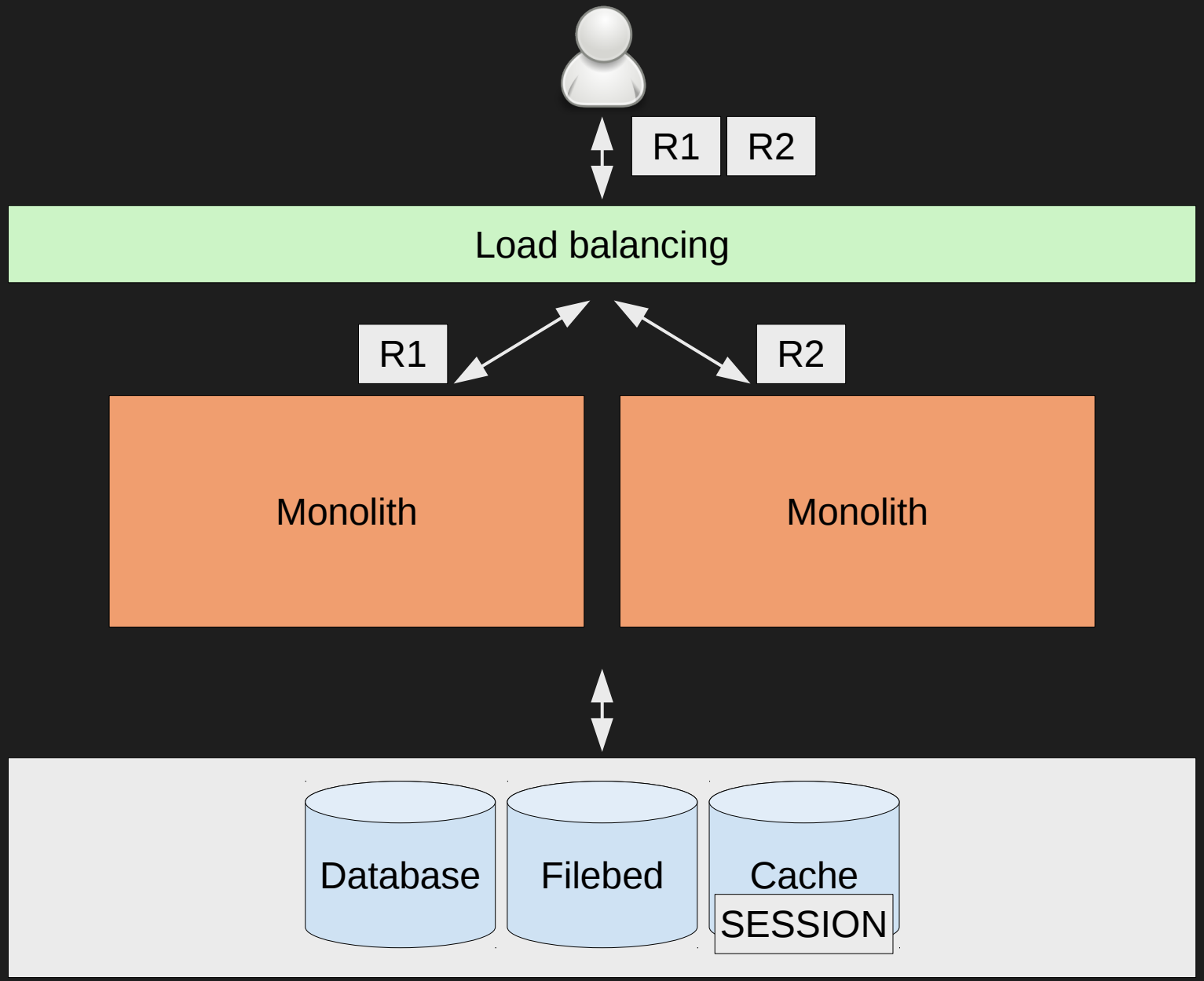
- ✔ 缺點
  - ✔ 需要修改應用程式 (容易)
  - ✔ 共享儲存庫多了一些工作 (效能耗損)
  - ✔ 共享儲存庫較難橫向擴展
  - ✔ Session 需要序列化 / 反序列化，影響效能

History > MUZIK 1.0 (Bach)

Save sessions in shared cache

(Memcached / Redis / Aerospike)
(shared-nothing architecture)

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Need to modify application code (easy)
  - ✔ Memcached need serialization / unserialization (lower performance)
    - ✔ Redis does not
  - ✔ Memcached do not support persistent store (thundering herd)
    - ✔ Redis / Aerospike does
  - ✔ Cache server do more works (lower performance)
  - ✔ Cache server is hard to scale-out
    - ✔ Aerospike is more easy

History > MUZIK 1.0 (Bach)

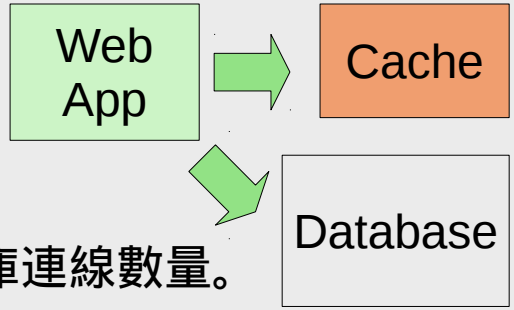
- ✔ 缺點
  - ✔ 需要修改應用程式 (容易)
  - ✔ Memcahced 需要序列化 / 反序列化 (稍影響效能)
    - ✔ Redis 不需要
  - ✔ Memcached 不支援永久儲存 (雪崩效應)
    - ✔ Redis / Aerospike 支援
  - ✔ 快取伺服器多了一些工作 (效能耗損)
  - ✔ 快取伺服器較難以擴展
    - ✔ Aerospike 相對簡單

History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Cache servers need to be monitoring (become more complex)
  - ✔ Cache servers need large memory (more expensive)
  - ✔ Cache servers itself may bring thundering herd problem (even Redis)
    - ✔ Cache server reduce number of database
    - ✔ If without cache server, database will need more power to handle
    - ✔ So if cache server crash, all traffic will transfer to database, and resulting database crash too.
    - ✔ Finally, database is depend on cache server.

History > MUZIK 1.0 (Bach)

- ✔ 缺點
  - ✔ 快取伺服器需要集群與監控 (架構變複雜)
  - ✔ 快取伺服器需要大量記憶體 (費用較昂貴)
  - ✔ 快取伺服器本身很可能是雪崩效應的元兇 (即使是 Redis)
    - ✔ 快取存在的意義，通常為了節省資料庫效能 / 減少資料庫連線數量。
    - ✔ 若沒有快取，資料庫很可能需要多 10 倍或更多的數量。
    - ✔ 一旦快取伺服器崩潰，流量若全轉向至資料庫，也會造成資料庫崩潰。
    - ✔ 資料庫集群的崩潰，除了自身外，也多依賴了快取伺服器。

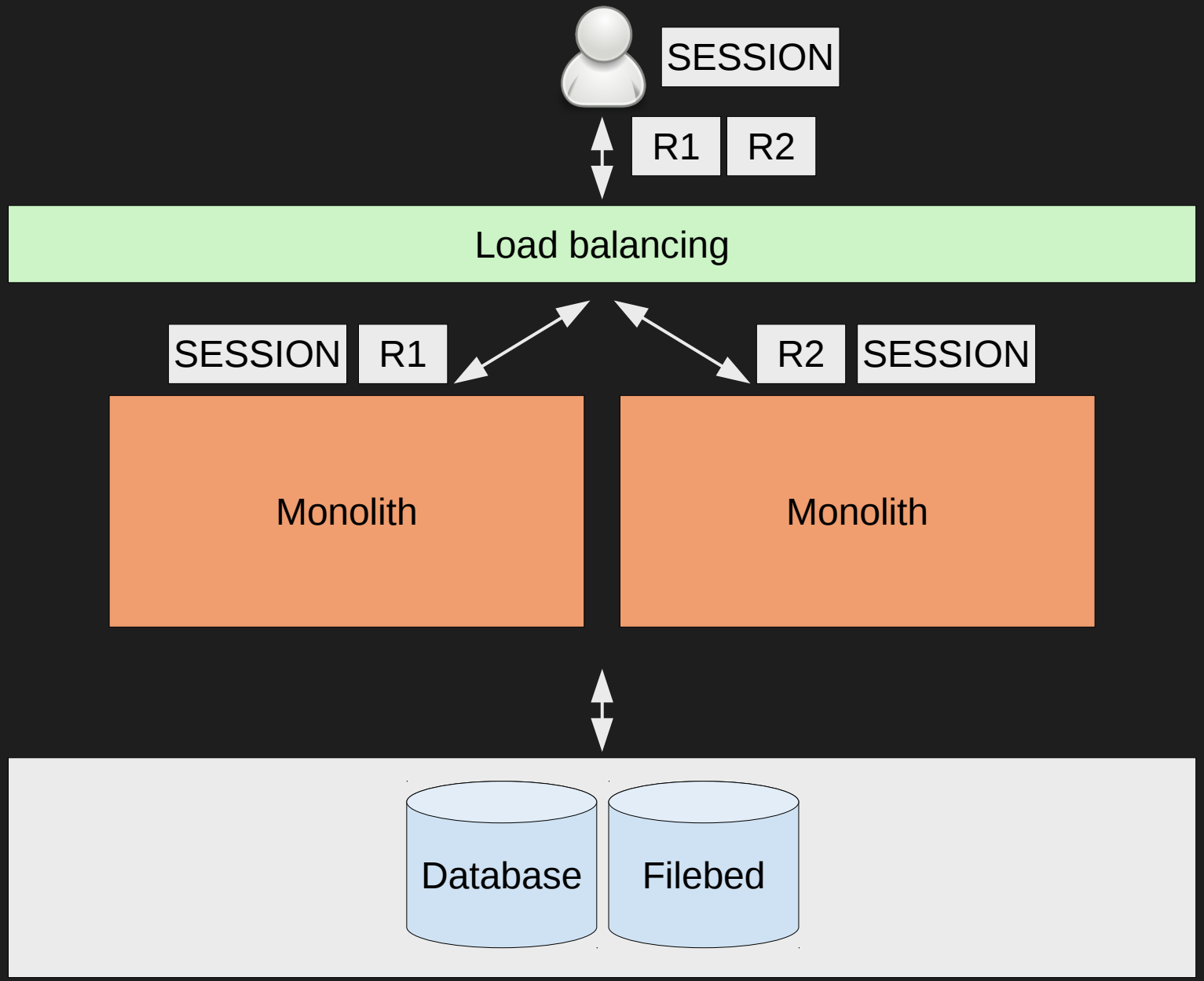


History > MUZIK 1.0 (Bach)

Save sessions in client-side

(shared-nothing architecture)

History > MUZIK 1.0 (Bach)



History > MUZIK 1.0 (Bach)

- ✔ Pros
  - ✔ Do not depend on Load balancing
  - ✔ Do not depend on Application server
  - ✔ Do not depend on Database
  - ✔ Do not depend on Shared disk
  - ✔ Do not depend on Cache server
  - ✔ Shared-nothing architecture

History > MUZIK 1.0 (Bach)

- ✔ 優點
  - ✔ 不依賴負載平衡器
  - ✔ 不依賴應用伺服器
  - ✔ 不依賴資料庫
  - ✔ 不依賴共享儲存庫
  - ✔ 不依賴快取伺服器
  - ✔ Shared-nothing architecture

