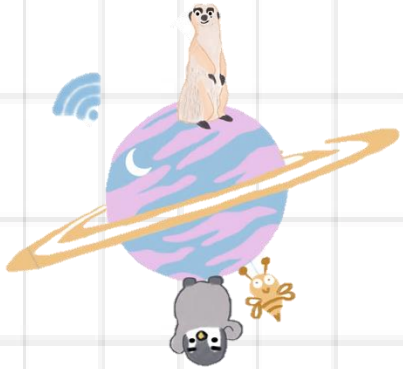
COSCUP

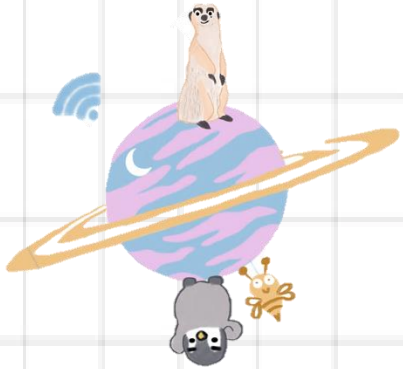


使用 Golang 開發 Linux Scheduler !

Ian / 陳毅

@ianchen0119

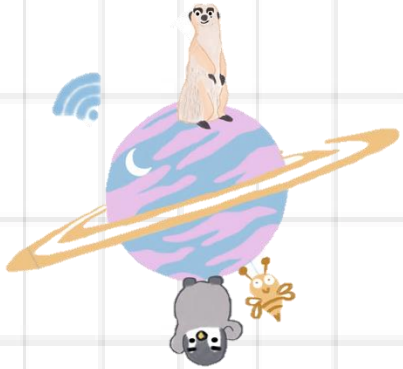
關於我



- 軟體工程師 at 禾薪科技
- 兼任講師 at 國立陽明交通大學
- 2 x 微軟 MVP
- TSC Member[at] free5GC
- 著有兩本資訊書籍
  - EN 帶你寫個作業系統
  - EN 帶你入門 5G 核心網路

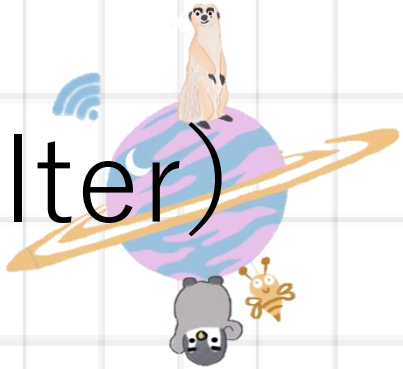
@ianchen0119

大綱

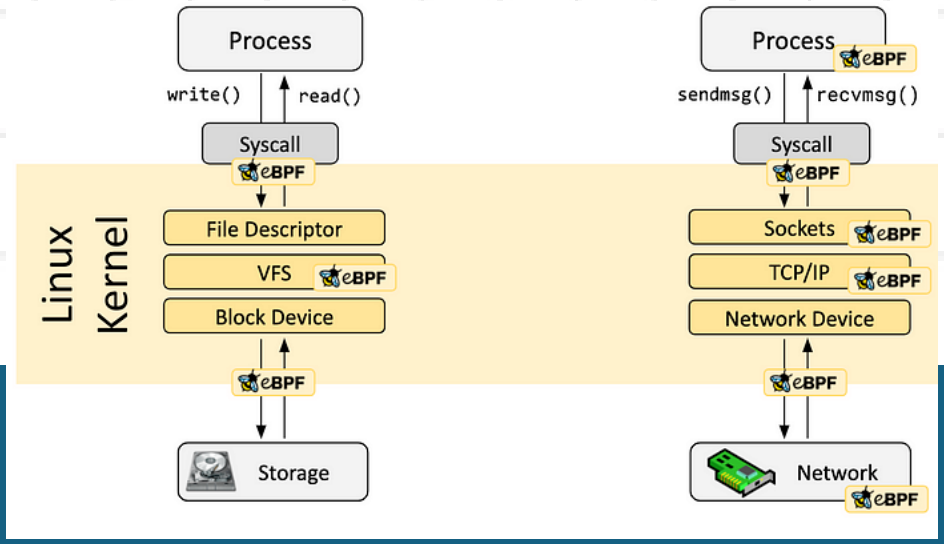


- Introduce to eBPF
  - Background
  - Toolchain
  - struct_ops & sched_ext
- The scx_rustland
  - Architecture
  - Design
- The scx_goland
- How to develop your own scheduler with scx_goland_core
- Future Work
- Q&A

