非同步系統的服務水準保證

淺談非同步系統的 SLO 設計

吳剛志 / Andrew Wu (91APP 首席架構師)

講師簡介

講師簡介

TECH-DAY

現任 91APP 首席架構師，負責帶領架構設計部，執行架構改善，以及跨團隊的大型專案推進任務。

從 2016 至今，榮獲多次 Microsoft MVP 微軟最有價值專家，並且多次擔任 Microsoft TechDays, DevOpsDays Taiwan, .NET Conf 等大型研討會講師。

平日喜歡談論各種軟體開發與設計的大小事，並持續在部落格上分享過去 20 年來，累積的大型軟體開發與設計經驗。喜歡研究各種技術背後的原理與實作細節，期許自己做個優秀的系統架構師。

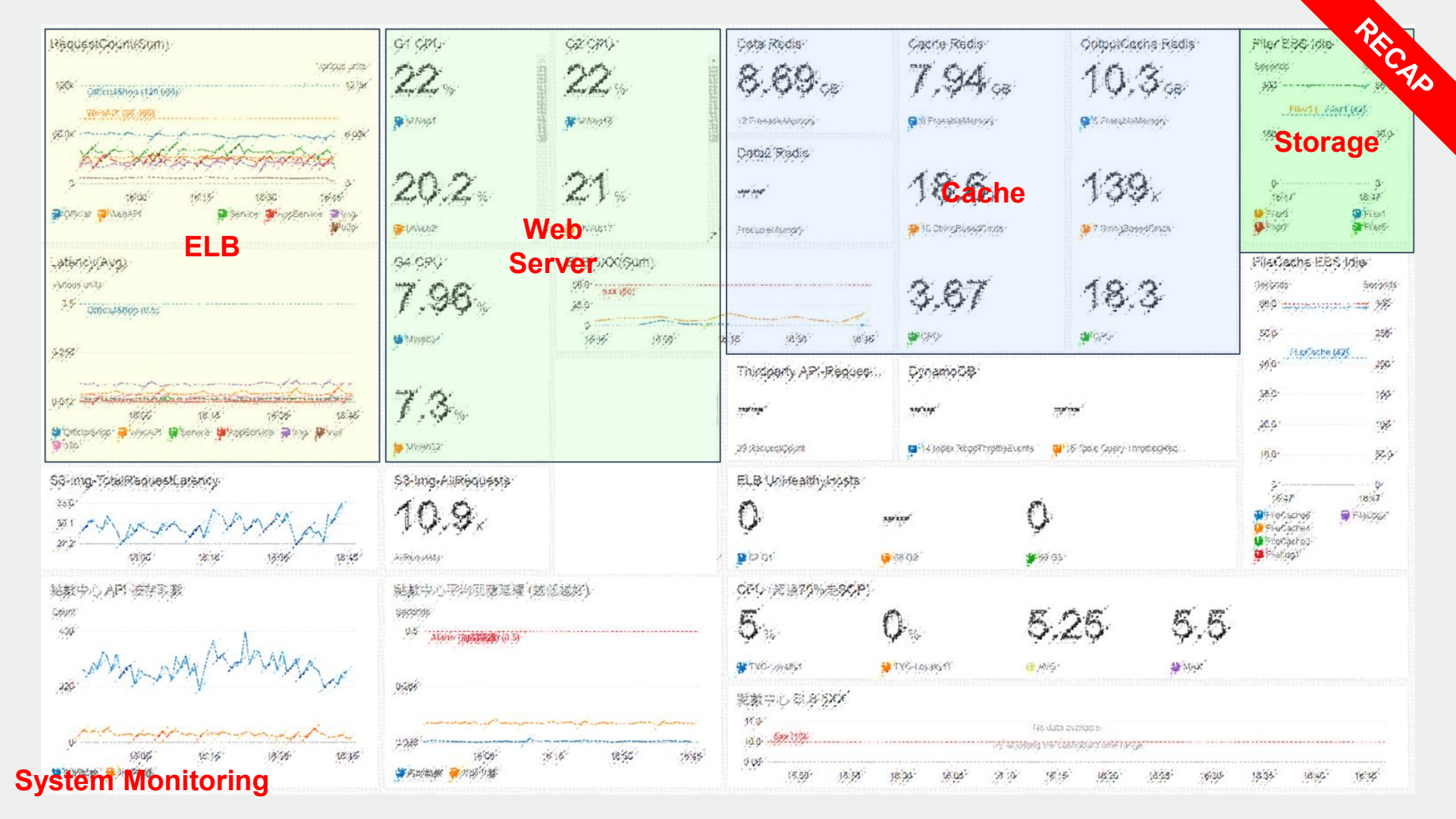
前言

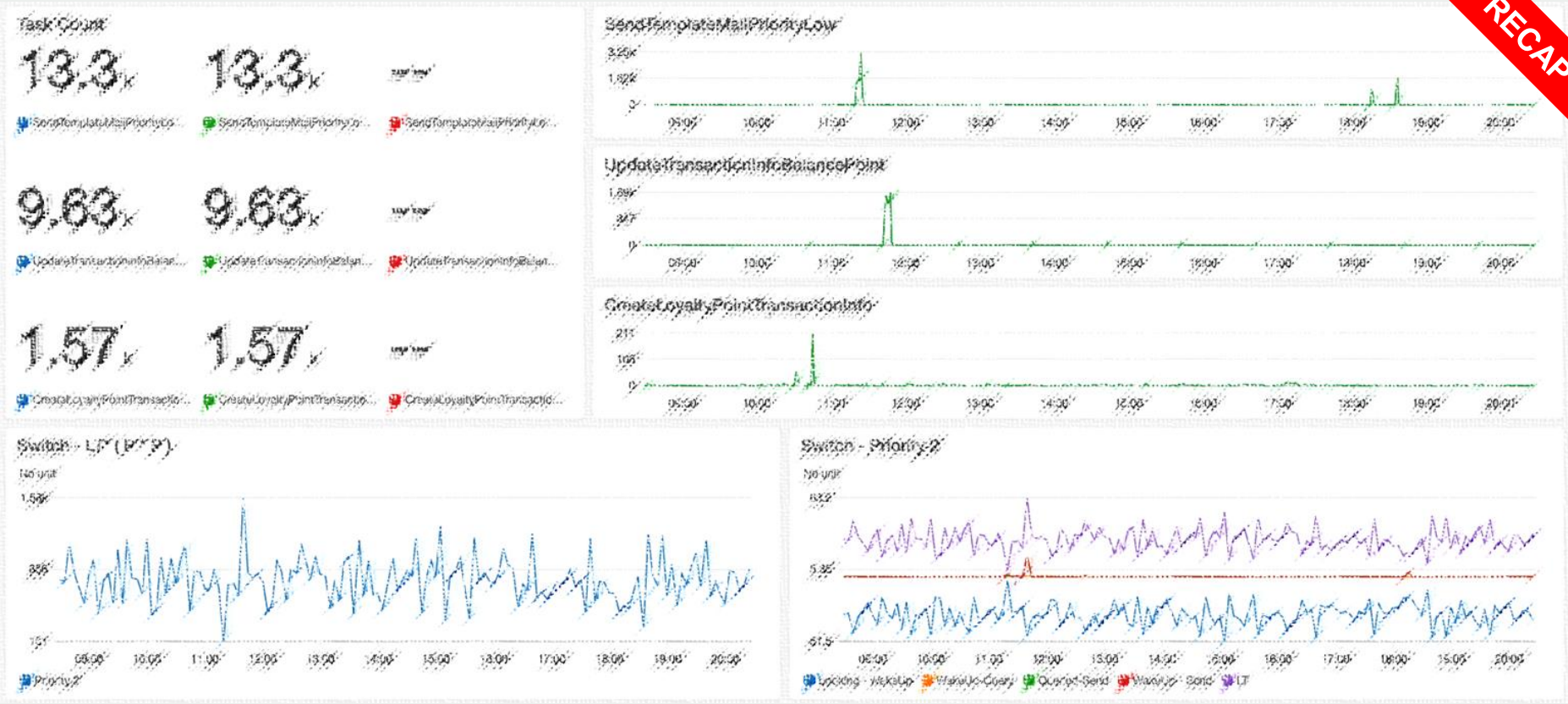
服務水準的概觀: SLA, SLO, SLI ?