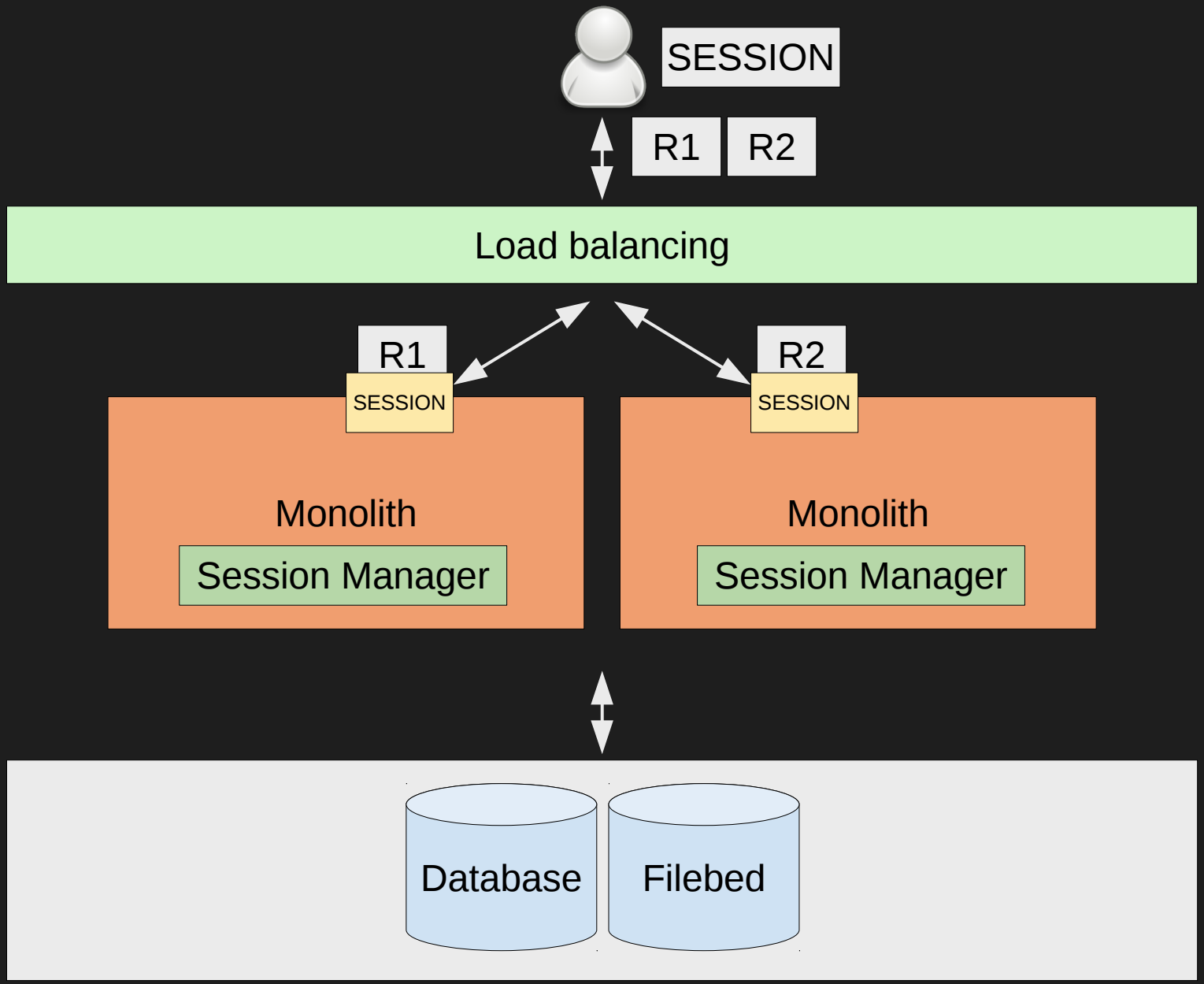
History > MUZIK 1.0 (Bach)

- ✔ Cons
  - ✔ Session size limited
  - ✔ Session need encode / decode (lower performance)
  - ✔ Less secure (depends on how to implement)
  - ✔ More package size (few bytes more)

History > MUZIK 1.0 (Bach)

- ✔ 缺點
  - ✔ Session 大小有限制
  - ✔ Session 需要編碼 / 反編碼 (效能耗損)
  - ✔ 安全性稍低 (視如何實作)
  - ✔ 耗費更多網路頻寬 (多了幾 bytes)

History > MUZIK 1.0 (Bach)



Migrations

- ✔ Vanilla PHP → Phalcon/MVC
  - ✔ MVC is HOT, and we need it by this time
  - ✔ Standard design pattern
  - ✔ Full stack for Web
  - ✔ High performance (C extension)
  - ✔ Well document material
  - ✔ Commit regularly
- ✔ Apache prefork → Nginx / PHP-FPM

遷移

- ✔ Vanilla PHP → Phalcon/MVC
  - ✔ MVC 很熱門，而且當下時間點很適合我們
  - ✔ 標準的設計模式
  - ✔ 網頁全端開發
  - ✔ 高效能 (基於 C 擴展)
  - ✔ 完整的文件
  - ✔ 官方更新頻繁
- ✔ Apache prefork → Nginx / PHP-FPM

Migrations

- ✔ Database schema re-design
  - ✔ The metadata for classical music is much more complex than others
    - ✔ Composer / Period
    - ✔ Album / Genre / Work / Movement / Track
    - ✔ Performer / Instrument
    - ✔ etc.

遷移

- ✔ 資料庫結構重新設計
  - ✔ 古典樂的後設資料非常複雜，比流行音樂複雜 N 倍
    - ✔ 作曲家 / 時期
    - ✔ 專輯 / 曲種 / 作品 / 樂章 / 音檔
    - ✔ 演奏家 / 樂器
    - ✔ 其它

Why we still use MySQL

- ✔ Easy / Mature than NoSQL / NewSQL
- ✔ Our team are familiar with MySQL
- ✔ Connection pool
  - ✔ Almost everyone told me connection pool is good
    - ✔ But can you tell me what is the drawbacks ?
  - ✔ Hello guys, PHP support connection pool

為什麼繼續使用 MySQL

- ✔ 目前業務場景下， RDBMS 相較於 NoSQL / NewSQL 更容易且成熟
- ✔ 團隊全員熟悉 MySQL
- ✔ Connection pool
  - ✔ 幾乎每個人都會告訴我 connection pool 很好
    - ✔ 但可以告訴我坑在哪嗎？
  - ✔ 嗨！開發者們 , PHP 其實支援 connection pool

Why we still use MySQL

- ✔ Connection pool
  - ✔ Not all application support connection pool
  - ✔ Connection pool can not know much about servers status / loading
  - ✔ Connection pool need to do resource cleanup when failed
    - ✔ If your application is complexity, you need to do yourself
  - ✔ Connection pool keep connection
    - ✔ Occupied server connection / thread cache

為什麼繼續使用 MySQL

- ✔ Connection pool
  - ✔ 不是所有應用程式都支援 connection pool
  - ✔ Connection pool 無法得知伺服器的狀態及承載
  - ✔ Connection pool 遭遇錯誤時，必須執行完整的資源清理
    - ✔ 如果你的應用很複雜，有自己的資源配置，那麼你必須自己清理回收
  - ✔ Connection pool 會保持連線
    - ✔ 佔用伺服器連線數及線程快取

Why we still use MySQL

- ✔ Connection pool
  - ✔ An architecture problems
    - ✔ If your database can keep 500 connections the most
    - ✔ Each of application server's connection pool is 100
    - ✔ In some cases, if you add the 6th application server, it almost cannot create any connection
    - ✔ The 'connection pool' variable is a cross-architecture problem
      - ✔ As mentioned above, connection pool usually cannot be adjusted depends on database status
      - ✔ If your have DBA and Backend team, the problem arise

為什麼繼續使用 MySQL

- ✔ Connection pool
  - ✔ 一個架構問題
    - ✔ 如果你的資料庫最多能保持 500 連線數
    - ✔ 每個應用伺服器的 connection pool 為 100
    - ✔ 某些情況下，當你增加第六台應用伺服器時，該伺服器幾乎無法建立任何連線
    - ✔ 'connection pool' 變數成為跨架構的全域變數
      - ✔ 如前所述，通常 connection pool 無法得知伺服器狀態，所以無法自調適
      - ✔ 如果成員又區分為 DBA 及後端兩組人馬時，這個問題會更嚴重

Why we still use MySQL

- ✔ MySQL
  - ✔ MySQL is threaded-base model
    - ✔ Support mass concurrency without connection pool
    - ✔ Fit short database connection
  - ✔ MySQL 5.7 connection overhead reduction
    - ✔ 5.6: 62.5%
    - ✔ 5.5: 40%

為什麼繼續使用 MySQL

- ✔ MySQL
  - ✔ MySQL 是線程模式
    - ✔ 不需 Connection pool 就可以支持高併發
    - ✔ 支持短連接使用資料庫
  - ✔ MySQL 5.7 建立連線的開銷更少
    - ✔ 是 5.6 版的 62.5%
    - ✔ 是 5.5 版的 40%

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

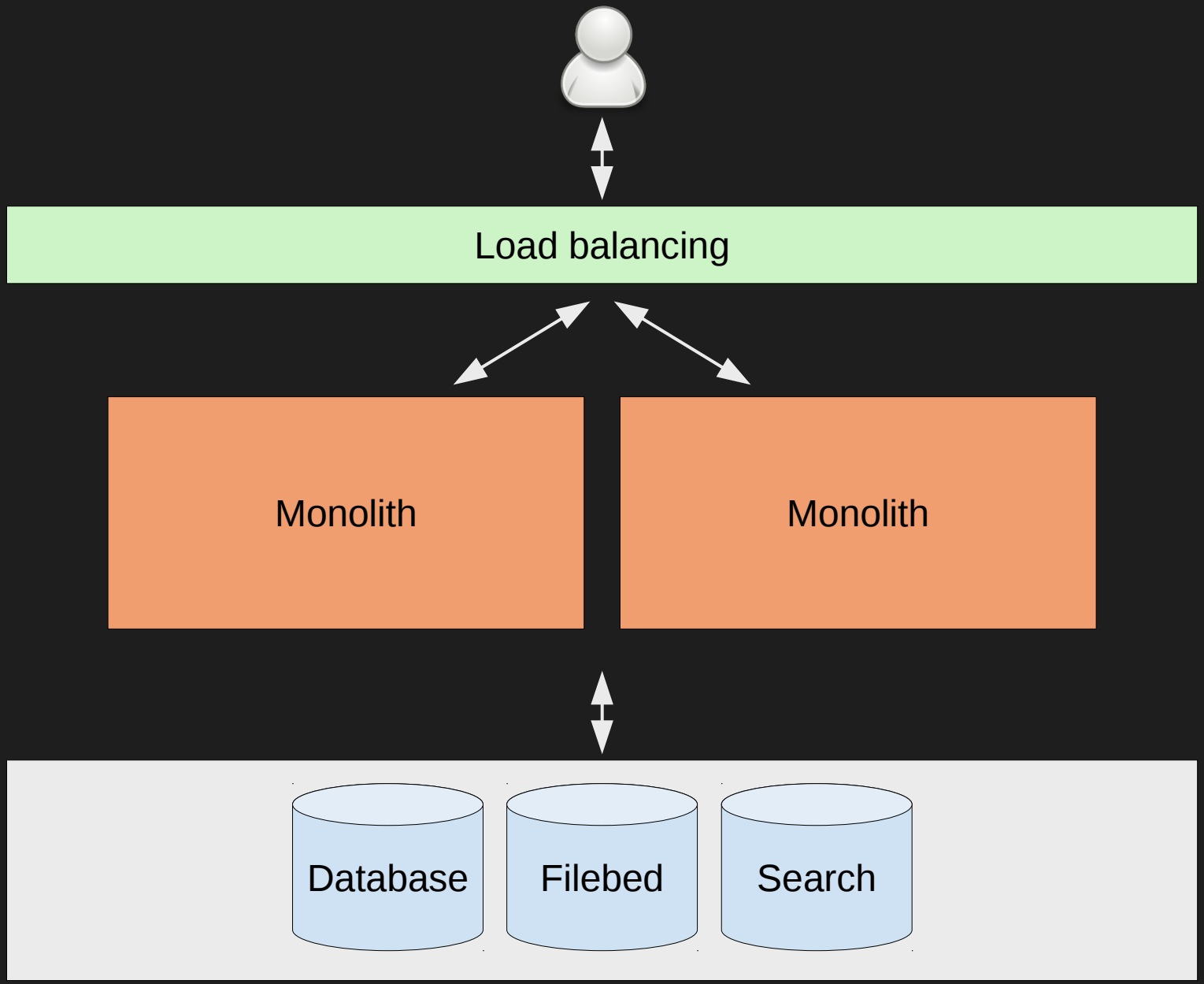
海頓 (Haydn)

交響曲之父

百納海川，方成其大

搜尋框架，各司其職

History > MUZIK 2.0 (Haydn)



New features

- ✔ ElasticSearch
  - ✔ Solr is mature, but ElasticSearch is easy
  - ✔ ElasticSearch is nature clustering / sharding
    - ✔ Eventually Consistent
  - ✔ For future purposes
    - ✔ Monitoring

遷移

- ✔ ElasticSearch
  - ✔ Solr 更成熟，但 ElasticSearch 比較容易上手
  - ✔ ElasticSearch 天生支援 clustering / sharding
    - ✔ Eventually Consistent (最終一致性)
  - ✔ 為了往後的架構設計
    - ✔ Monitoring (監控)