eBPF（extended Berkeley Packet Filter）

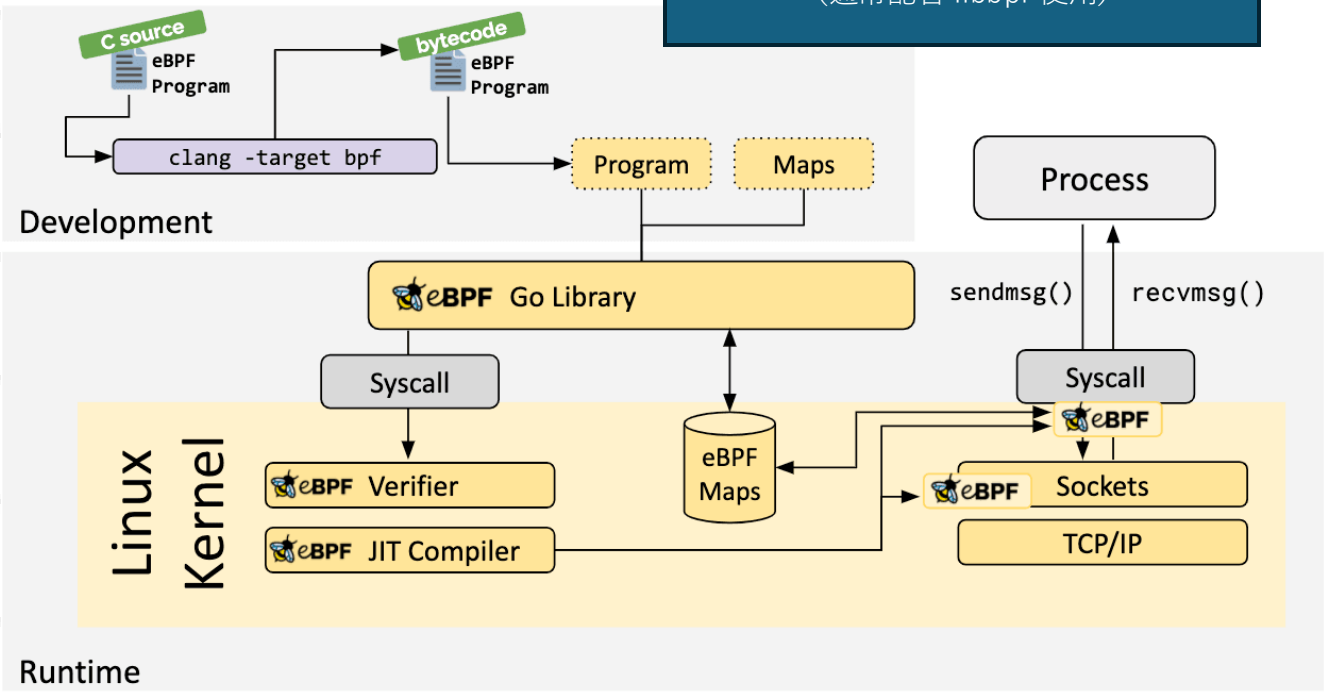


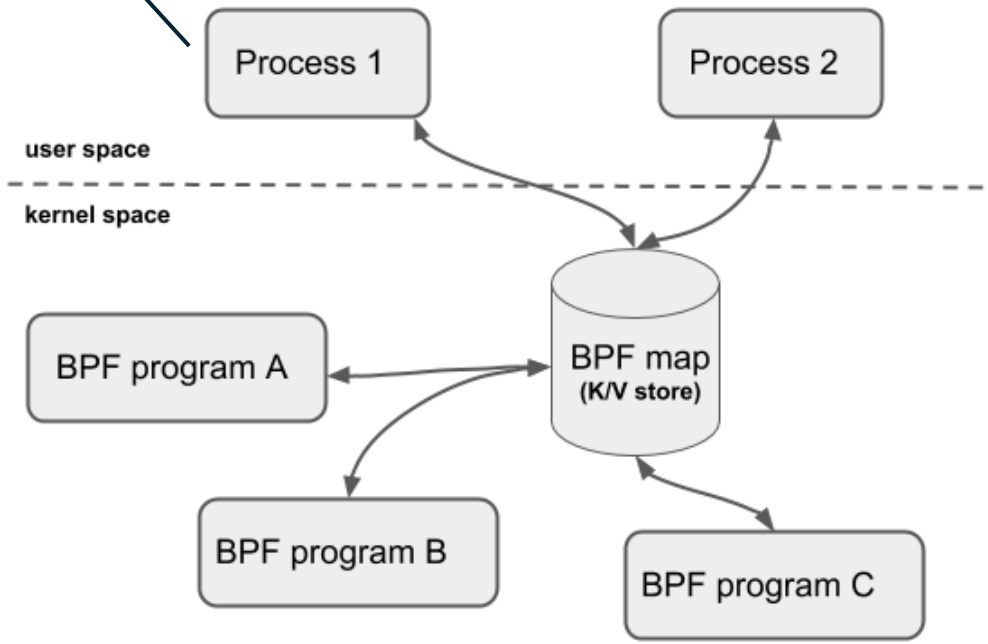
- 其實跟 BPF 沒有直接關係
- 最早是由 PLUMgrid 開發的 SDN kernel module（iovisor.ko）
- 嘗試貢獻到上游，為了被接受而發展成 eBPF
- eBPF program 可被注入到系統的各個層級，探測、甚至影響各種系統行為



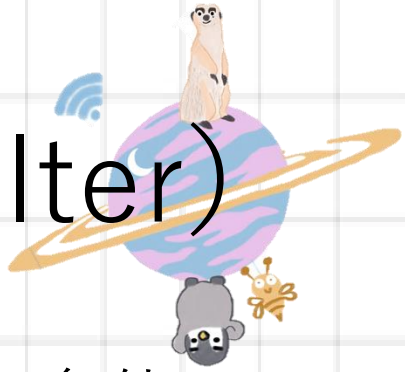
eBPF (extended Berkeley Packet Filter)

User space process 負責管理 eBPF Program 與 eBPF Maps，它可以是 bpftool、bcc 或是使用者自行開發的程式（通常配合 libbpf 使用）