RECAP

維運管理重點

- 系統監控
  - 「能被量測的系統，才能被控制」
- 預防型維運管理
  - 設定目標
  - 量測指標
  - 提前改善


RECAP
ELB
Web Server
Cache
Storage
System Monitoring



Application Monitoring / *Business Monitoring

系統監控

RECAP

Event Alert / Notification

RECAP

預防型維運管理

- 決定服務等級目標 - Service-Level Objective (SLO)
  - 99% 前台每秒User訪問延遲 < 300ms
- 測量服務當前狀態 - Service-Level Indicator (SLI)
  - 目前狀況：99% 前台每秒User訪問延遲 < 75ms
- 決定服務等級領先目標
  - 綠燈：99% 前台每秒User訪問延遲 < 150ms
  - 黃燈：99% 前台每秒User訪問延遲介於150ms 到 200ms
  - 紅燈：99% 前台每秒User訪問延遲 > 200ms
- 定期每月、每季Review領先目標
  - 針對黃紅燈項目列出Action Item

SLA ·····▶ SERVICE LEVEL AGREEMENT

the agreement you make with your clients or users

SLOs ·····▶ SERVICE LEVEL OBJECTIVES

the objectives your team must hit to meet that agreement

SLIs ·····▶ SERVICE LEVEL INDICATORS

the real numbers on your performance

https://www.atlassian.com/incident-management/kpis/sla-vs-slo-vs-sli

Case Study

狀況: 帳號註冊的驗證簡訊發送

情境:

會員在註冊帳號的過程中，需要驗證手機號碼。91APP 系統會發出驗證簡訊，使用者收到後輸入驗證碼，即可完成手機號碼驗證。

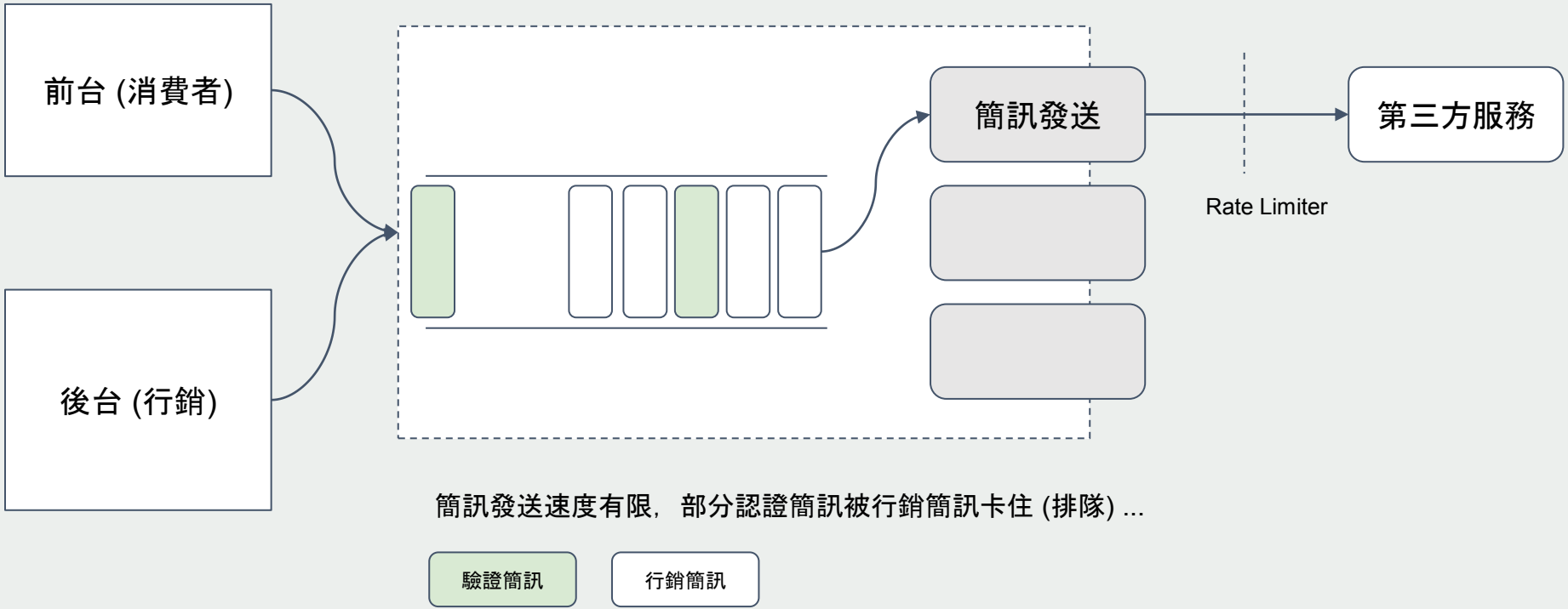
要求:

為了顧及使用者的體驗，**系統必須在 5 sec 內完成發送的作業**。

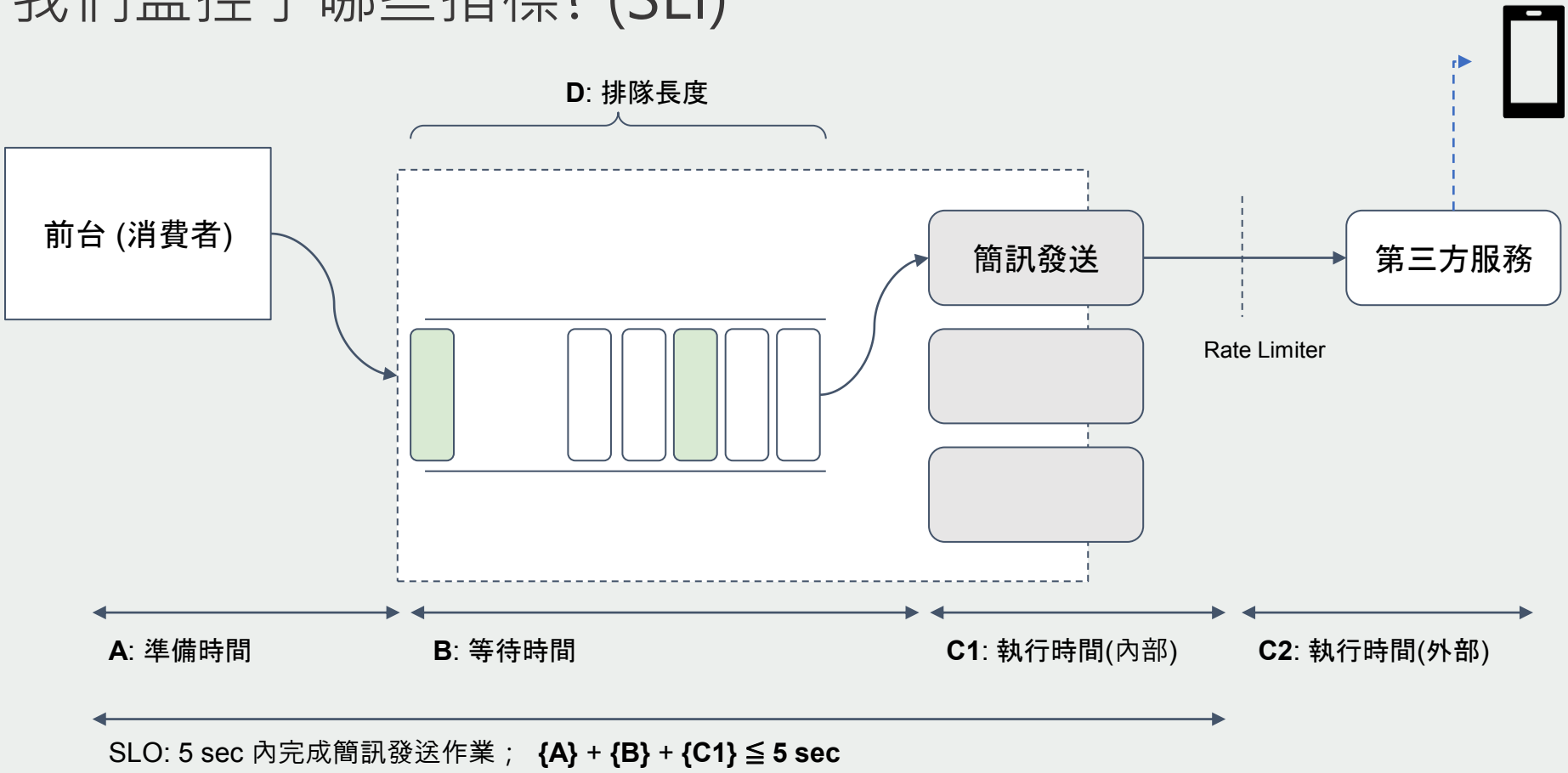
挑戰:

1. 對時間敏感的任務，必須盡量避免被干擾
   (但是過度配置專屬資源會提高成本)
2. 要顧及外部系統的處理能力
   (避免簡訊堆積在外部系統)
   維運團隊需要第一時間掌控狀況

Case: 搶購前會有大量的會員註冊 & 行銷簡訊發送...