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

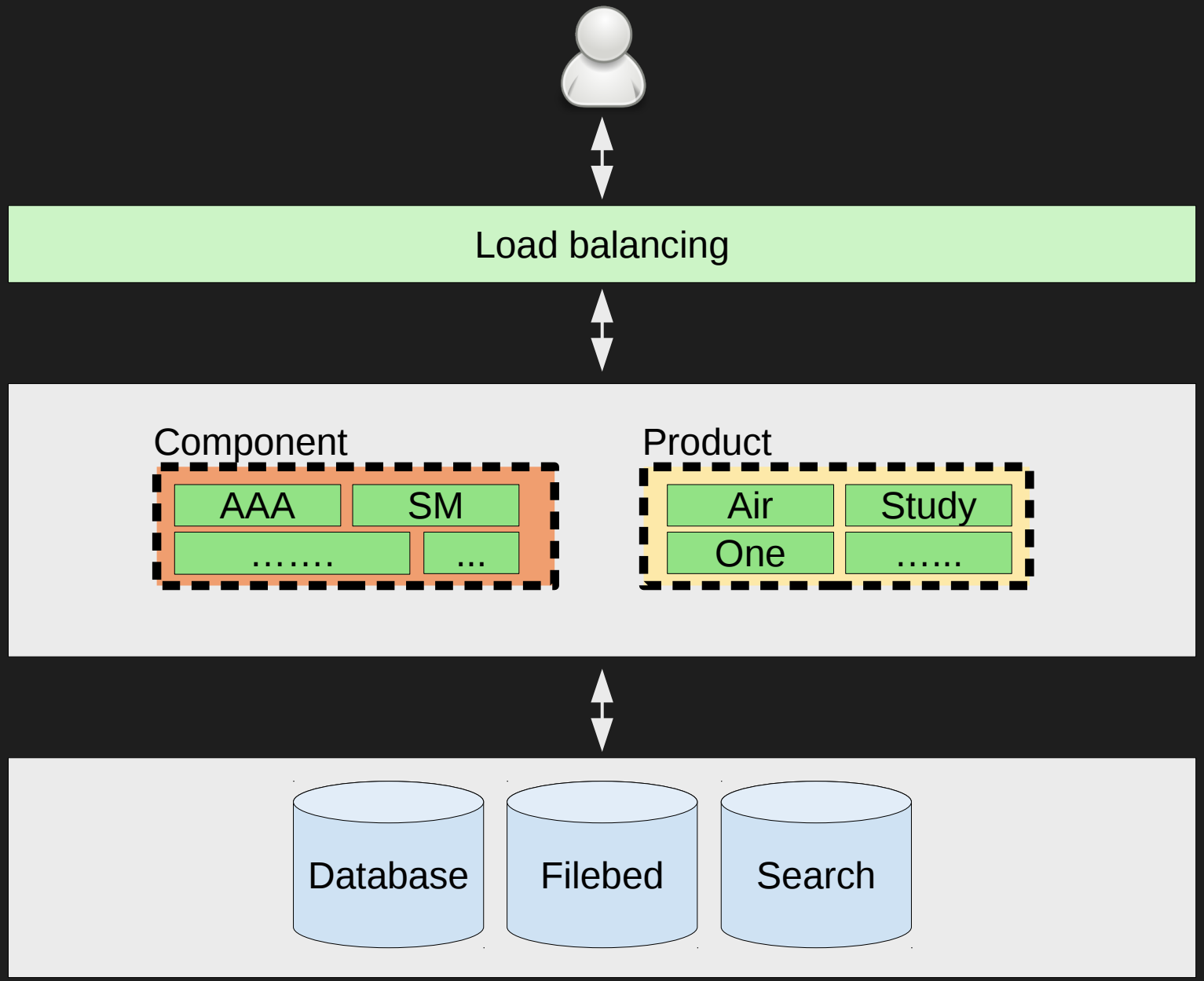
莫札特（Mozart）

音樂神童

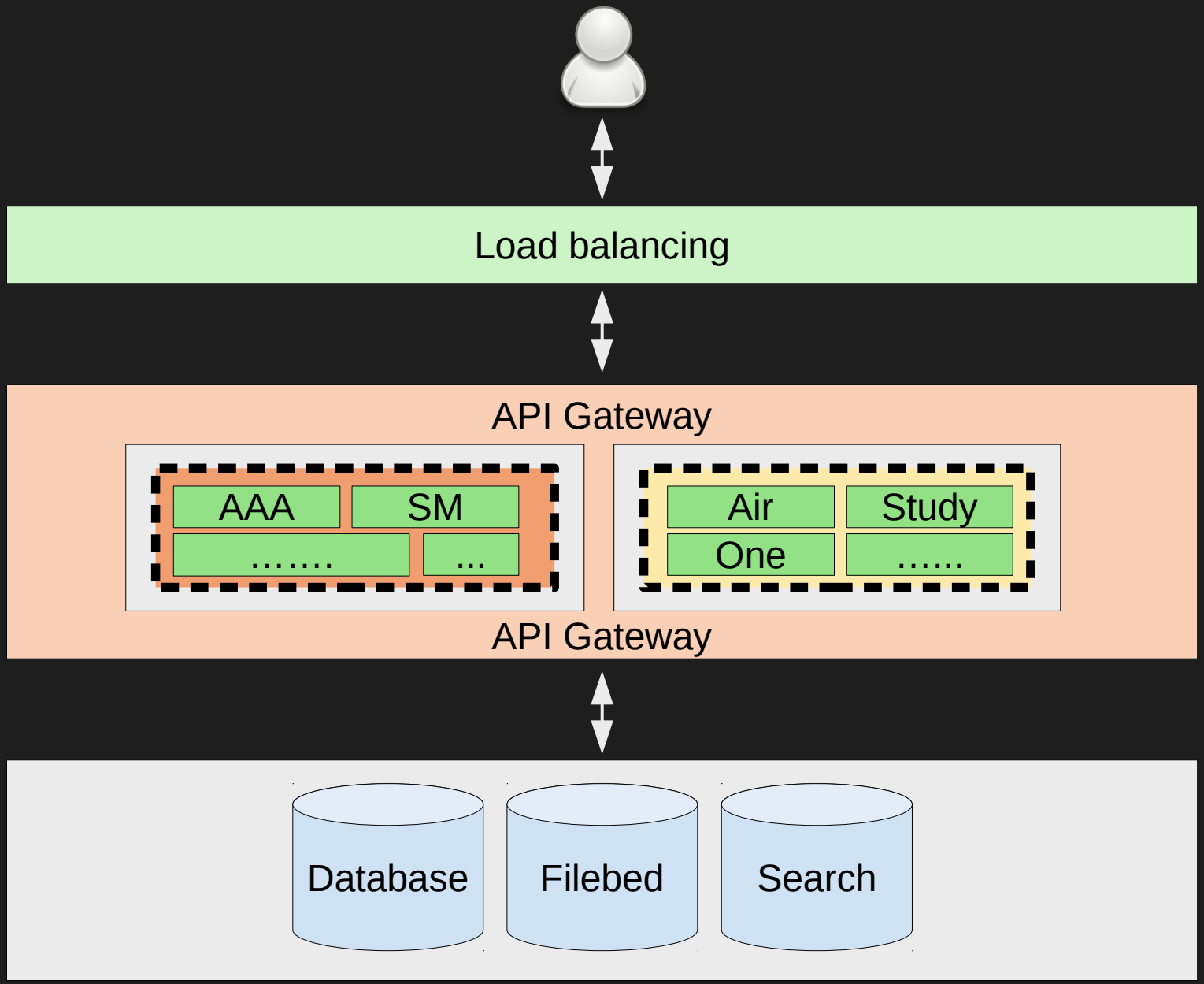
不朽傑作，分而治之

英年早逝，順應變遷

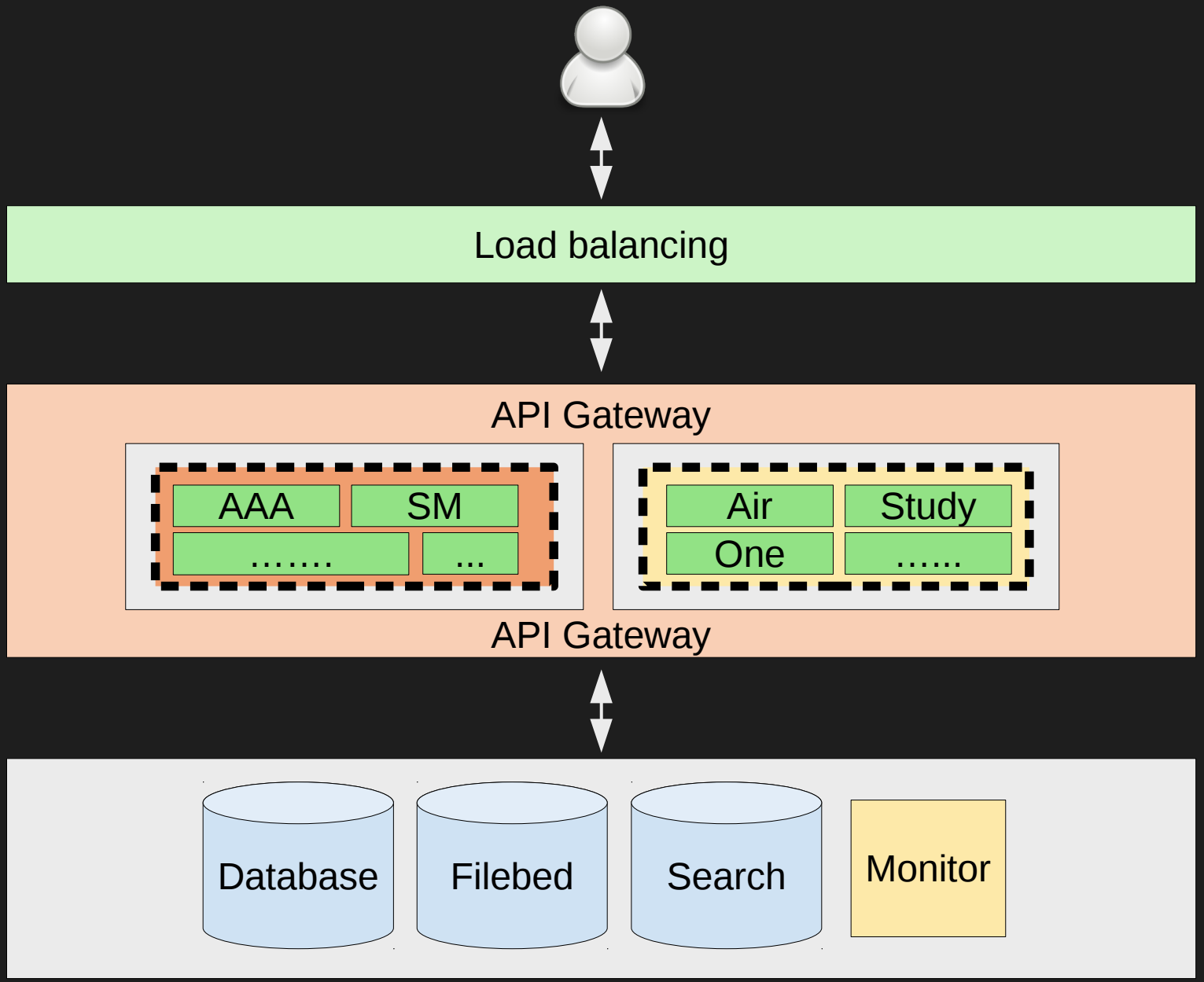
History > MUZIK 3.0 (Mozart)



History > MUZIK 3.0 (Mozart)



History > MUZIK 3.0 (Mozart)



