獨孤派作業系統
spinlock

中正大學 作業系統實驗室
指導教授：羅習五

負責助教

- 作業目標：
  - spinlock是Linux kernel中廣泛使用的鎖定技術
  - 了解當critical section很短時，應該使用spinlock
  - 了解硬體實現spinlock時的限制
  - 更進一步的了解SNOOP在軟體造成的現象
  - 知道在何時該選擇什麼樣的spinlock
- 教學網頁：https://www.cs.ccu.edu.tw/~shiwulo/course/2018-os/ecourse/hw05/
- 負責助教
  - 吳孟昕
  - 607410008@alum.ccu.edu.tw
- 繳交期限：107/12/13 23:59:59，繳交方式：助教於12/6前公佈

作業資訊

- 你可以使用12核心，24硬體執行緒的電腦（AMD Threadripper）
  - 帳號我都已經建立好了，名稱：學號。密碼：你ecourse上的mail。
- 無論你使用哪種方式，建議使用who指令看看系統中有哪些人正在使用電腦，盡可能的在單純的環境下進行測試
- 作業的程式碼位於：https://goo.gl/cWHGaf
  - 請注意，作業中的程式碼寫得不是很好，可能偶爾會有錯誤，請自行修正

評分方式

請選定ARM平台或者是x86平台，請參考20、28頁，並回答下列問題。

1. （60pt）重複實驗（spinlock[1~3]），並解釋為什麼每個thread拿到lock的機率並不一樣。申論。
2. （20pt）測試效能（spinlock[1~3]），請問這三種spinlock的效能比較，說明你的比較方法。申論。
3. （10pt）雖然ticketlock看起來很公平，但while loop執行的太密集，請你加入「 asm("pause") 」讓CPU更有效率。程式碼。
4. （10pt）你是否可以使用atomic_compare_exchange實作出具有FIFO功能的spinlock。程式碼。可參考delivery_spinlock.c

注意事項：

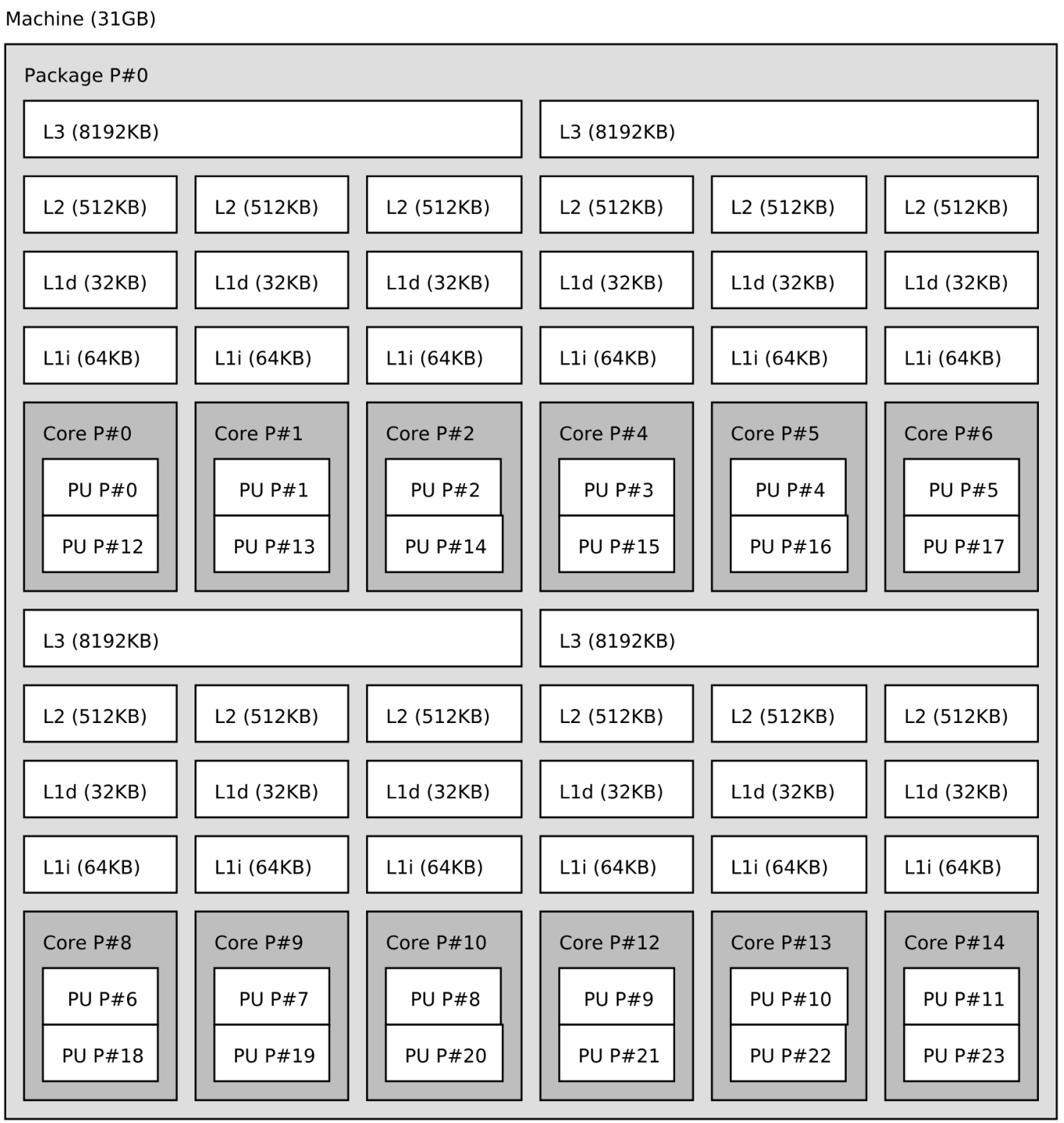
- 目前已經針對AMD Threadripper進行分析（第6~17頁）
- 請大家模仿6~17頁的分析，對自己的電腦進行分析
  - 例如：系上的Linux主機
  - 例如：OSLab的主機：numa1.cs.ccu.edu.tw，lonux.cs.ccu.edu.tw
  - 例如：自行安裝的virtual machine
- 關於AWS
  - AWS（亞馬遜雲端服務）會依照開機時間收費，在這段期間內請同學盡量使用
  - 由於AWS收費較貴，暫時只開放雙核心ARM伺服器：18.212.120.210
  - 如果很多同學想要使用ARM伺服器，我們可以租用16核心的來玩
  - 想玩的人請填寫表格：https://goo.gl/SpGDZd，填寫時間到12/12:02:00