我們監控了哪些指標? (SLI)



診斷: 有監控數據 (診) 才能找出效能瓶頸的所在 (斷)

如果:

(A) 的數值過高: 前端系統產生驗證簡訊的速度太慢；

(B) 的數值過高: 訊息在 Queue 裡面排隊花太多時間

Queue 堆積太多行銷簡訊

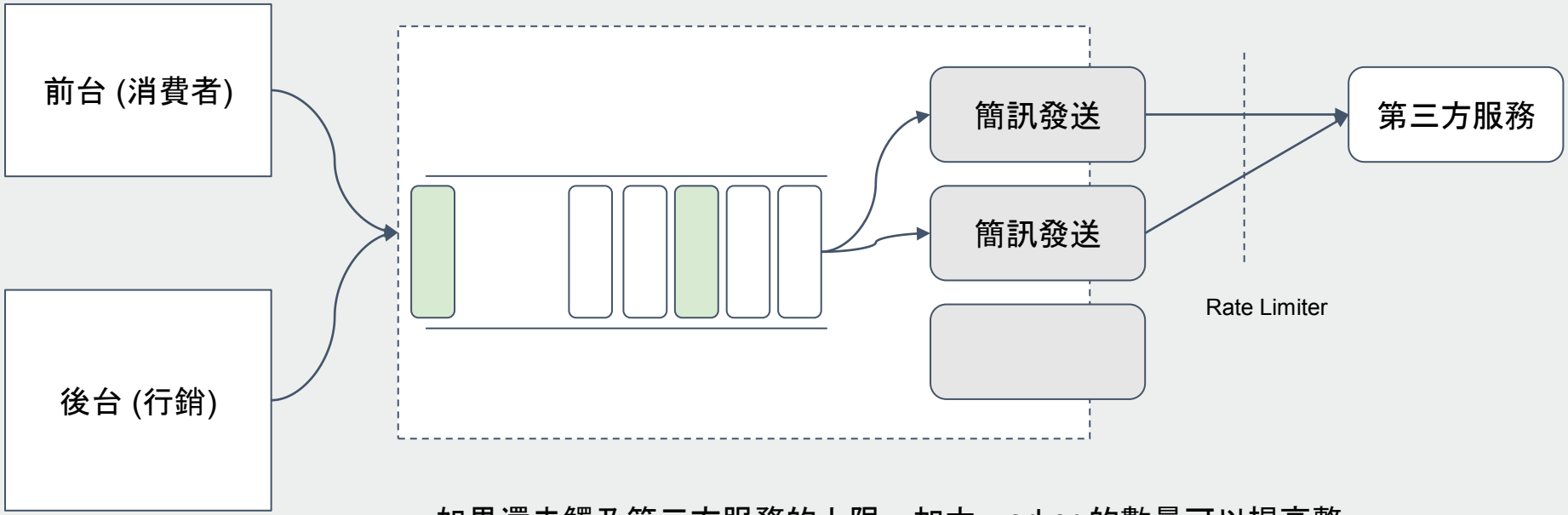
(D) 過高, 訊息堆積太多

(D) 不高, 訊息消化太慢

(C1) 的數值過高: 訊息消化太慢 (查詢資料庫，套用訊息內容等等)

(C2) 的數值過高: 第三方的處理效能太慢

Solution #1, Message Worker Scaleout ...

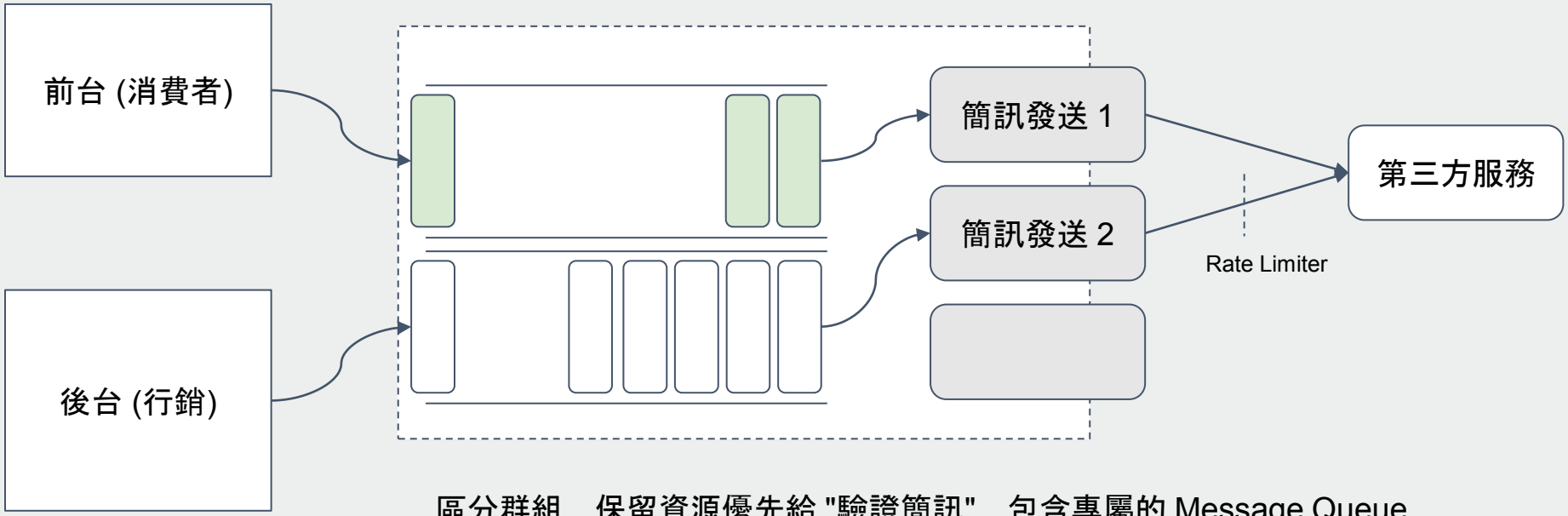


如果還未觸及第三方服務的上限，加大 worker 的數量可以提高整體消化速度。

代價: 耗用兩倍的運算能力

=> 錢沒花在刀口上，因為不需要被加速的行銷簡訊也被加速了...

Solution #2, 降低 Queue Length 的方法 => 分群組



區分群組，保留資源優先給 "驗證簡訊"，包含專屬的 Message Queue, Worker, Rate Limiter …

資源花在刀口上，完全用於加速驗證簡訊的發送。

Q: 我當下怎知該怎麼做?

想辦法擁有“上帝視角”。
把你需要的指標，放到監控系統內

目標導向: 從開發的第一天，就弄清楚你期待的 SLO ...

定義: 消費者按下 "發送驗證簡訊"，**5 秒內** 就要送到手機上

拆解: 這 5 秒內總共要完成哪些事情?