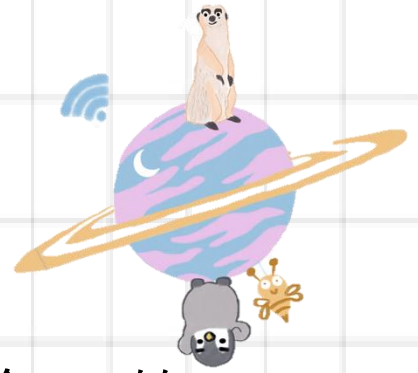


eBPF（extended Berkeley Packet Filter）

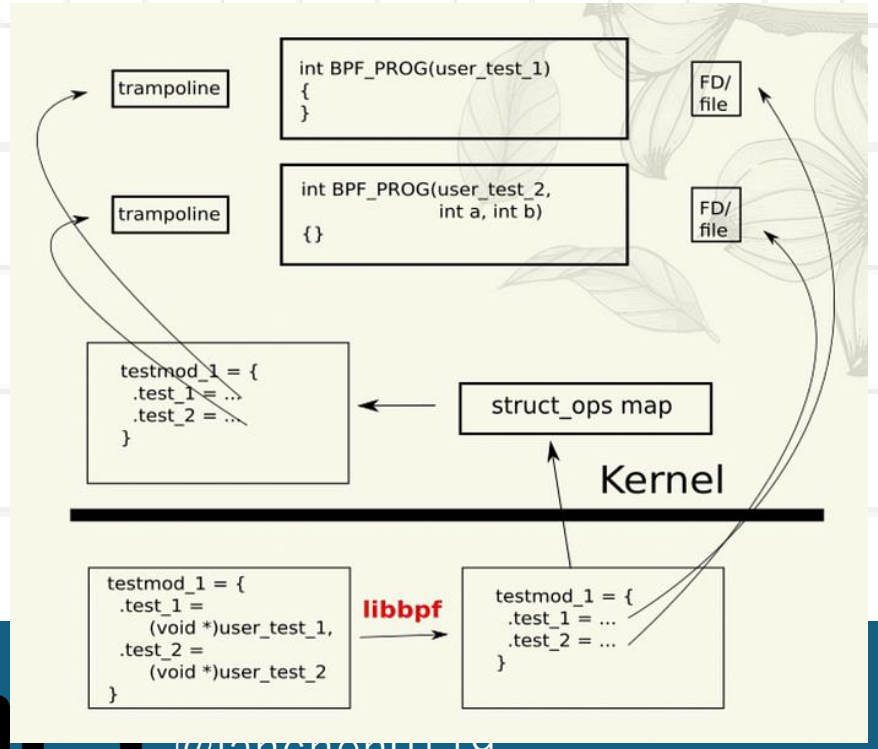


- 隨著越多科技巨頭投入 eBPF 生態圈，eBPF 也被賦予了更多能力：
  - 追蹤：觀察 kernel functions 的呼叫、cGroup 追蹤
  - 網路：traffic control、XDP、netfilter
  - 安全：Linux Security Module
- 再後來，Meta 為了用 eBPF 開發客製化的 tcp congestion control 新增了 struct_ops 類型的 eBPF program

struct ops

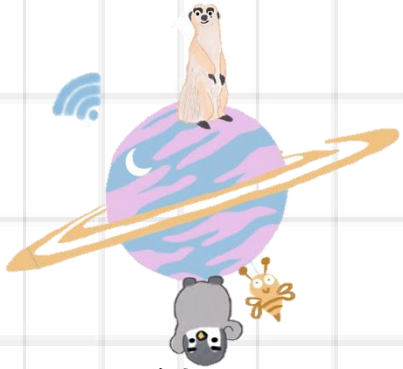


struct_ops 本質上是一個 eBPF map，記載實作了 kernel 介面的 eBPF program(s)：



Ref:
https://lpc.events/event/17/contributions/1607/attachments/1164/2407/lpc-struct_ops.pdf

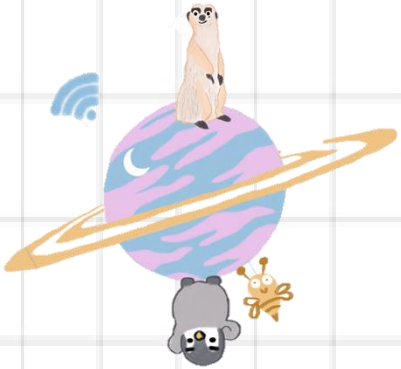
Scheduler extension (sched_ext)



Linux Kernel 自 v6.12 開始正式支援 sched_ext，開發者只需要使用相關 API + 利用 struct ops，就能夠用 eBPF 開發客製化的排程器！

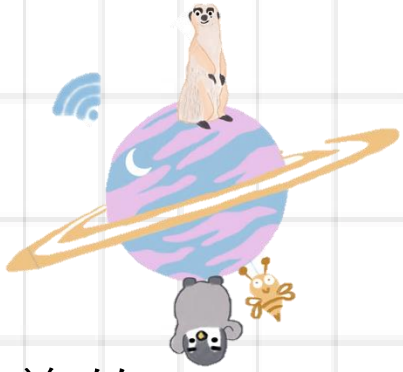
```
SEC(".struct_ops")
struct sched_ext_ops simple_ops = {
        .select_cpu             = (void *)simple_select_cpu,
        .enqueue                = (void *)simple_enqueue,
        .init                   = (void *)simple_init,
        .exit                   = (void *)simple_exit,
        .name                   = "simple",
};
```