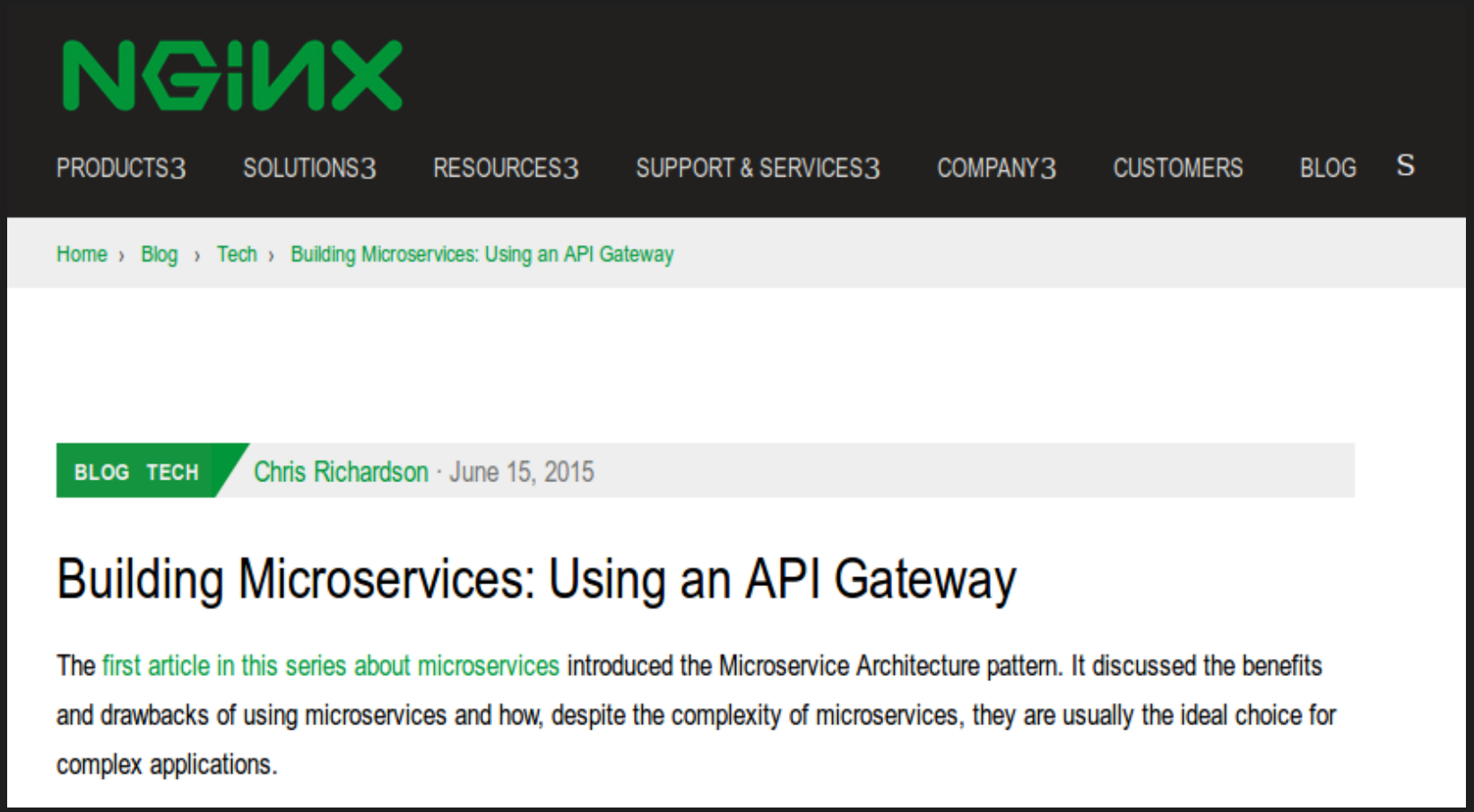
Migrations

- ✔ Phalcon → Phalcon Micro
  - ✔ Microservices
    - ✔ API Gateway (Q4 2014)
  - ✔ With MUZIK micro framework
    - ✔ Fast
    - ✔ Components
    - ✔ Secure
    - ✔ Shared-nothing architecture

遷移

- ✔ Phalcon → Phalcon Micro
  - ✔ Microservices
    - ✔ API Gateway (Q4 2014)
  - ✔ 融合 MUZIK 自主開發的微框架
    - ✔ 快
    - ✔ 元件化
    - ✔ 安全
    - ✔ Shared-nothing architecture

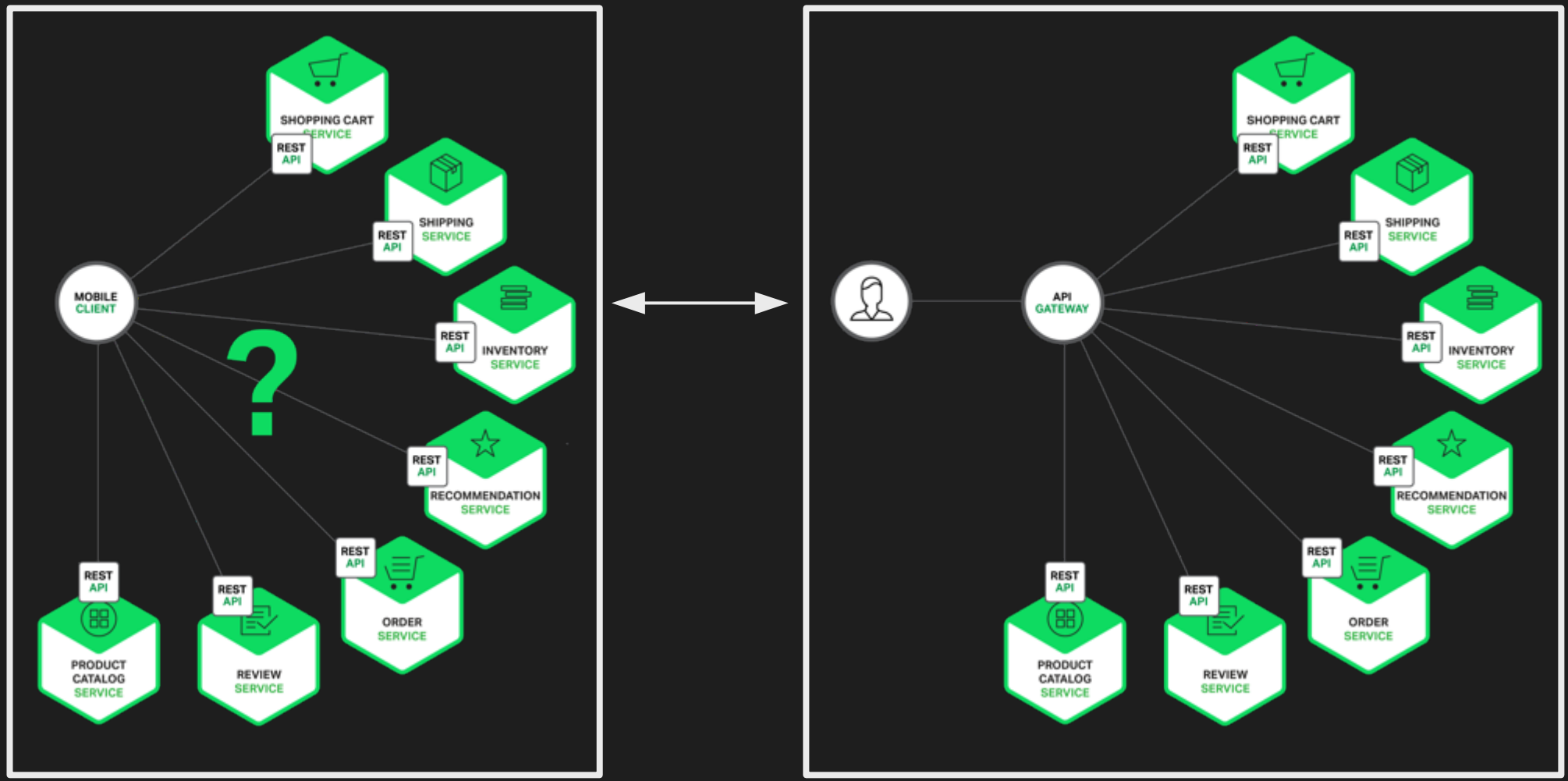
Migrations



2015-06-15

Ref: https://www.nginx.com/blog/building-microservices-using-an-api-gateway/

Migrations



Ref: https://www.nginx.com/blog/building-microservices-using-an-api-gateway/

Migrations

- ✔ Phalcon → Phalcon Micro
  - ✔ Microservices
    - ✔ API Gateway (Q4 2014)
      - ✔ Cons
        - ✔ Not 100% microservices
        - ✔ Carefully use it if will migrate to microservices in the future
          - ✔ Merge or separate microservices those face client directly

遷移

- ✔ Phalcon → Phalcon Micro
  - ✔ Microservices
    - ✔ API Gateway (Q4 2014)
      - ✔ 缺點
        - ✔ 並不是 100% microservices
        - ✔ 需特別注意未來架構演化的限制
          - ✔ 如未來需要合併或拆分那些直接面對客戶端的 microservices

Migrations

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ Requirements
      - ✔ Geo-replication
      - ✔ Automatic node joining
      - ✔ Scalability
      - ✔ Easy
      - ✔ Maintenance

遷移

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ 需求
      - ✔ Geo-replication (地域複製)
      - ✔ 自動增減節點
      - ✔ 擴展性
      - ✔ 易用性
      - ✔ 維護性

Migrations

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ Pros
      - ✔ Read scalability
      - ✔ Write scalability (?)
      - ✔ etc.

遷移

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ 優點
      - ✔ Read scalability (讀擴展性)
      - ✔ Write scalability (寫擴展性？)
      - ✔ 其它

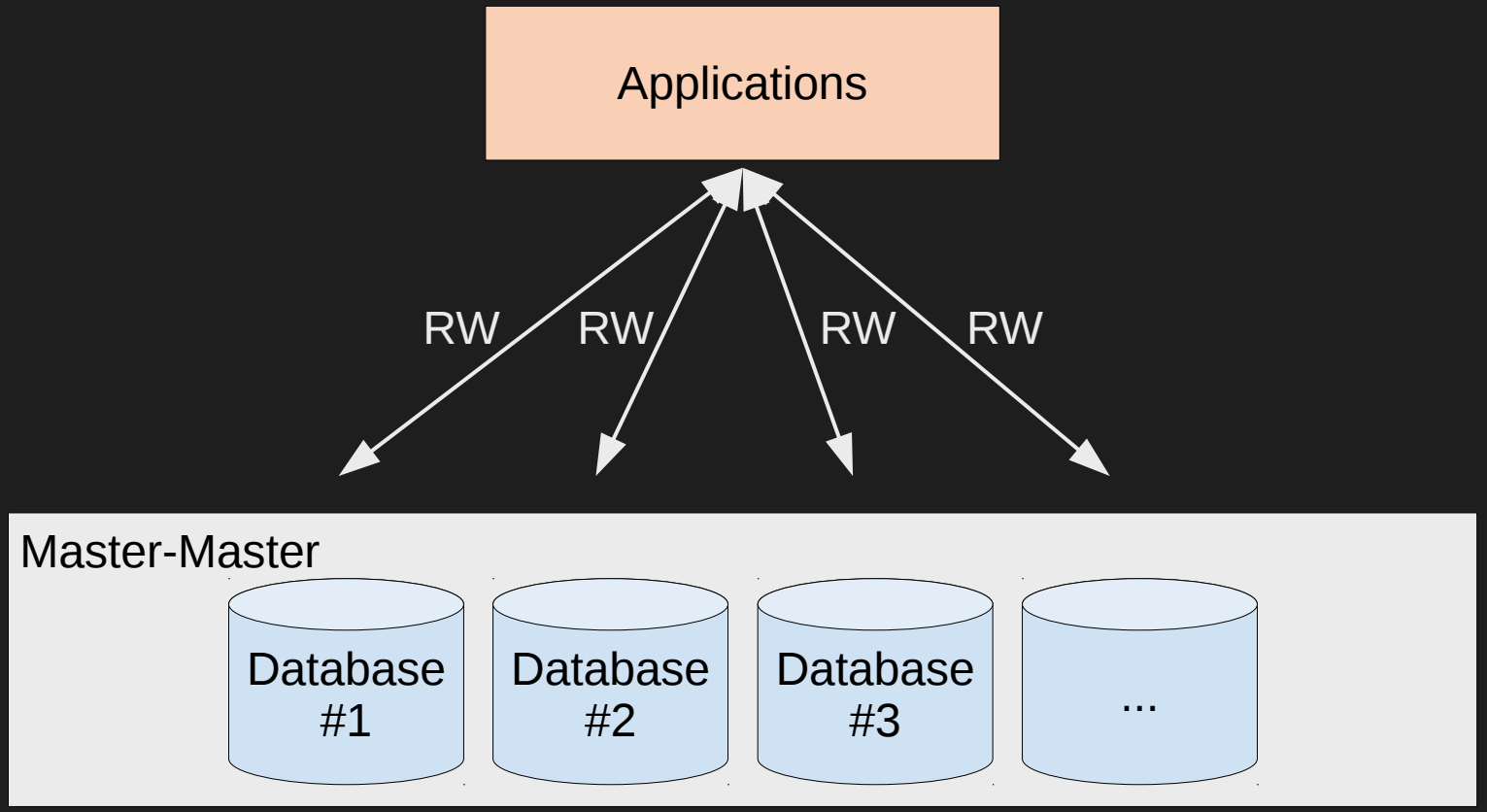
Migrations

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ Cons
      - ✔ Split brain
      - ✔ Commit latency
      - ✔ Hot-spot
      - ✔ Deadlock on commit
        - ✔ More nodes increase the transaction rollback rate