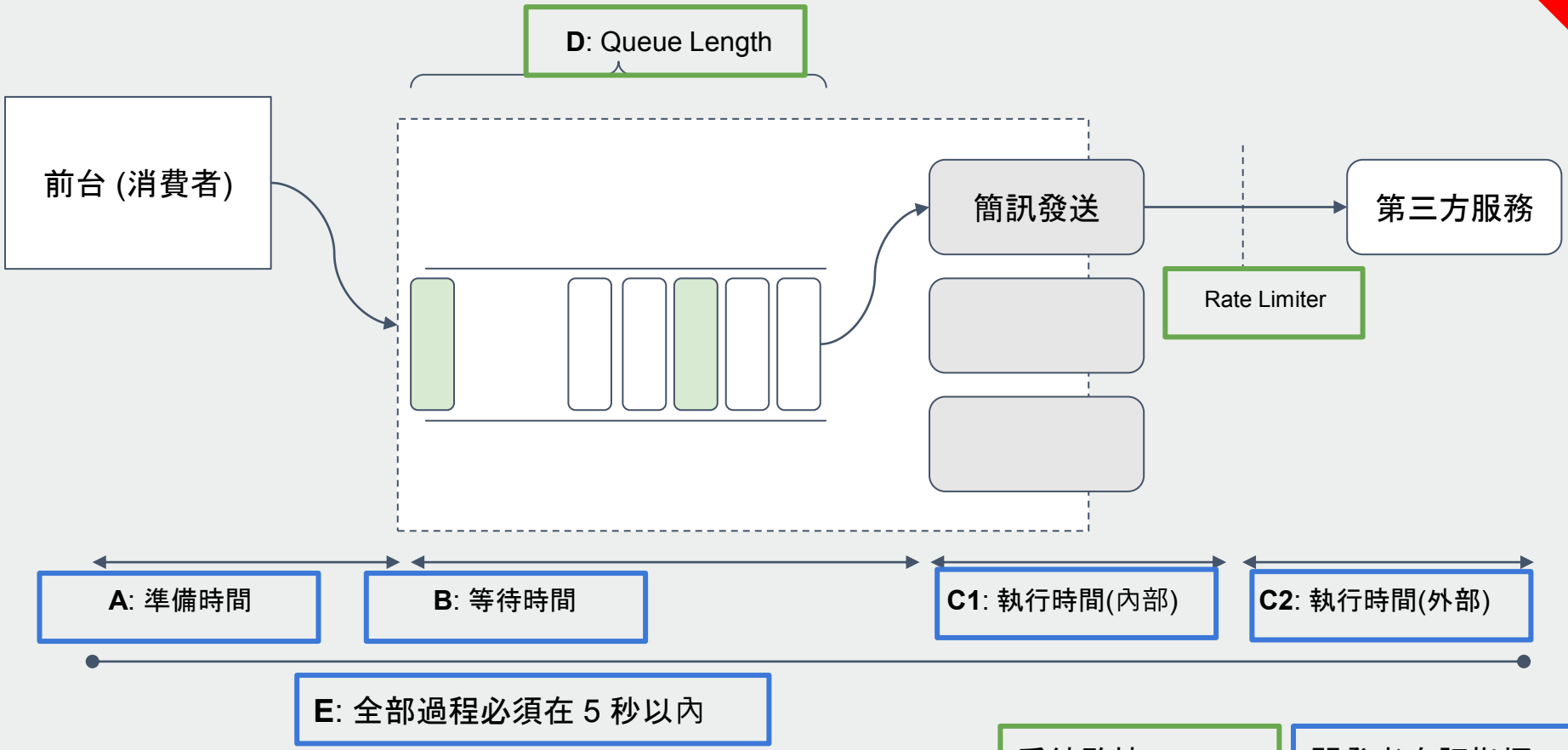
盤點: 我能掌握哪些指標?

改善: 如何處理我無法掌握的指標? (為了維運而設計)

行動: 善用監控的服務，透過 logs 分析，或是 metrics API 來達成

SLI: 我們監控了什麼?

RECAP



Next:

統一收集與管理運用這些指標

RECAP

淺談系統監控
與
AWS CloudWatch 的應用

Rick Hwang
AWS User Group Taiwan
Jun 21, 2017

如果我想要看的指標
CloudWatch 沒有怎麼辦？

RECAP

CloudWatch Custom Metrics

- 兩個常見的需求
  - EC2 Memory Utilization
  - EC2 Disk Utilization
- How
  - AWS CLI / SDK: put-metric-data
  - AWS CloudWatch Logs
  - Third Party Agents:
    - Collected
    - Telegraf

```
aws cloudwatch put-metric-data \
    --metric-name mem \
    --namespace /CWL-Demo/App \
    --unit Percent --value 23 \
    --dimensions InstanceId=1-23456789,InstanceType=t2.small
```

RECAP

監

Watch

Monitor

Observe

Measure

Dashboard

Targets

控

Control

Command

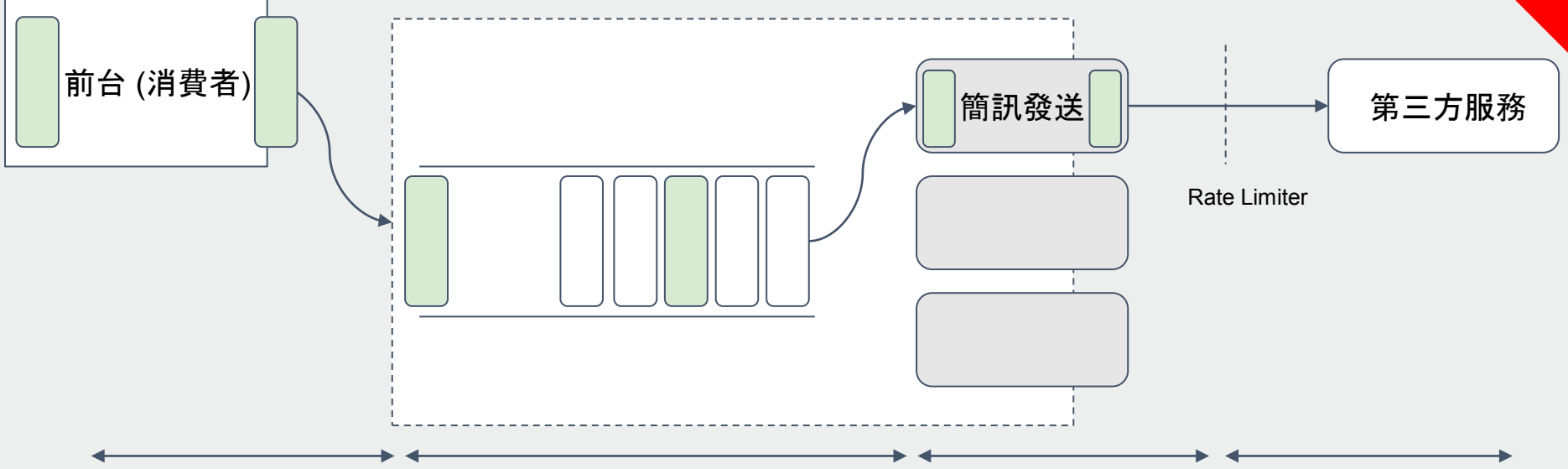
Handle

Manage

Console

怎麼標示與取得這些指標? (for developer)

RECAP



1. 在 message header 上面標示 create time

2. 用 now – create time 就能計算出 {A}

2-1. 將 {A} 放進 buffer, 用非同步的方式透過 SDK 定期清空 buffer 送出數據

3. …
…

4. …
…

Notes: Buffer (in-memory) 是必要的， 即使只有緩衝 1 sec 都能發揮極大的效益。呼叫 cloud provider 的監控數據收集 API 是要按次計費的。
如果不設置 buffer, 那麼費用與 RPS (request per second) 成正比 (變動費用)。若設置 1 sec 的 buffer, 則費用變成與 worker 數量成正比 (固定費用)。

Code Sample (AWS CloudWatch SDK):

Publish metric data points example

```
var client = new AmazonCloudWatchClient();

var dimension = new Dimension
{
  Name = "Desktop Machine Metrics",
  Value = "Virtual Desktop Machine Usage"
};

var metric1 = new MetricDatum
{
  Dimensions = new List<Dimension>(),
  MetricName = "Desktop Machines Online",
  StatisticValues = new StatisticSet(),
  Timestamp = DateTime.Today,
  Unit = StandardUnit.Count,
  Value = 14
};

var metric2 = new MetricDatum
{
  Dimensions = new List<Dimension>(),
  MetricName = "Desktop Machines Offline",
  StatisticValues = new StatisticSet(),
  Timestamp = DateTime.Today,
  Unit = StandardUnit.Count,
  Value = 7
};
```

```
var metric3 = new MetricDatum
{
  Dimensions = new List<Dimension>(),
  MetricName = "Desktop Machines Online",
  StatisticValues = new StatisticSet(),
  Timestamp = DateTime.Today,
  Unit = StandardUnit.Count,
  Value = 12
};

var metric4 = new MetricDatum
{
  Dimensions = new List<Dimension>(),
  MetricName = "Desktop Machines Offline",
  StatisticValues = new StatisticSet(),
  Timestamp = DateTime.Today,
  Unit = StandardUnit.Count,
  Value = 9
};

var request = new PutMetricDataRequest
{
  MetricData = new List<MetricDatum>() { metric1, metric2,
    metric3, metric4 },
  Namespace = "Example.com Custom Metrics"
};

client.PutMetricData(request);
```

https://docs.aws.amazon.com/sdkfornet/v3/apidocs/index.html?page=CloudWatch/MCloudWatchPutMetricDataPutMetricDataRequest.html

Code Sample (Azure Application Insight API)

ApplicationInsights. TelemetryClient. TrackMetric 不是傳送計量的慣用方法。 您應該一律預先彙總一段時間的計量，再加以傳送。 使用其中一個 GetMetric(..) 多載來取得可供存取 SDK 預先彙總功能的計量物件。 如果您要執行自己的預先匯總邏輯，您可以使用 TrackMetric (# A1 方法來傳送產生的匯總。 如果您的應用程式需要每次傳送個別遙測項目 (未隨時間彙總)，則可能有事件遙測的使用案例；請參閱 TelemetryClient.TrackEvent (Microsoft.ApplicationInsights.DataContracts.EventTelemetry)。