Scheduler extension (sched_ext)



- 每個 CPU 都有屬於自己的 DSQ（Dispatch Queue）
- 系統會有一個預設的 global DSQ
- 開發者可用 API 新增更多 DSQ
- Details: https://medium.com/@ianchen0119/ebpf-%E9%9A%A8%E7%AD%86-%E4%B8%83-sched-ext-f7b60ea28976
- Example: https://github.com/sched-ext/scx/blob/main/scheds/c/scx_simple.bpf.c

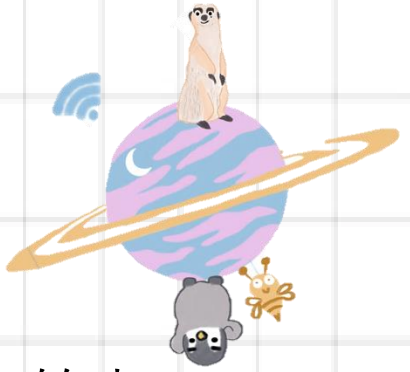
Scheduler extension (sched_ext)



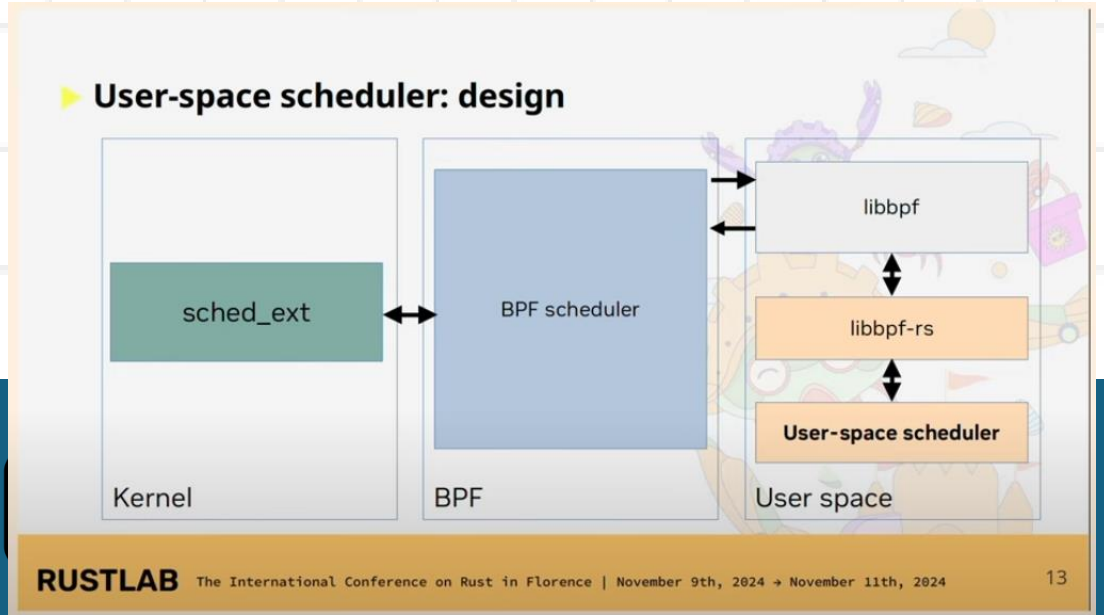
言歸正傳，sched_ext 就是 Linux kernel 為排程器子系統定義的擴充介面，讓我們能用 struct_ops 的方式開發客製化的排程器。

所以…用 eBPF 開發排程器跟 gopher 的關聯是…？

scx_rustland

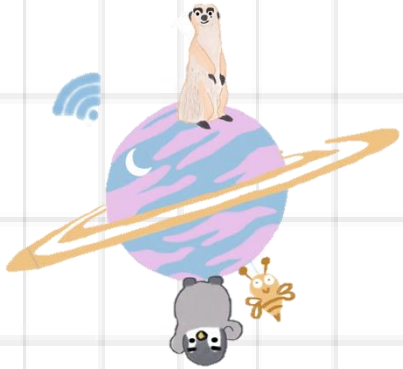


任職於 NVIDIA 的工程師 Andrea Righi 提出了一個非常天才的想法：把需要排程的 processes 透過 eBPF Map 的機制丟給 user space process 讓它決定排程後，再將決定透過 MAP 丟給 eBPF program。這個想法也就變成了 scx 專案下的 rustland 排程器（user space program 用 rust 開發）。



Ref: https://www.youtube.com/watch?v=L-39aeUQdS8

scx_rustland

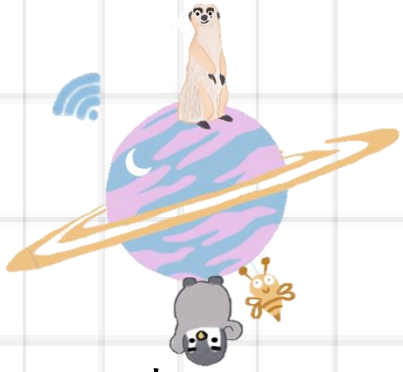


scx_rustland 的核心設計大致如下：

- 綁定特定 CPU 的 kernel thread 由 eBPF program 直接排程
- User space program 本身由 eBPF program 直接排程
- User space program 為給定任務選擇 CPU 時需考慮 CPU topology (cache)
- User space program 為給定任務設定 deadline 時會考慮 voluntary context switches
- User space program 為給定任務分配可變的 time slice（這取決於當下有多少任務在排隊）