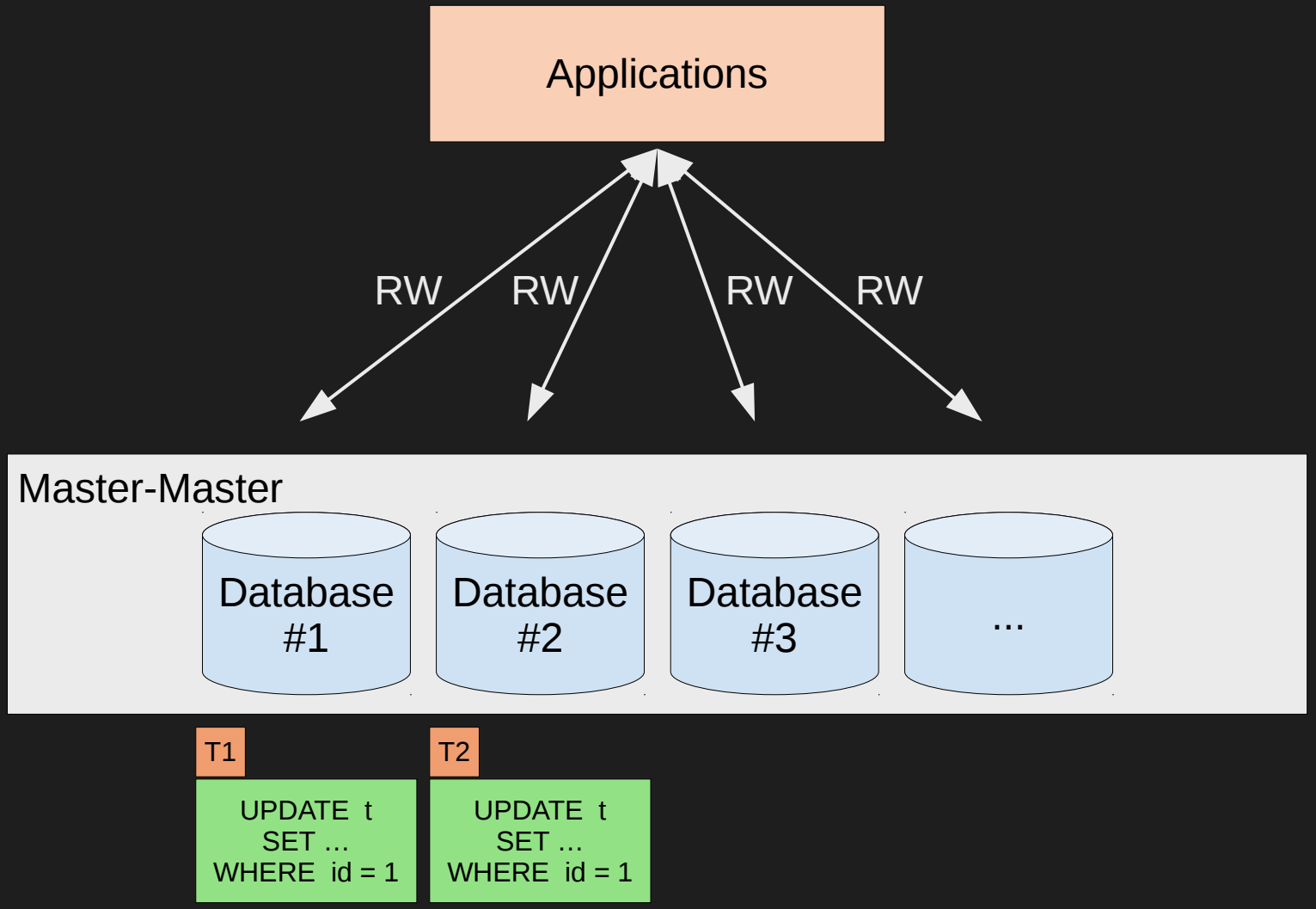
遷移

- ✔ MySQL 5.6 → MySQL 5.6 Master-Master Cluster
  - ✔ Master-Master
    - ✔ 缺點
      - ✔ Split brain (腦裂)
      - ✔ Commit latency (提交延遲)
      - ✔ Hot-spot (熱點)
      - ✔ Deadlock on commit (提交死鎖)
        - ✔ 當節點愈多，交易回滾機率愈高

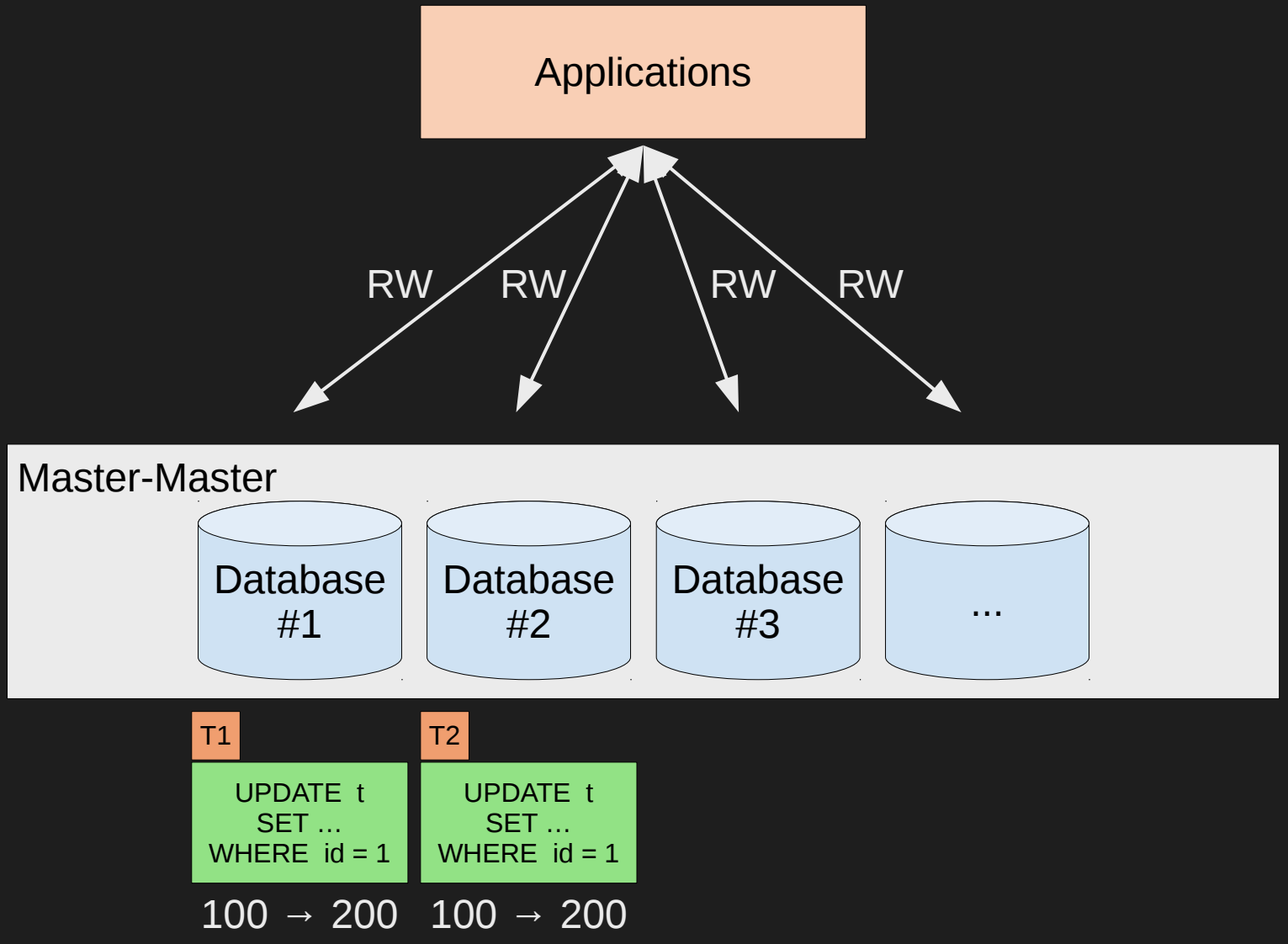
History > MUZIK 3.0 (Mozart)



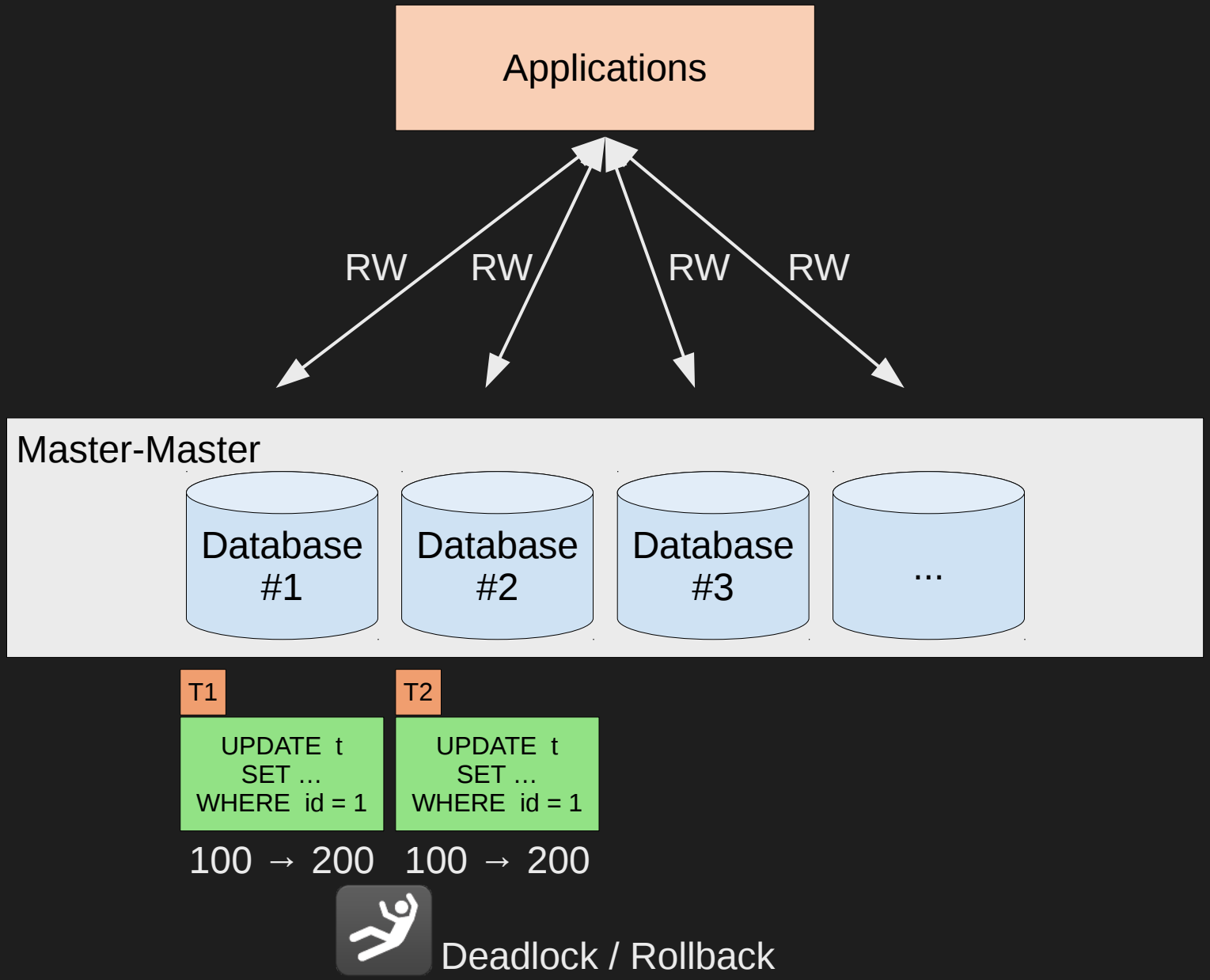
History > MUZIK 3.0 (Mozart)



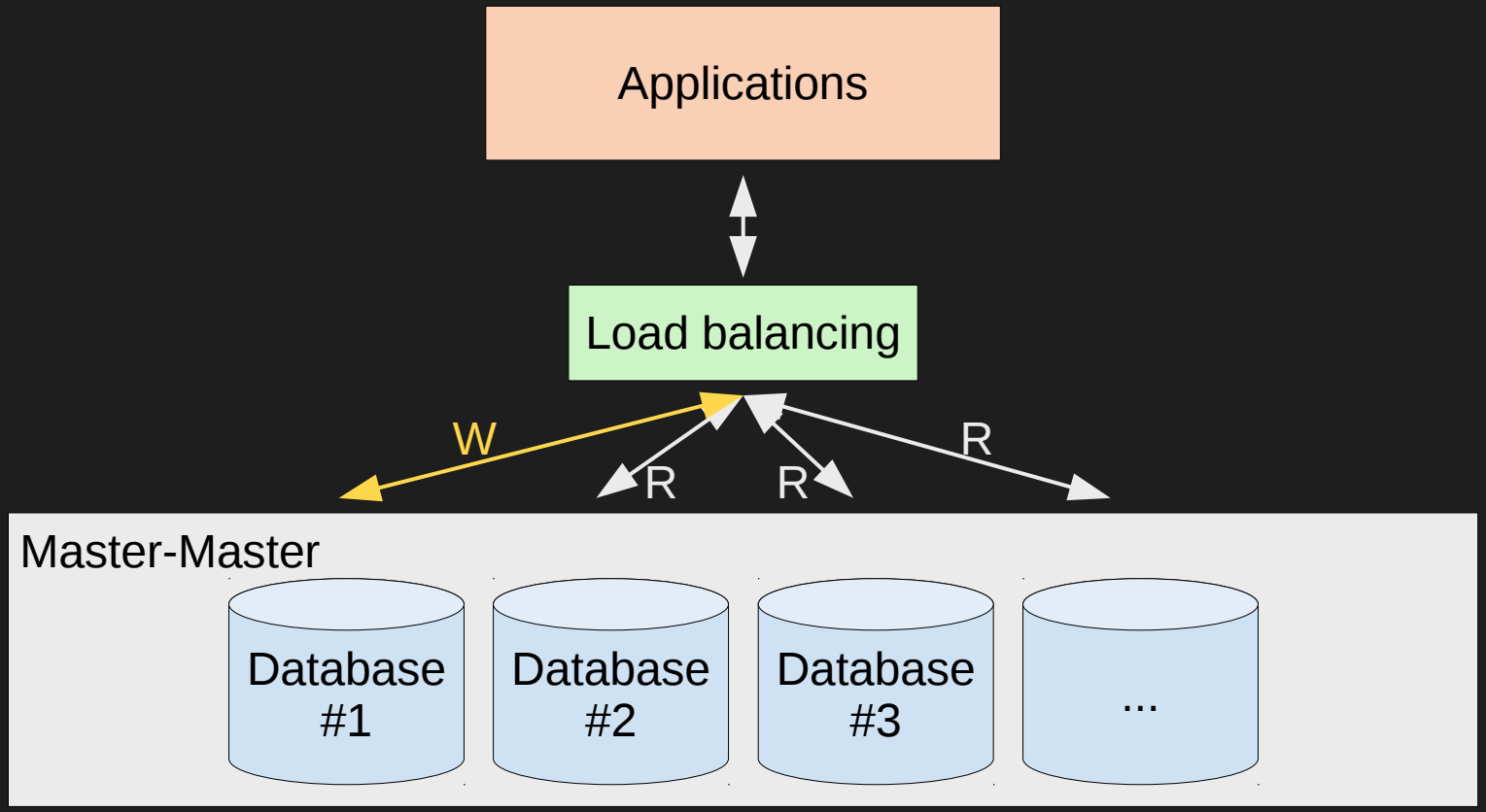
History > MUZIK 3.0 (Mozart)