Application Insights 可以將未附加至特定事件的計量繪製成圖表。 例如，您可以定期監視佇列長度。 當您使用計量時，個別測量的重要性就不如變化和趨勢，因此統計圖表很有用。

為了將計量傳送至 Application Insights，您可以使用 `TrackMetric(..)` API。 您有兩種方式可以傳送計量：

- 單一值： 每次在應用程式中執行一個測量，都會將對應值傳送至 Application Insights。 例如，假設您有一個描述容器中項目數的計量。 在特定期間內，您先將 3 個項目放入容器中，再移除 2 個項目。 因此，您會呼叫 `TrackMetric` 兩次：先傳遞值 `3`，再傳遞值 `-2`。 Application Insights 會代替您儲存這兩個值。
- 彙總： 使用計量時，每個單一測量並不重要。 相反地，在特定期間內發生的狀況摘要才重要。 這類摘要稱為_彙總_。 在上述範例中，該期間的彙總計量總和為 `1`，而計量值的計數為 `2`。 使用彙總方法時，您只會在每段期間叫用 `TrackMetric` 一次，並傳送彙總值。 這是建議的方法，因為它可以藉由傳送較少資料點至 Application Insights，同時仍收集所有相關資訊，來大幅降低成本和效能負擔。

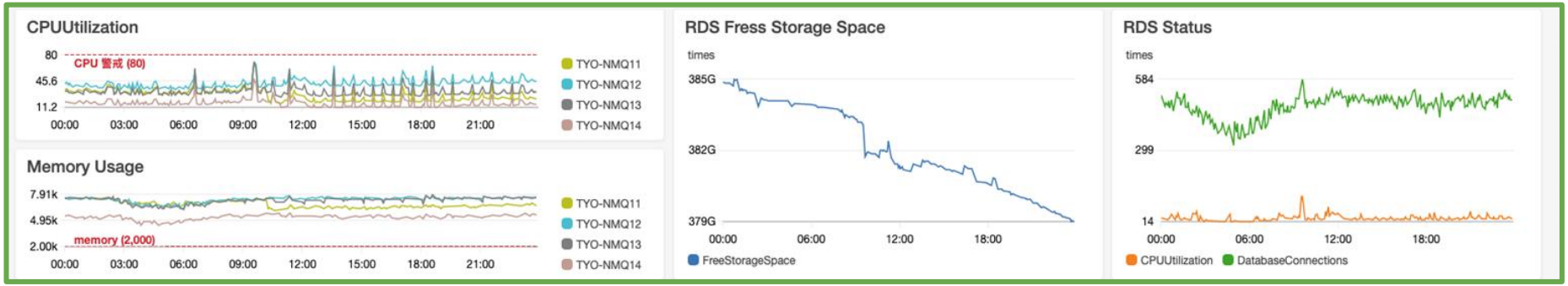
C#

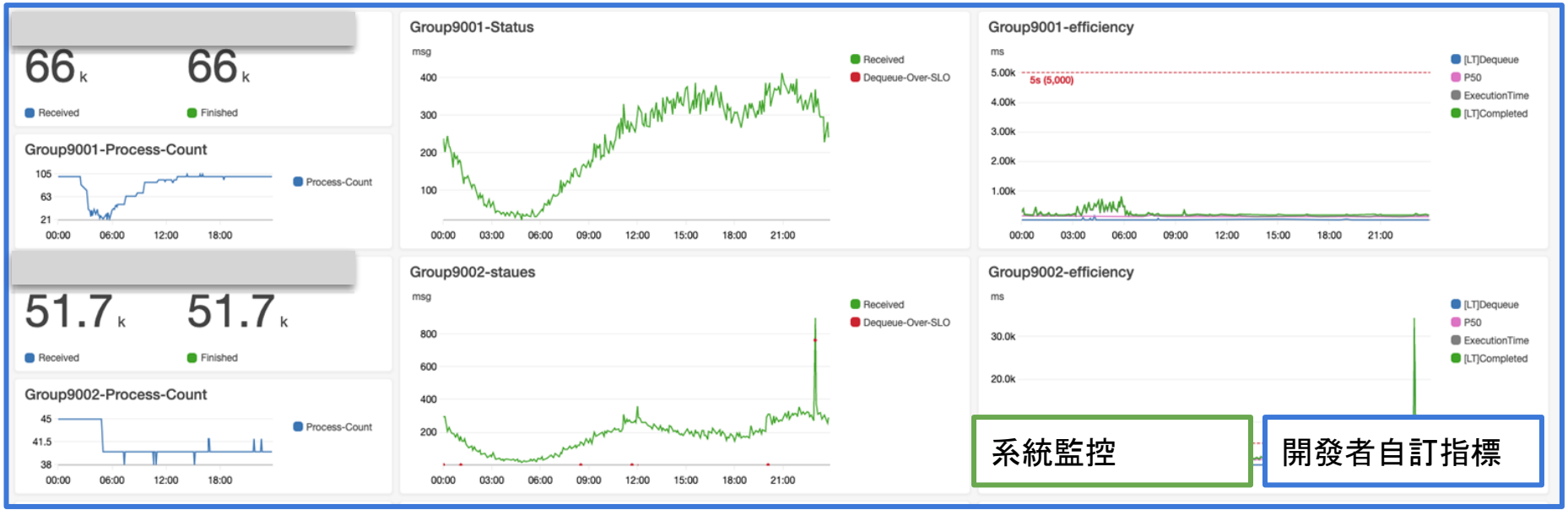
```
var sample = new MetricTelemetry();
sample.Name = "metric name";
sample.Value = 42.3;
telemetryClient.TrackMetric(sample);
```

https://docs.microsoft.com/zh-tw/azure/azure-monitor/app/api-custom-events-metrics#trackmetric

2020 雙十一 監控 dashboard: 我們監控了什麼?

91APP



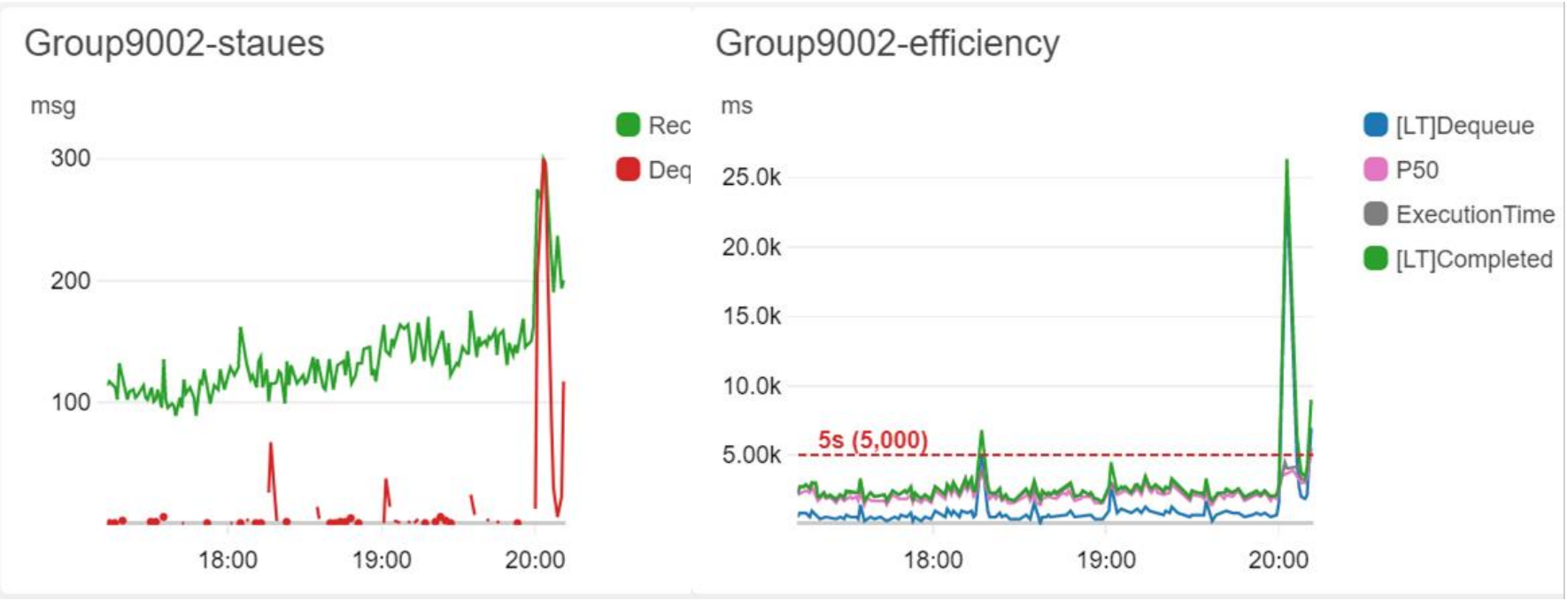


指標: Group Queue Process Status

指標定義:

- Received: Task 從 Message Queue 取出執行的數量統計
- Dequeue-Over-SLO: 所有從 Queue 取出的 Task 中, 取出當下就已經超過 SLO 要求的數量
  (**A** + **B** > **5** sec, 持續 **3** 分鐘狀況沒解除就會發送警告通知)

指標: Group Queue Process Efficiency



指標定義:

- Efficiency: 每個任務從 create task 開始，到 task complete 為止的時間 (**A** + **B** + **C**)

Q: 有了上帝視角之後?