Ref: https://www.youtube.com/watch?v=L-39aeUQdS8

scx_goland

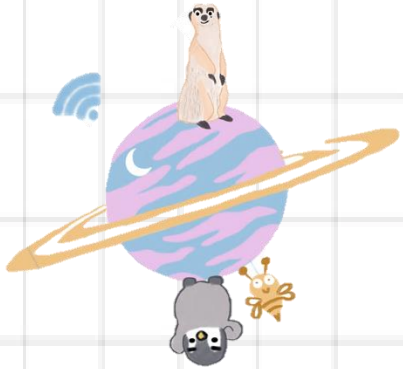


看完大神演講的我想著「golang 完全也可以做到 scx_rustland 中 rust 所扮演的角色吧」，畢竟 go 可是有…

- Cgo
- Libbpf-go：由 aqua security 封裝的 golang 版 libbpf
- 聰明的記憶體、線程管理機制

Ref: https://www.youtube.com/watch?v=L-39aeUQdS8

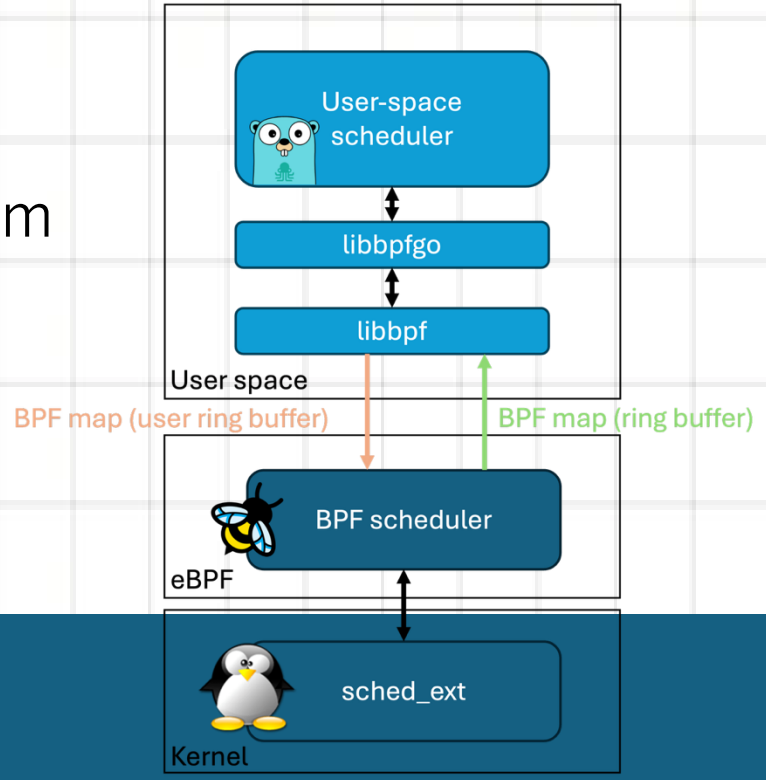
scx_goland



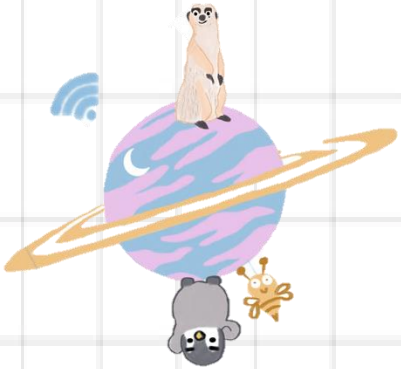
初步構想：

- User space program 呼叫 libbpf-go 提供的 API 載入 eBPF program
- 透過 ring buffer 從 eBPF program 取得待排程的任務
- 實作基於 voluntary context switches 的 EDF 排程器
- 將 decision 透過 user ring buffer 傳遞至 eBPF program