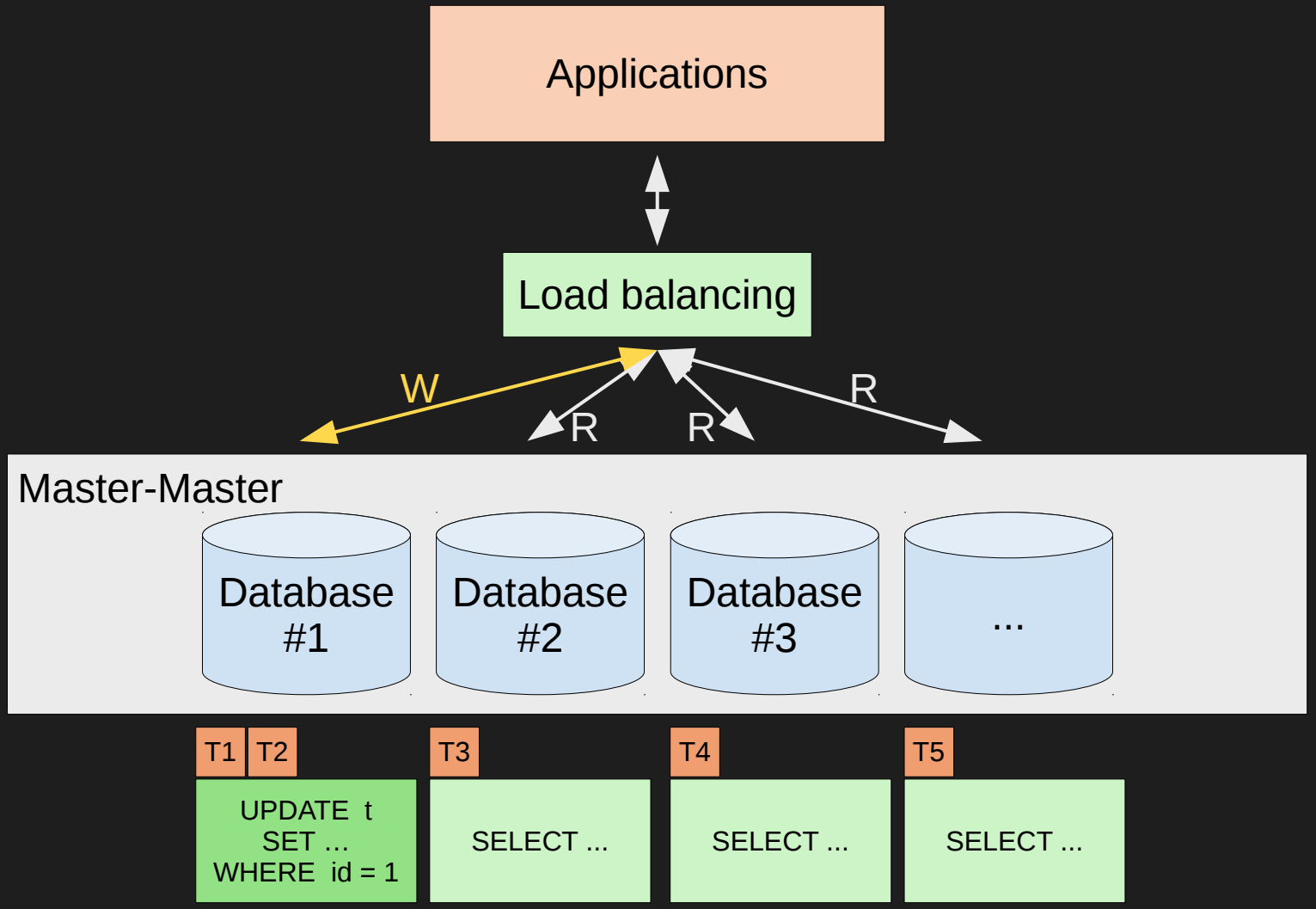
History > MUZIK 3.0 (Mozart)



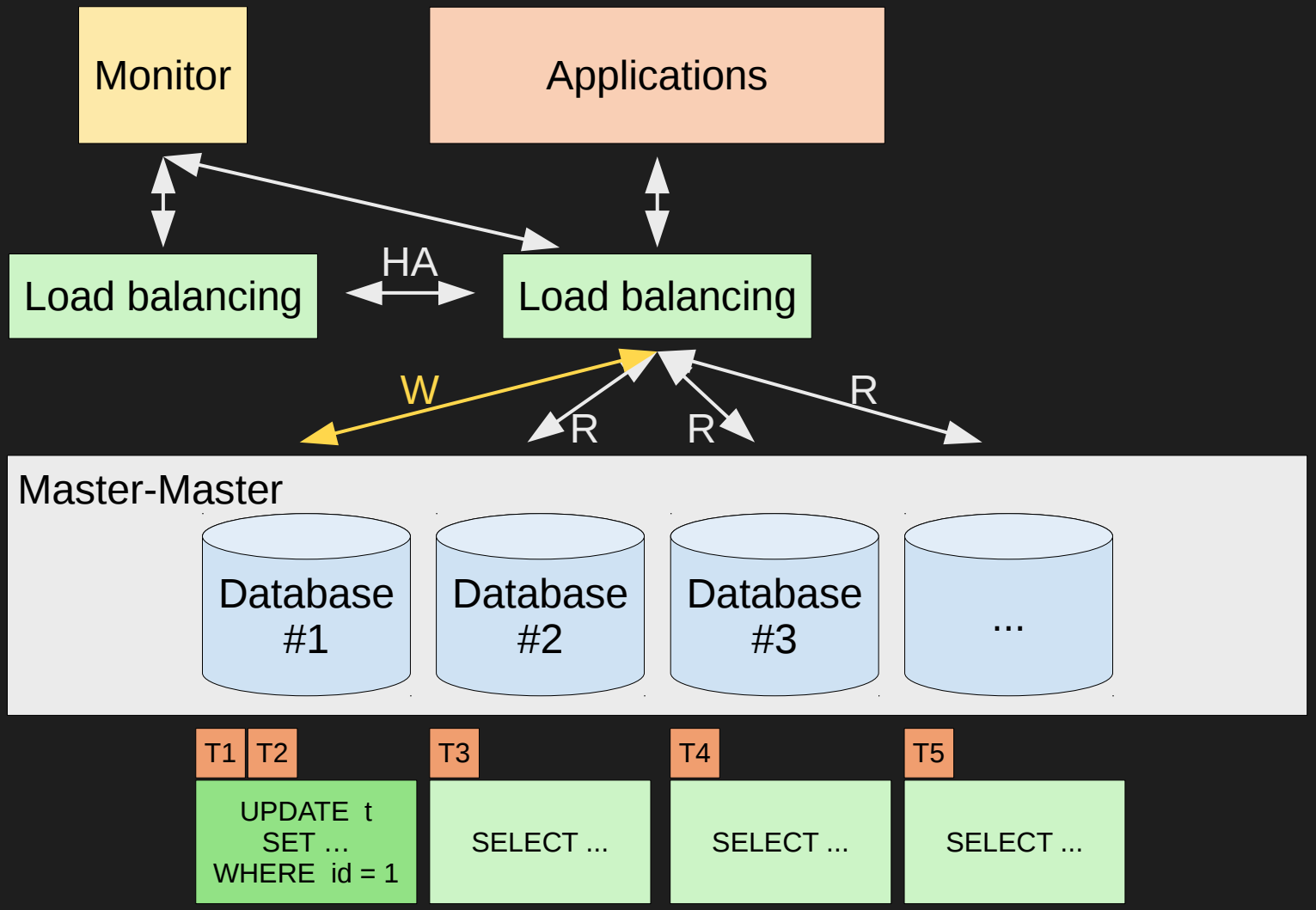
History > MUZIK 3.0 (Mozart)



History > MUZIK 3.0 (Mozart)



History > MUZIK 3.0 (Mozart)



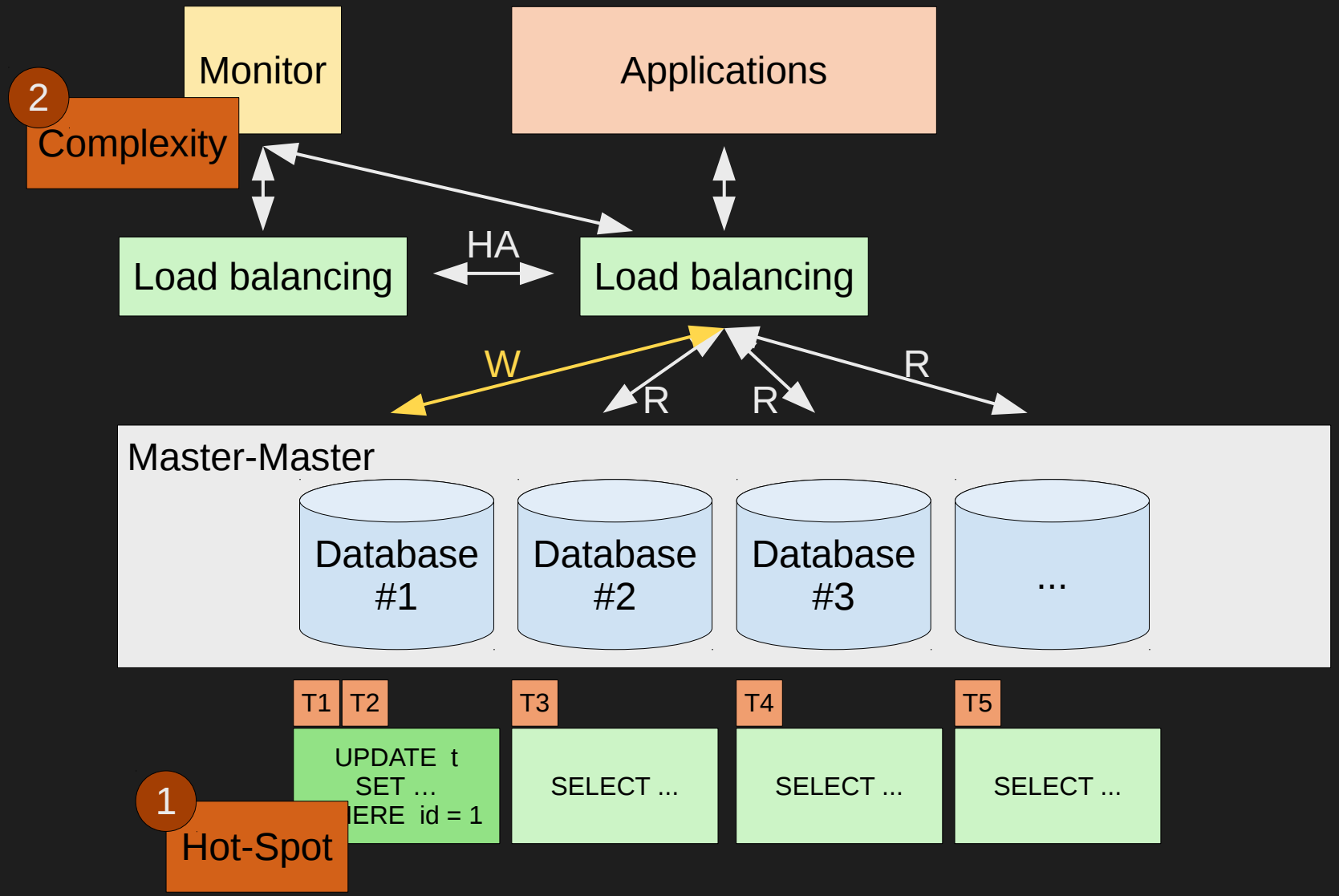
History > MUZIK 3.0 (Mozart)