訂定面對各種狀況的應對方式

案例1, 突然有大量的簡訊發送任務, 都超出 SLO 的要求..

RECAP

指標定義:

- Received: Task 從 Message Queue 取出執行的數量統計
- Dequeue-Over-SLO: 所有從 Queue 取出的 Task 中, 取出當下就已經超過 SLO 要求的數量 (**A** + **B** > **5** sec)

碰到這種狀況，你會...?

先列出所有你想的到，[可能] 的解決方式有:

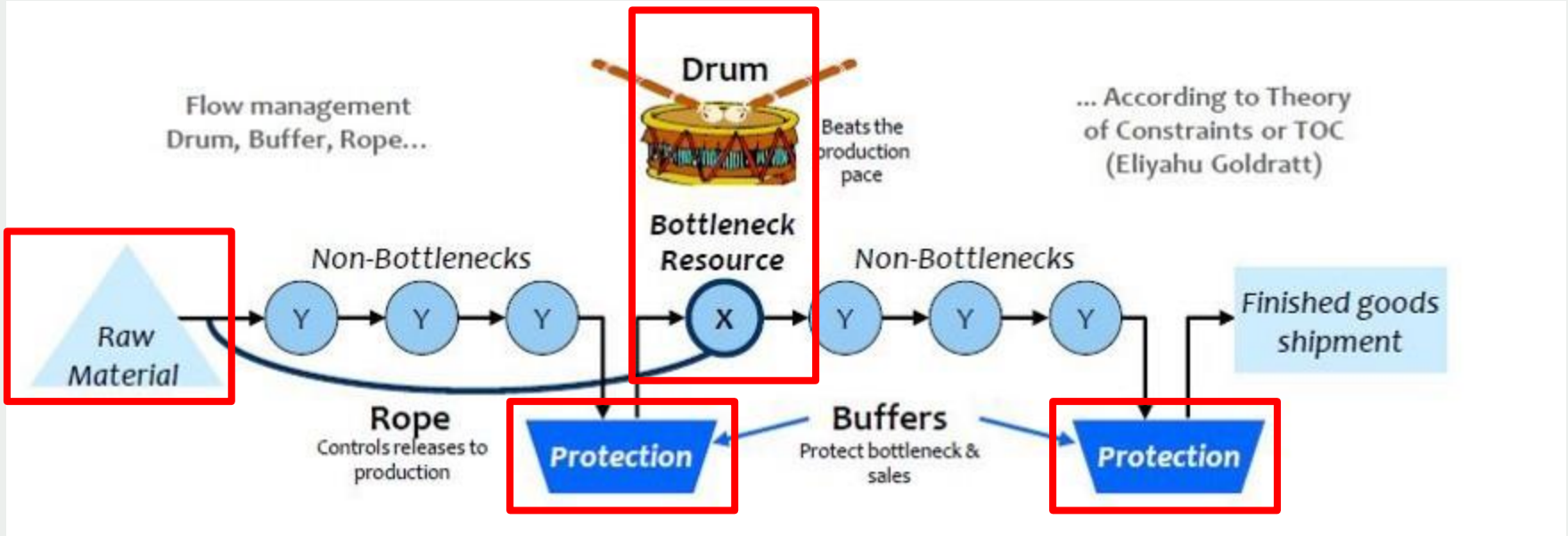
1. 加開 Worker, 增加 instance 個數
2. 改善 Queue 的效率
3. 改善 Task 的效率
4. 改善來源端 (Task Create) 的效率
5. 降低來源端 (Task Create) 的速度 ...

是否有較有系統的思考方式，能精準的找出應對方式?

從限制理論來看 (TOC, Theory Of Constraints)

RECAP

最佳控制點



控制原料，避免生產過快造成堆積。
可由 X 的狀況來控制原料的進貨狀況。

如果偵測到可能超過設計負荷，應該拉動繩子來告訴前端停止原料的採購，避免更大規模的損耗。

確保 X 能夠不中斷的順利進行
必須隨時確保 Buffers **沒被清空**。
不過過多的庫存也是浪費。

確保 X 能夠不中斷的順利進行
必須隨時確保 Buffers **不會滿載**。

『高效率團隊』如何運用限制理論 (Theory of Constraints) 於軟體開發
[EMBA 雜誌] 我該如何解決問題的瓶頸

解決瓶頸的步驟:

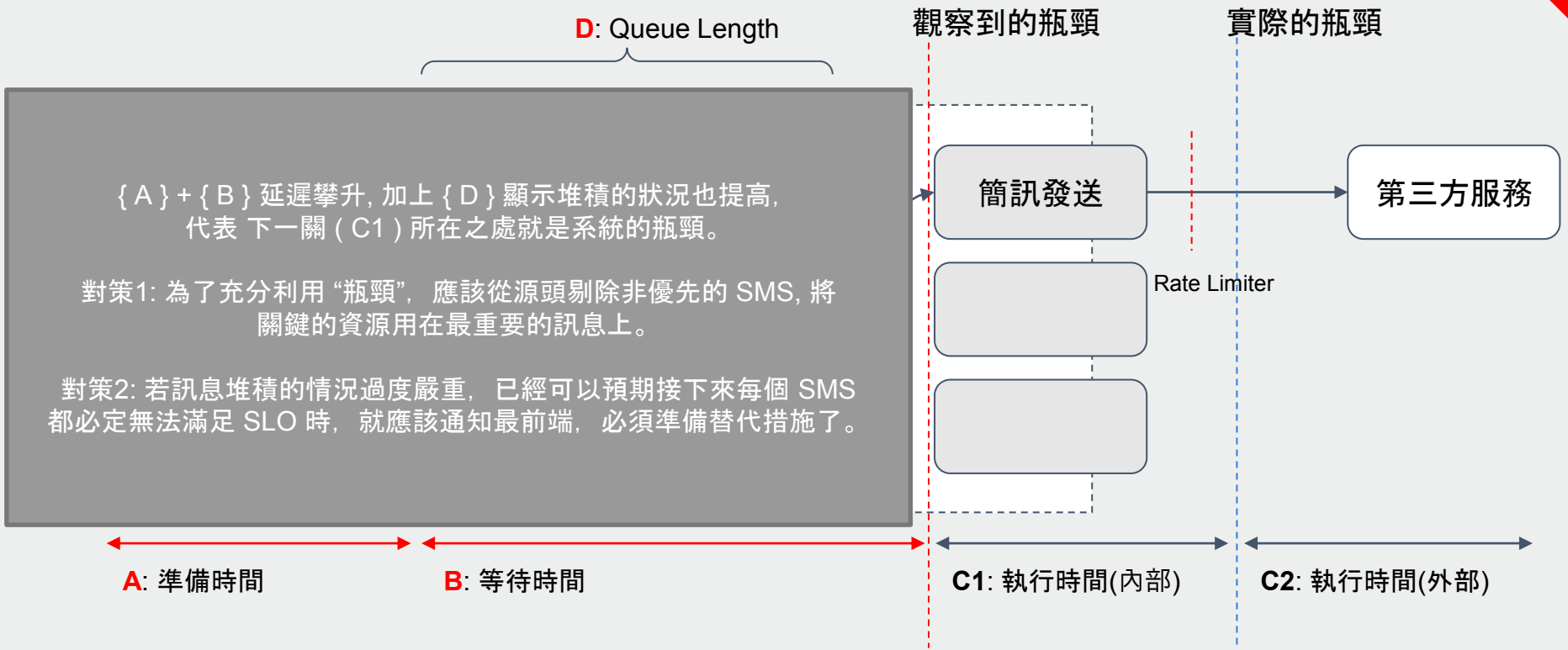
1. **找到你的瓶頸**； (特徵: 瓶頸的前一關通常都會出現庫存堆積的狀態)
2. **充分利用瓶頸**； (既然是瓶頸，就別讓他停下來，盡可能讓他維持100% 的產能)
3. 非瓶頸協助瓶頸；
4. **提升打破瓶頸**；
5. 回到步驟一（找出下一個瓶頸），周而復始。