大功告成！



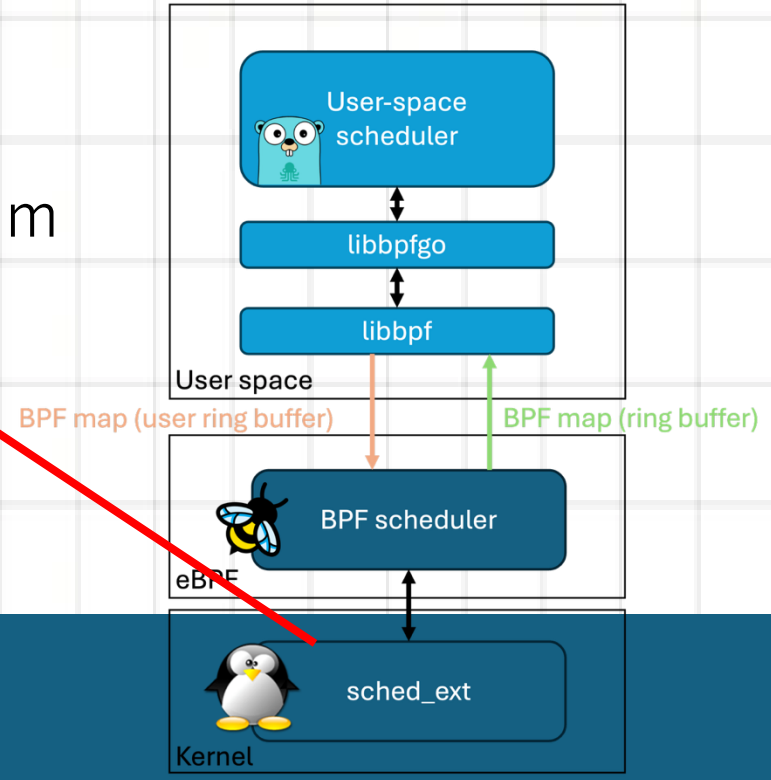
scx_goland



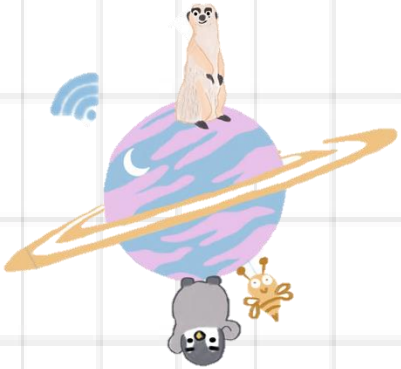
初步構想：

- User space program 呼叫 libbpf-go 提供的 API 載入 eBPF program
- 透過 ring buffer 從 eBPF program 取得待排程的任務
- 實作基於 voluntary context switches 的 EDF 排程器
- 將 decision 透過 user ring buffer 傳遞至 eBPF program