Percona XtraDB Cluster: Multi-node writing and Unexpected deadlocks
2012-08-17
https://www.percona.com/blog/2012/08/17/percona-xtradb-cluster-multi-node-writing-and-unexpected-deadlocks/

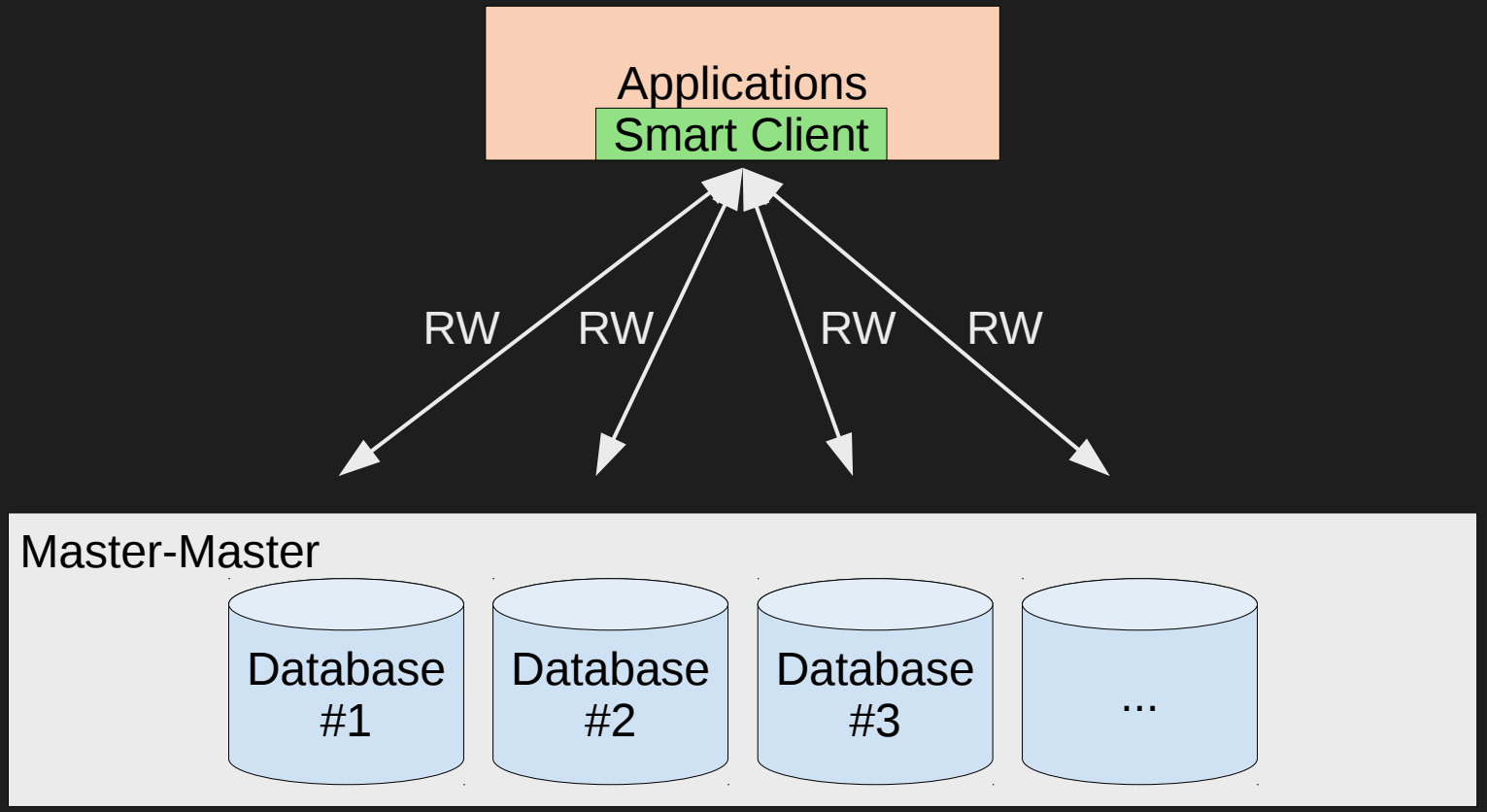
Avoiding Deadlocks in Galera - Set up HAProxy for single-node writes and multi-node reads
2013-09-17
http://www.severalnines.com/blog/avoiding-deadlocks-galera-set-haproxy-single-node-writes-and-multi-node-reads

Optimizing Percona XtraDB Cluster for write hotspots
2015-06-03
https://www.percona.com/blog/2015/06/03/optimizing-percona-xtradb-cluster-write-hotspots/

History > MUZIK 3.0 (Mozart)



History > MUZIK 3.0 (Mozart)



Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

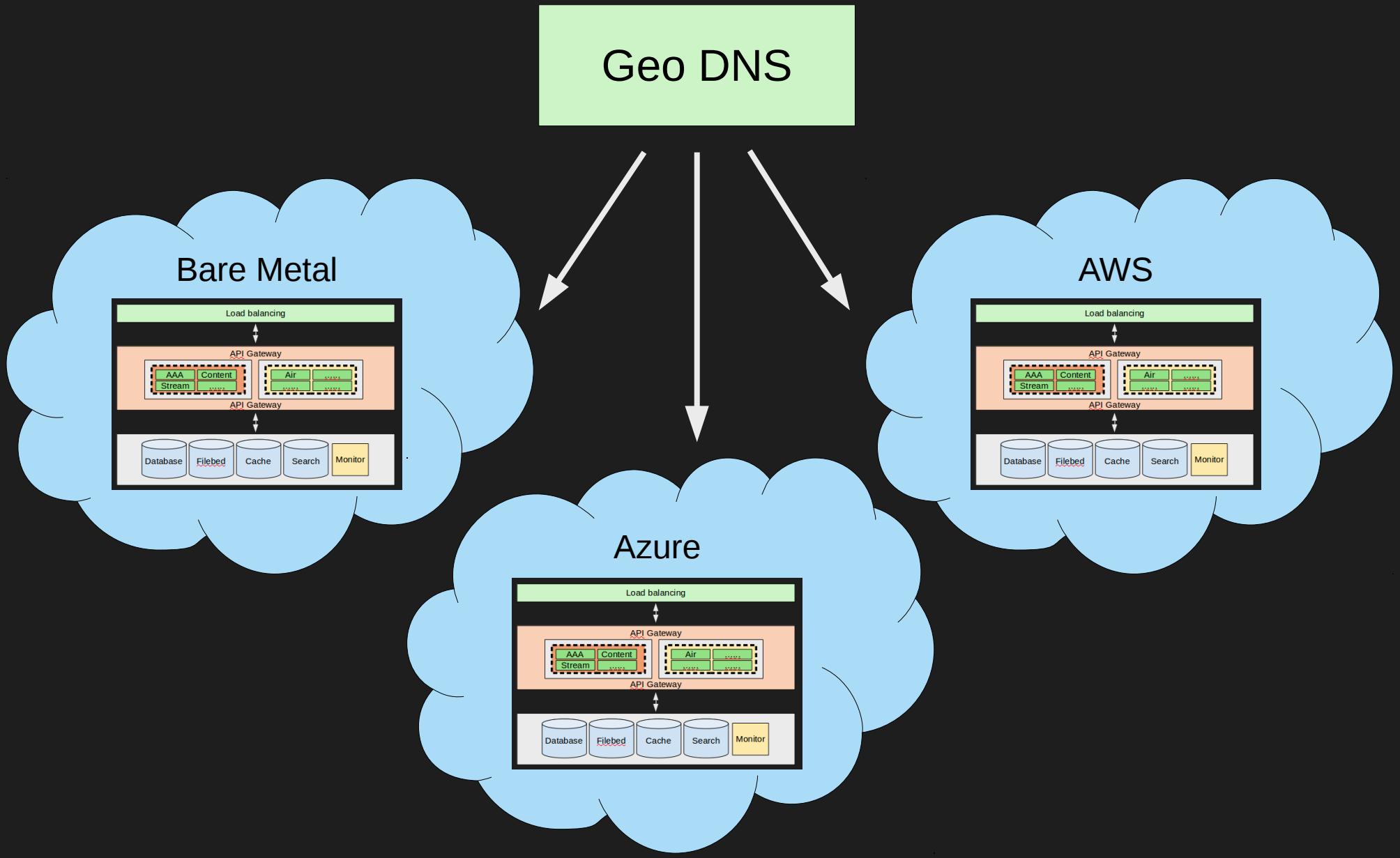
貝多芬 (Beethoven)

音樂神童

承襲精髓，造就極限

如飢似渴，深蹲練功

History > MUZIK 4.0 (Beethoven, +ing)



Why not use Docker ?

- ✔ Docker is not yet widely in production
  - ✔ http://sirupsen.com/production-docker/
  - ✔ http://www.theplatform.net/2015/09/29/why-containers-at-scale-is-hard/
- ✔ Docker is not stable in our test environment
- ✔ We use Docker lots in our develop environment
- ✔ Performance loss on single machine in our testing
  - ✔ Single Docker lost 8%
  - ✔ Two Dockers lost 31%
  - ✔ Four Dockers lost 51%

為何不使用 Docker ?

- ✔ Docker 為什麼還未在生產環境上普及
  - ✔ http://sirupsen.com/production-docker/
  - ✔ http://www.theplatform.net/2015/09/29/why-containers-at-scale-is-hard/
- ✔ Docker 在我們的測試環境中並不穩定，偶會無預警當機
- ✔ 我們仍然在我們的開發環境中大量使用
- ✔ 效能損失 (單機 / 我們的場景測試環境)
  - ✔ 單台 Docker 在單機時，損失 8 %
  - ✔ 雙台 Docker 在單機時，共損失 31%
  - ✔ 四台 Docker 在單機時，共損失 51%

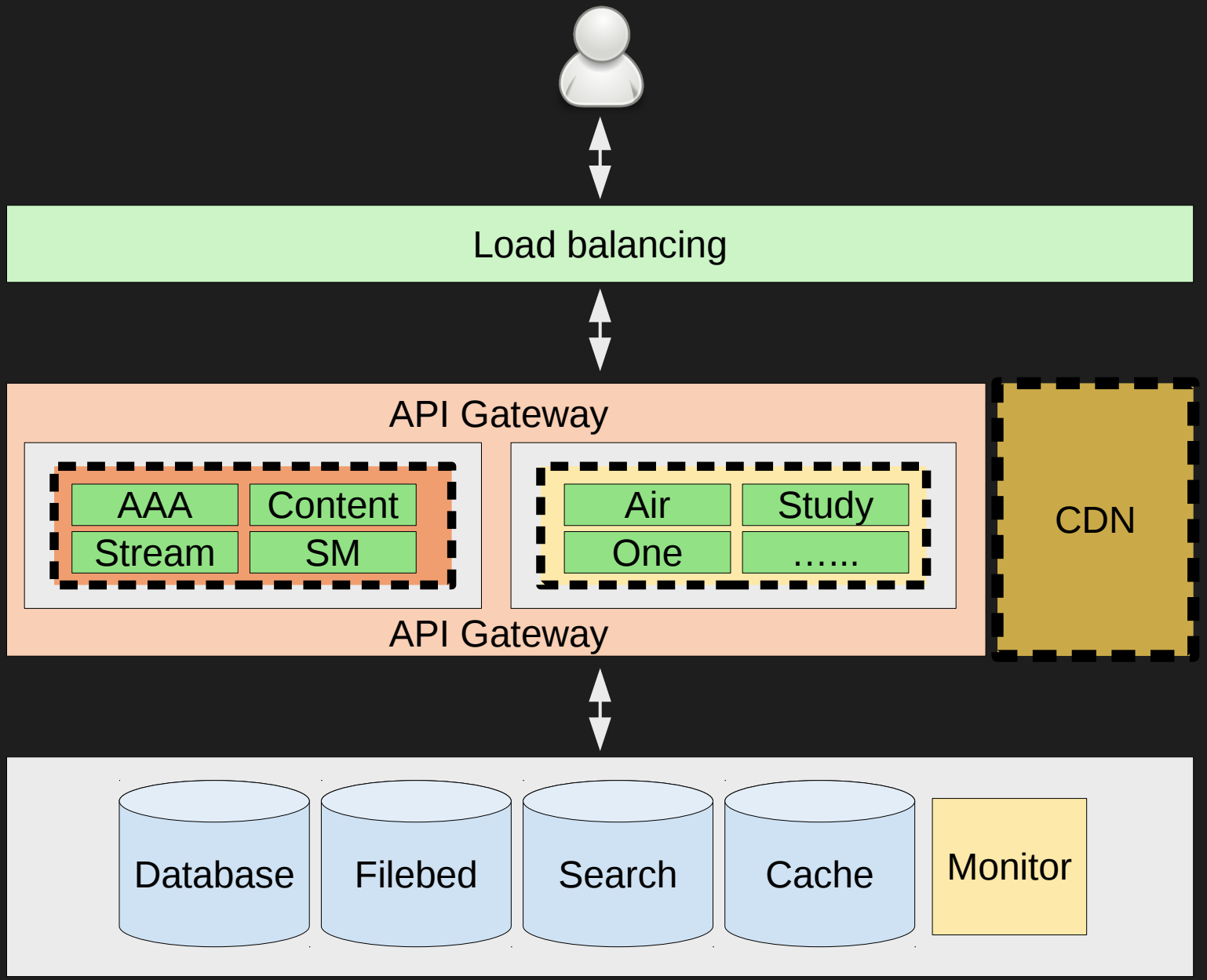
Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

History > MUZIK Next (maybe...)