延伸思考: 生產線的 [瓶頸]，就是整體效能的控制點。

1. **改善**瓶頸效能，就能改善整體效能
2. **控制**好 [瓶頸] 的速度，就能控制整條產線的速度
3. **若無法改善 [瓶頸]，則要有能力在生**產線失控 **(過多 WIP) 前從源頭停止。**

先從數據指標，還原實際的狀況

RECAP



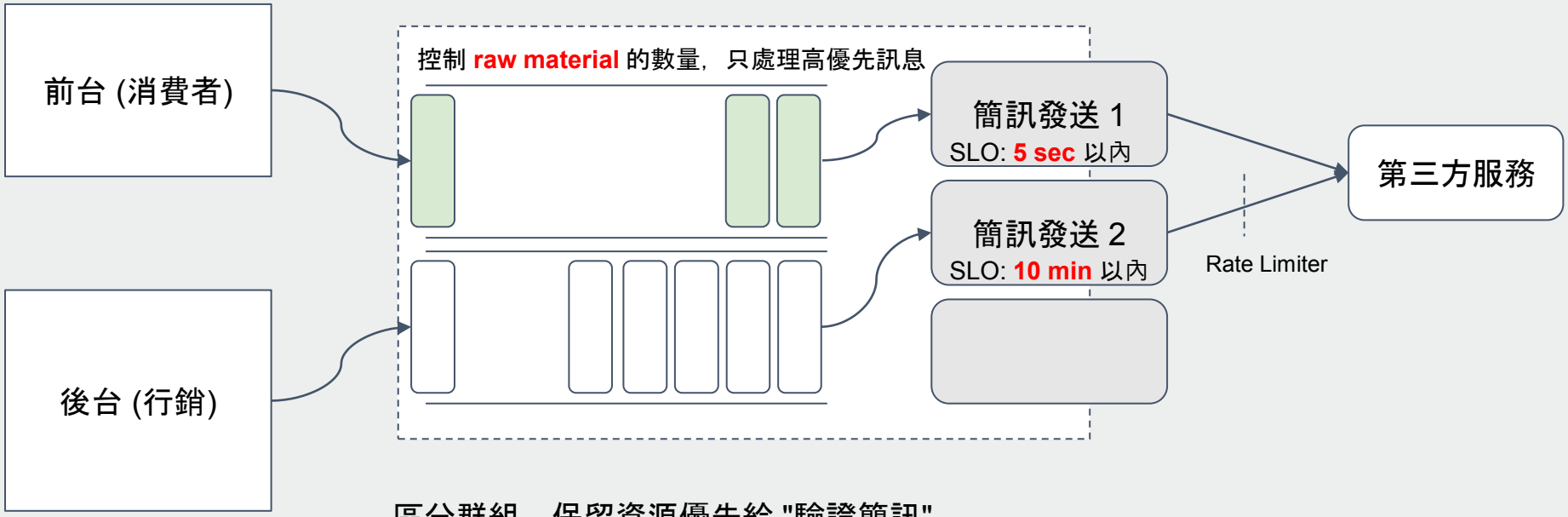
能夠選擇的做法

1. 控制來源: 前端切換其他方式，進行手機號碼驗證 (例如: 改用撥號驗證)
2. 控制來源: 排除非關鍵的任務進入這條生產線 (例如: 行銷簡訊)
3. 降低 SLO 的要求 (非關鍵通知不需要 5 sec 送達)
4. 擴大 91APP 與簡訊商的安全容量

   (與第三方廠商確認後，放寬 Rate Limit 限制)
5. 擴充 Worker 的處理能力
6. 改寫 Task, 做好最佳化改善執行速度

Solution #2, 降低 Queue Length 的方法 => 分群組

RECAP



區分群組，保留資源優先給 "驗證簡訊"，
包含專屬的 Message Queue, Worker, Rate Limiter …

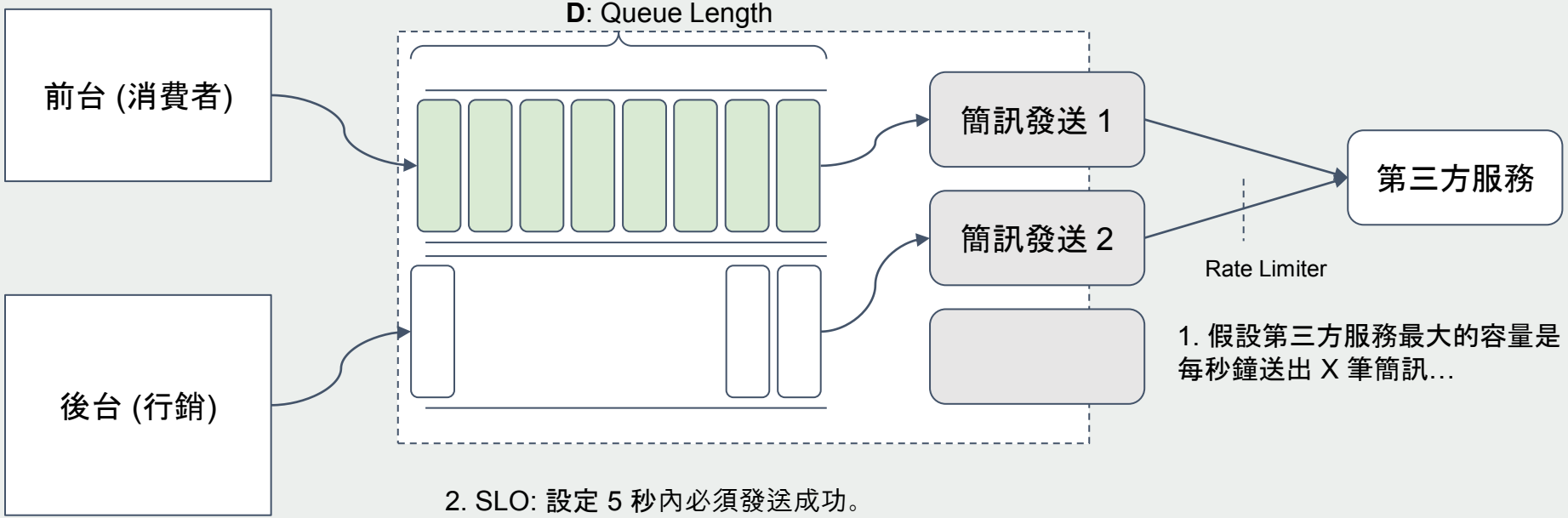
資源花在刀口上，完全用於加速驗證簡訊的發送。

Case: 如果長時間無法滿足SLO?

假想狀況: 服務量滿載，驗證簡訊註定無法如期發送..

RECAP

3. 只要 Queue Length 超過 30X 筆堆積的訊息，即使
第三方服務全速運轉，新加入的訊息註定無法在 30 秒內送出…



1. 假設第三方服務最大的容量是
每秒鐘送出 X 筆簡訊…

2. SLO: 設定 5 秒內必須發送成功。

(追加: 若超過 30 秒，則該簡訊直接失效。
送出的驗證碼只在 5 分鐘內有效，延遲過久
的簡訊就算成功送出，也沒有意義了)

Think: 註定無法在期限內送出的訊息，還需要發送嗎?

Q: 如何從數據判讀這種狀況?

Q: 發生當下，前台系統能知道嗎?

Q: 當下該如何處理?

Q: 事後改善該如何處理?

Q: 如何從數據判讀這種狀況?

{ Queue Length } 超過上限，同時 **{ 發送速率 }** > 處理的速率 **{ Rate Limit }** 。

Q: 發生當下，前台系統能知道嗎?

Design For Operation； 無法得知。除非你在開發的時候就有考慮到這個問題，或是搭配監控系統，偵測到這種狀況時自動調整系統 configuration / feature toggle, 來告訴系統實際的狀況。

Q: 當下該如何處理?