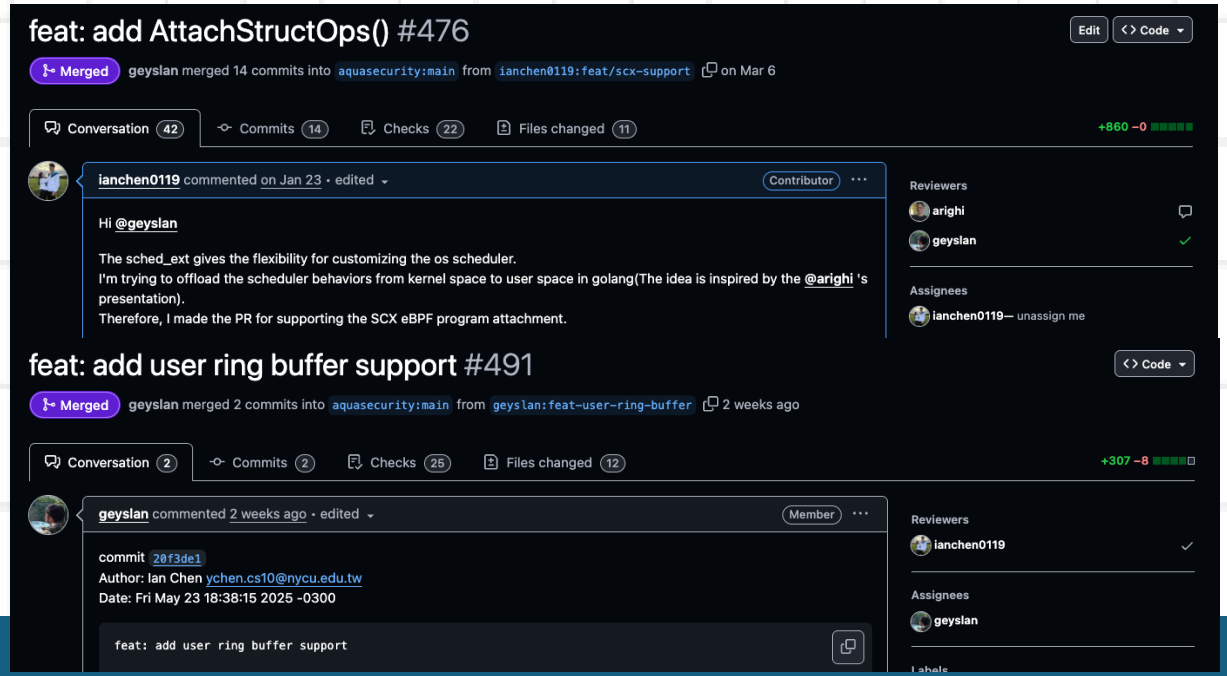
大功告成！



scx_goland

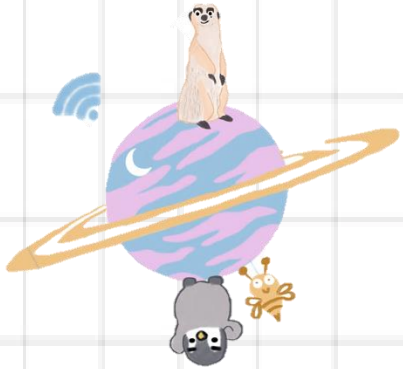


- Libbpf-go 不支援 user-ring buffer map 相關操作
- Libbpf-go 不支援 struct-ops 相關操作



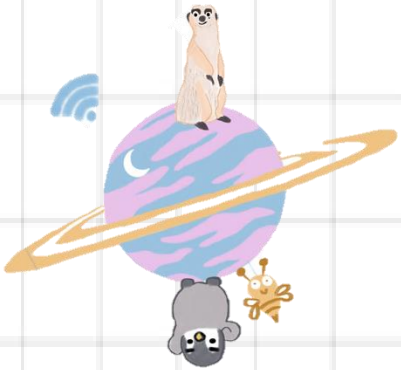
@ianchen0119

scx_goland

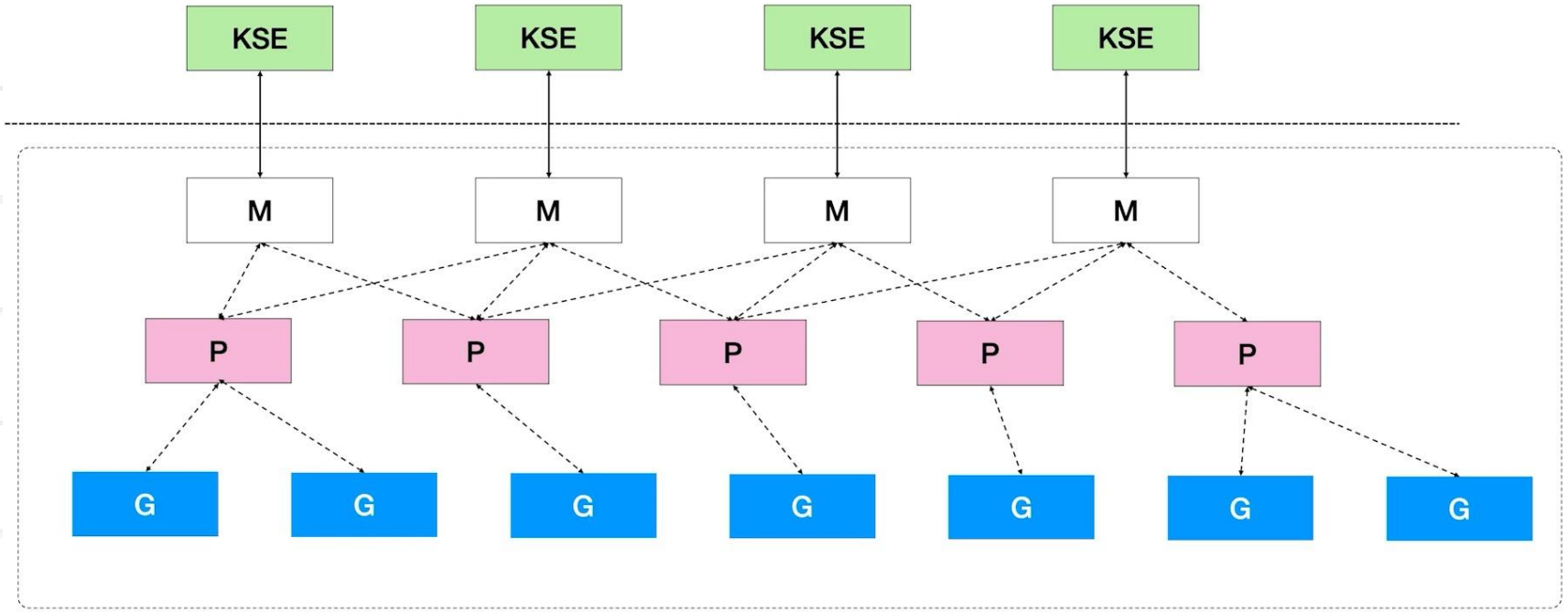


解決了 toolchain 的問題後，終於成功載入 eBPF scheduler！但是…
載入後系統卡住 5 秒後 scheduler 直接被踢掉..🤔

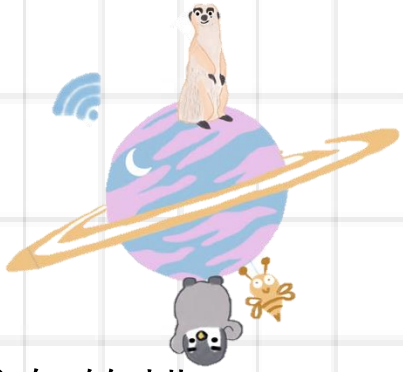
scx_goland



經過了幾晚的思考，終於找到問題所在：



scx_goland_core



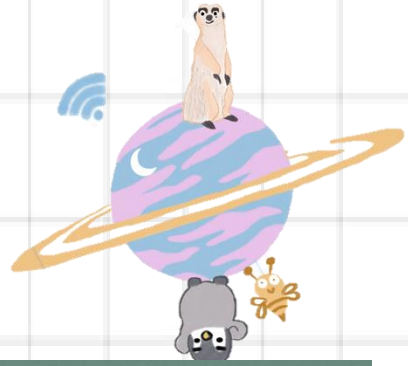
scx_goland_core 能夠接受來自 Kernel Space 的排程事件。因此，所有的排程器邏輯都能用 golang 實作：

```
func GetNrQueued() uint64
func GetNrScheduled() uint64
func GetUserSchedPid() int
func IsSMTActive() (bool, error)
func KhugepagePid() uint32
func LoadSkel() unsafe.Pointer
func NotifyComplete(nr_pending uint64) error
type BssData
    func (data BssData) String() string
type BssMap
type CustomScheduler
type DispatchedTask
    func NewDispatchedTask(task *QueuedTask) *DispatchedTask
type QueuedTask
type Rodata
type RodataMap
```

```
type Sched
    func LoadSched(objPath string) *Sched
    func (s *Sched) AssignUserSchedPid(pid int) error
    func (s *Sched) Attach() error
    func (s *Sched) BlockTilReadyForDequeue(ctx context.Context)
    func (s *Sched) Close()
    func (s *Sched) DequeueTask(task *QueuedTask)
    func (s *Sched) DispatchTask(t *DispatchedTask) error
    func (s *Sched) EnableSiblingCpu(lvlId, cpuId, siblingCpuId int32) error
    func (s *Sched) GetBssData() (BssData, error)
    func (s *Sched) GetRoData() (Rodata, error)
    func (s *Sched) GetUeiData() (UserExitInfo, error)
    func (s *Sched) ReadyForDequeue() bool
    func (s *Sched) SelectCPU(t *QueuedTask) (error, int32)
    func (s *Sched) SetDebug(enabled bool)
    func (s *Sched) Start()
    func (s *Sched) Stopped() bool
    func (s *Sched) SubNrQueued() error
type UeiMap
type UserExitInfo
    func (uei *UserExitInfo) GetMessage() string
    func (uei *UserExitInfo) GetReason() string
```