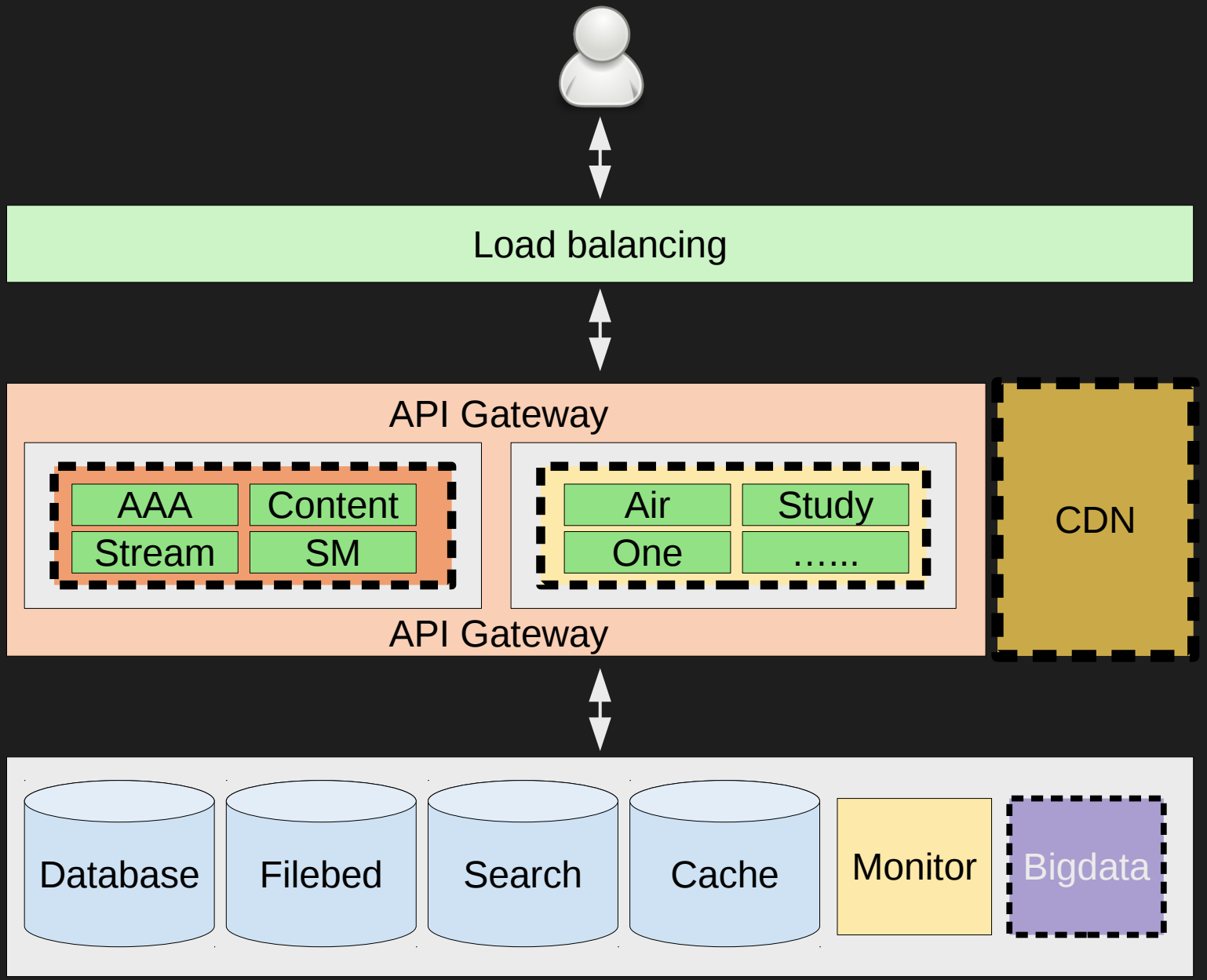
History > MUZIK Next (maybe...)

- ✔ CDN
  - ✔ We are building our own CDN
  - ✔ Hybrid MUZIK CDN with third-party services

History > MUZIK Next (maybe...)

- ✔ CDN
  - ✔ 我們一直研究並嘗試自主研發 CDN
  - ✔ 混用自主研發 CDN 及第三方的服務

History > MUZIK Next (maybe...)



History > MUZIK Next (maybe...)

- ✔ BigData
  - ✔ We are building our own BigData
  - ✔ Hybrid MUZIK BigData with third-party services

History > MUZIK Next (maybe...)

- ✔ BigData
  - ✔ 我們一直研究並嘗試自主研發 BigData
  - ✔ 混用自主研發 BigData 及第三方的服務

History > MUZIK Next (maybe...)

- ✔ BigData
  - ✔ Questions
    - ✔ Spark is faster than Hadoop ? What is the trade-off ?
    - ✔ What is the limit of Spark ?
    - ✔ The difference between Spark streaming and Storm streaming ?
    - ✔ What is the limit of Amazon Redshift ?

History > MUZIK Next (maybe...)

- ✔ BigData
  - ✔ 問題
    - ✔ Spark 確定比 Hadoop 快嗎 ？ 請問若有的話，是拿什麼條件交換 ？
    - ✔ Spark 的限制是什麼 ？
    - ✔ Spark streaming 與 Storm streaming 本質上的差異 ？
    - ✔ Amazon Redshift 的限制是什麼 ？

Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

PHP 7 ?

- ✔ PHP 7 ? HHVM ?
  - ✔ Hybrid PHP 7 / HHVM, but most PHP 7
- ✔ Migrating from PHP 5.6.x to PHP 7.0.x
  - ✔ http://php.net/manual/en/migration70.php
- ✔ PHP 7 Compatibility Checker
  - ✔ https://github.com/sstalle/php7cc

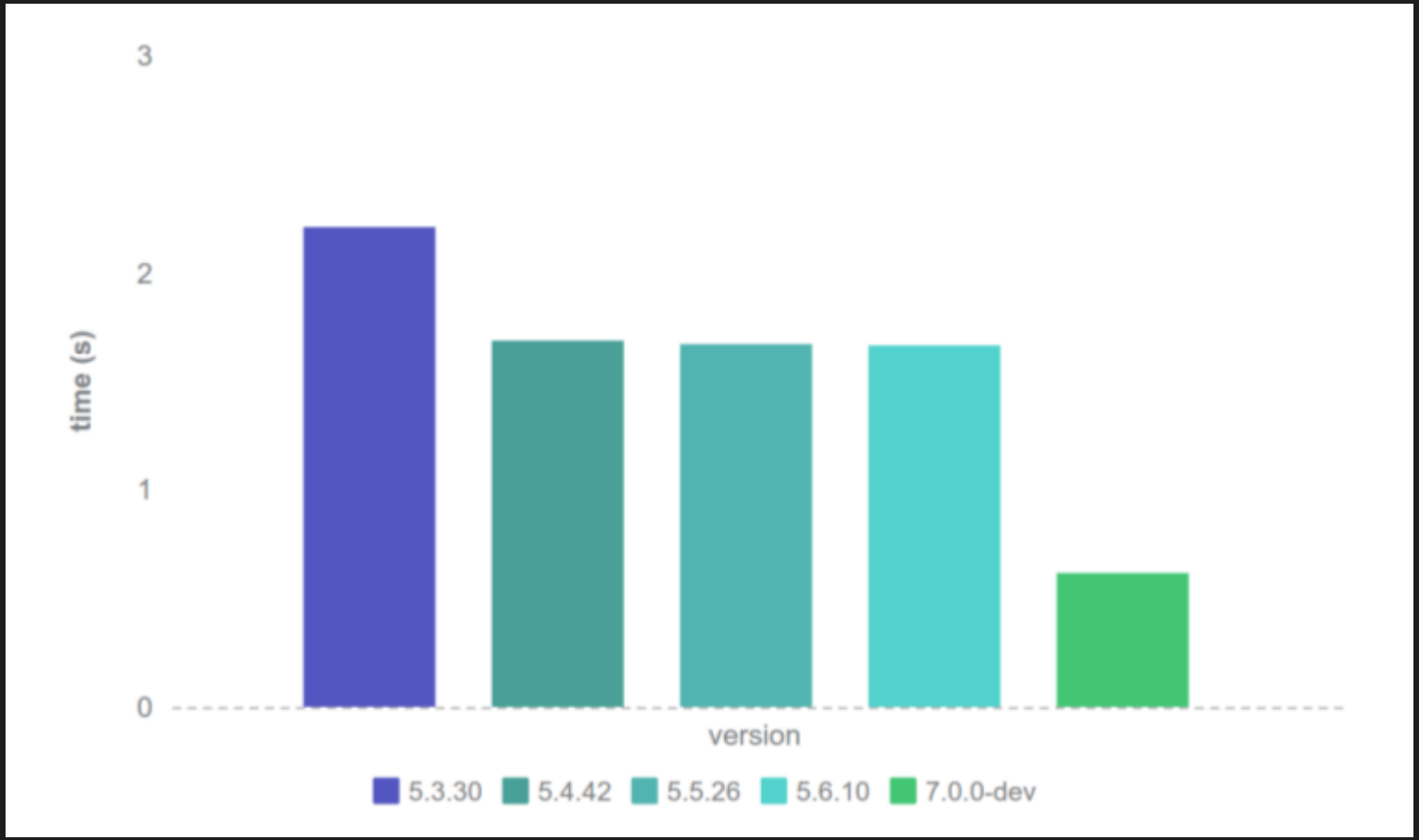
- ✔ Architecture painless (but code?)

PHP 7 ?

- ✔ PHP 7 ? HHVM ?
  - ✔ 混用 PHP 7 與 HHVM ，但大多都是 PHP 7
- ✔ Migrating from PHP 5.6.x to PHP 7.0.x
  - ✔ http://php.net/manual/en/migration70.php
- ✔ PHP 7 Compatibility Checker
  - ✔ https://github.com/sstalle/php7cc

- ✔ 架構無痛（但程式碼？）

PHP 7 ?



Agenda

- ✔ Prologue
- ✔ History of MUZIK architecture
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 architecture design rules

議程

- ✔ 楔子
- ✔ MUZIK 架構演化史
  - ✔ MUZIK Zero
  - ✔ MUZIK 1.0 (Bach)
  - ✔ MUZIK 2.0 (Haydn)
  - ✔ MUZIK 3.0 (Mozart)
  - ✔ MUZIK 4.0 (Beethoven)
  - ✔ MUZIK Next
- ✔ PHP 7 ?
- ✔ 15 條架構設計心法

15 architecture design rules

01. 冪等操作設計
Idempotent
02. 分層低耦設計
Decoupling
03. 服務節流設計
Throttling
04. 熱點緩衝設計
Leveraging
05. 服務同質檢測
Immutablility
06. 服務降級設計
Degrade
07. 拓撲調優設計
Topology
08. 日誌追蹤設計
Tracing
09. 監控警報設計
Monitoring
10. 自動修復設計
Self-healthy
11. 異常預測設計
Anomaly detection
12. 持續灰度發布
Continuous deployment
13. 自動擴展設計
Auto-scaling
14. 異地多活設計
Multi regional resiliency
15. 混沌破壞設計
Chaos tolerance

Feedback

勘誤回報

yftzeng@gmail.com

End

結語

① 架構先決。

② 沒有完美的架構，只有最適的架構。

③ 架構是演進的，預想但不過早調優。

每一個華麗架構背後，
都有一堆齷齪的實現。

朕知道了

We ❤ PHP

Shared-nothing architecture
Monolith → Microservice
Phalcon → Phalcon Micro
Caching !?
Connection pool !?
Extension (Zephir) / HHVM / PHP 7 !?