需要靠“繩子”做為回饋機制 (生產者 / 消費者問題)。例如:

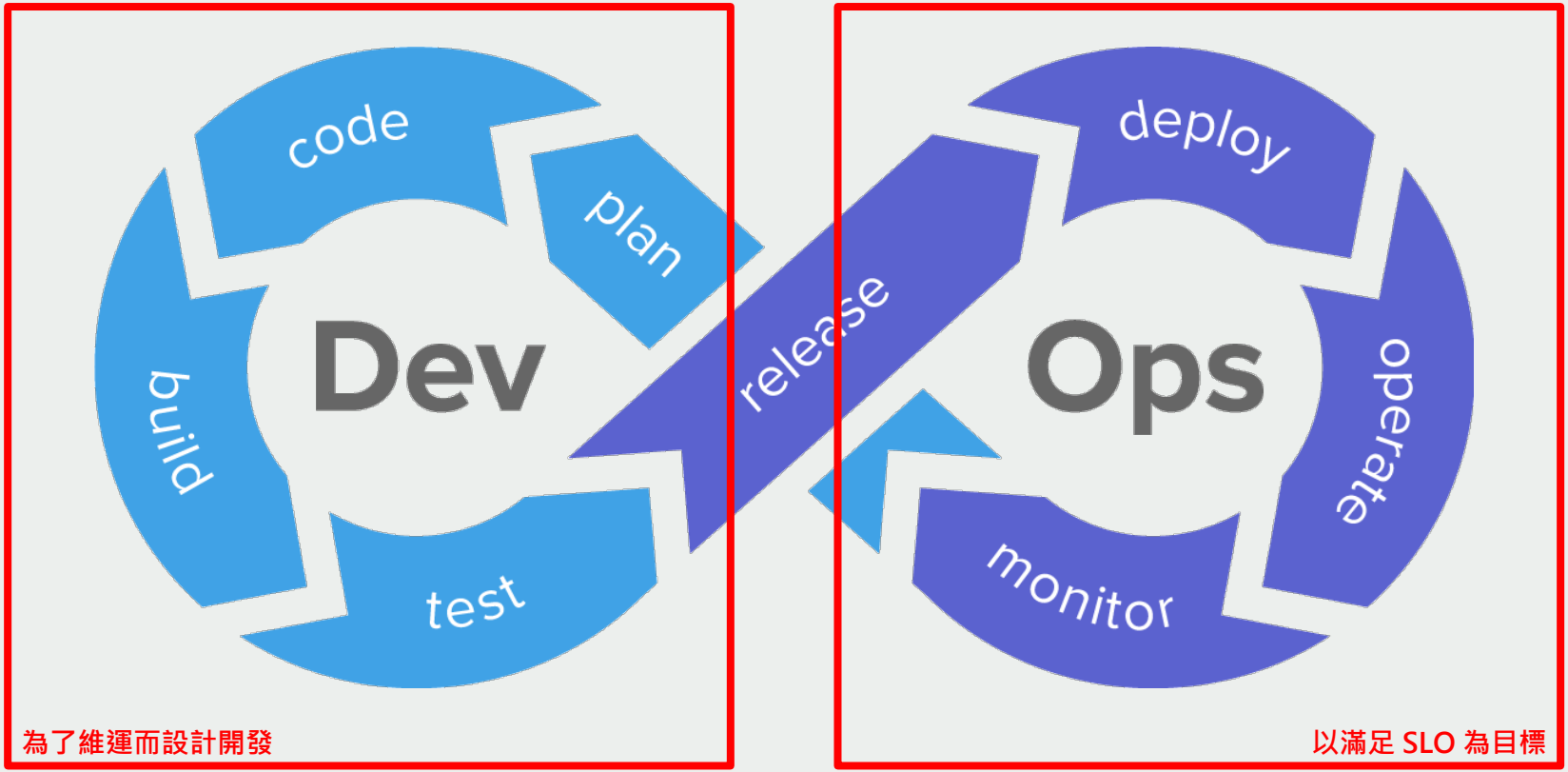
1. 若狀況發生，停用前台發送驗證簡訊的功能 (配合熔斷機制、健康偵測機制)。
2. 提供替代的方案，例如播號驗證 (提供 routing)。
3. 啟用備援方案，例如備援用的第二家簡訊商。

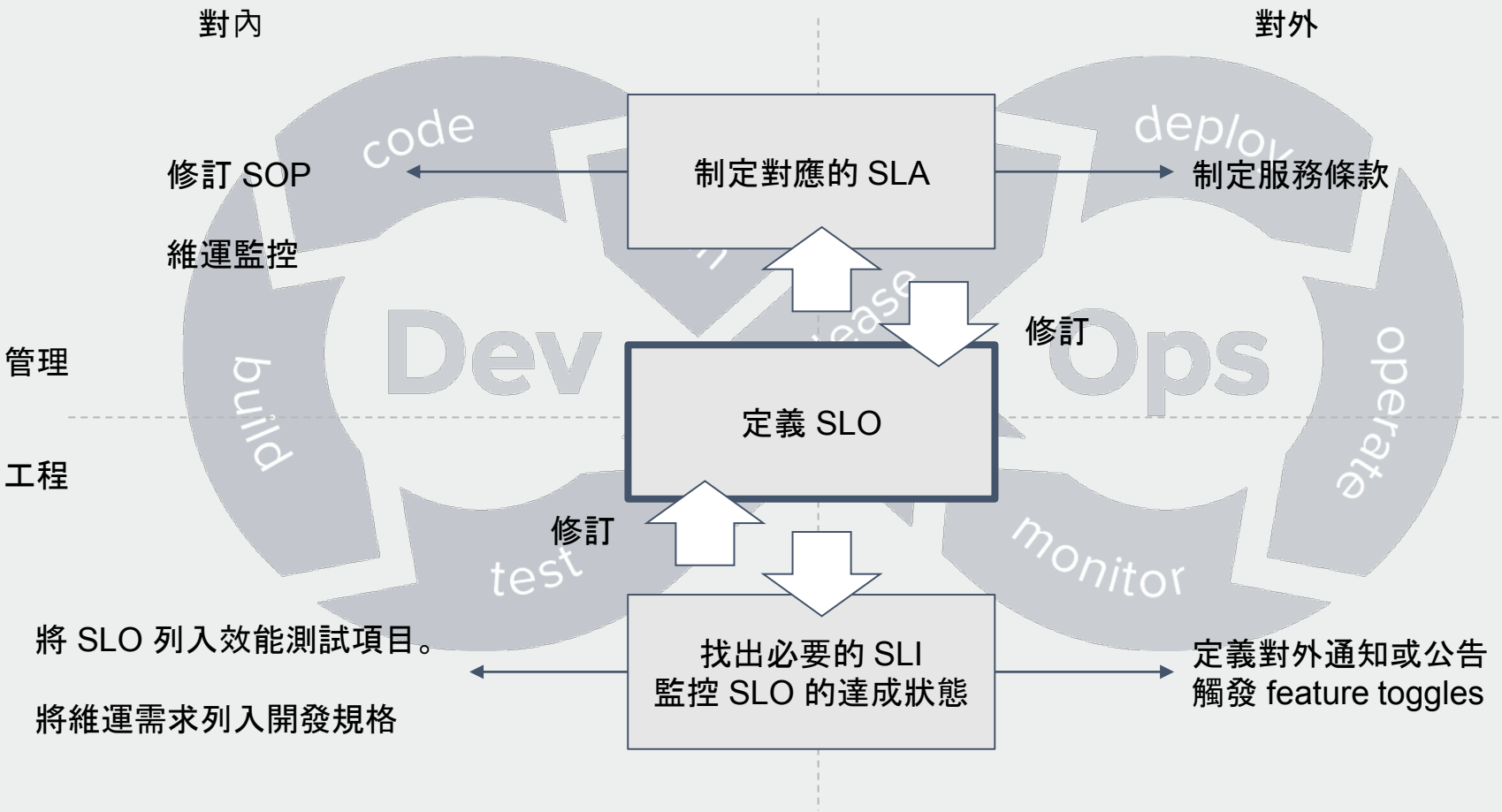
Q: 事後改善該如何處理?

對外: 針對瓶頸改善。擴大 91APP / 第三方 之間的乘載量上限

對內: 從產品設計與開發，就建立回饋機制，進行自動化控制
(例如: 自動啟用備援服務，自動啟用熔斷機制等等)

落實 DevOps 的精神


code
plan
build
Dev
test
release
deploy
operate
Ops
monitor
為了維運而設計開發
以滿足 SLO 為目標


對內
對外
修訂 SOP
維運監控
制定對應的 SLA
制定服務條款
code
deploy
Dev
Ops
build
operate
管理
修訂
定義 SLO
工程
修訂
test
monitor
將 SLO 列入效能測試項目。
將維運需求列入開發規格
找出必要的 SLI
監控 SLO 的達成狀態
定義對外通知或公告
觸發 feature toggles

Thanks for your listening.

謝謝你的聆聽。