@ianchen0119

Next Step: Gthulhu



完成初版的 scx_goland_core 後，我開始思考一個問題：

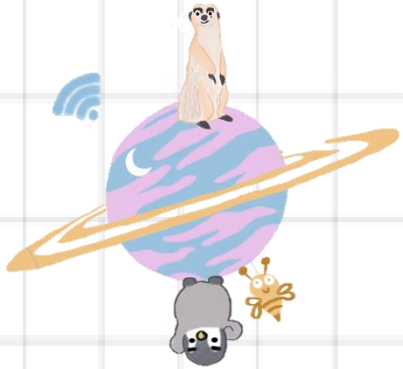
- 框架的目標受眾在哪？
- 我能利用框架解決什麼痛點？

為此，我丟出了一個名爲 Scheduling Policy as Configuration（排程策略即配置）的概念，使用者只需要透過 Restful API 就能調整多節點叢集下每個機器的排程邏輯。

為何大型語言模型與服務可觀測性，未來有機會讓 AI 根據既有的效能指標動態調整排程邏輯以最佳化 workload 的處理能力。

@ianchen0119

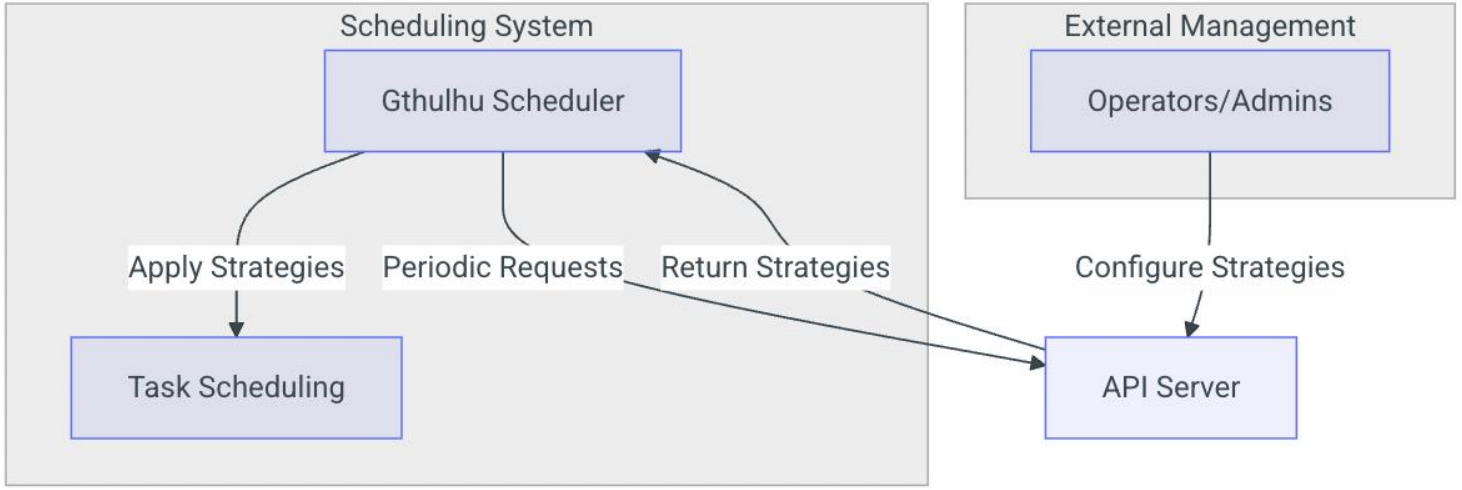
Next Step: Gthulhu



為了解決這個問題，克蘇魯誕生了！

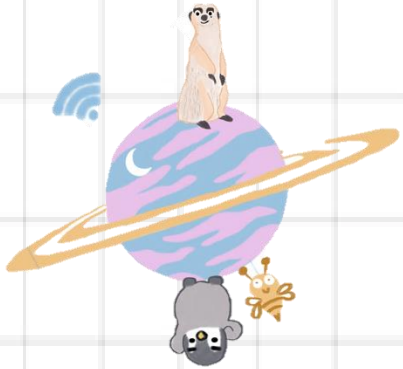
Gthulhu 目前已經能根據使用者提供的設定檔為特定的 workloads 達到最佳化。接下來，我們將試圖：

- 最佳化網路問題
- 開發多叢集的管理系統
- 累積更多案例
- CNCF landscape



https://gthulhu.github.io/docs/

問與答



@ianchen0119