spinlock1：pthread spinlock

- 主要由下列二函數所組成
  - pthread_spin_lock()
  - pthread_spin_unlock()
- 初始化和結束使用
  - pthread_spin_destroy(3)
  - pthread_spin_init(3)

架構圖：
140.123.97.156，
numa1.cs.ccu.edu.tw

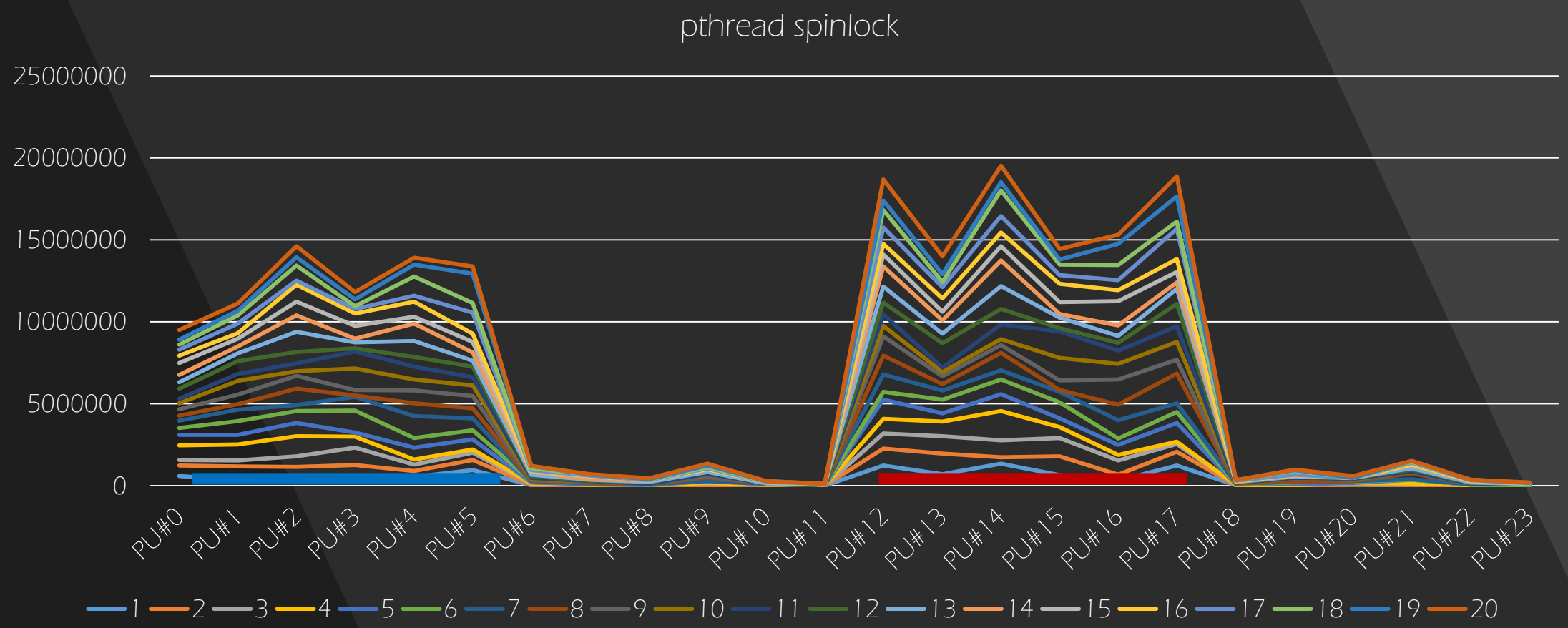
- 注意一下編號方式，PU#0和PU#12位在同一個core上



實驗方法

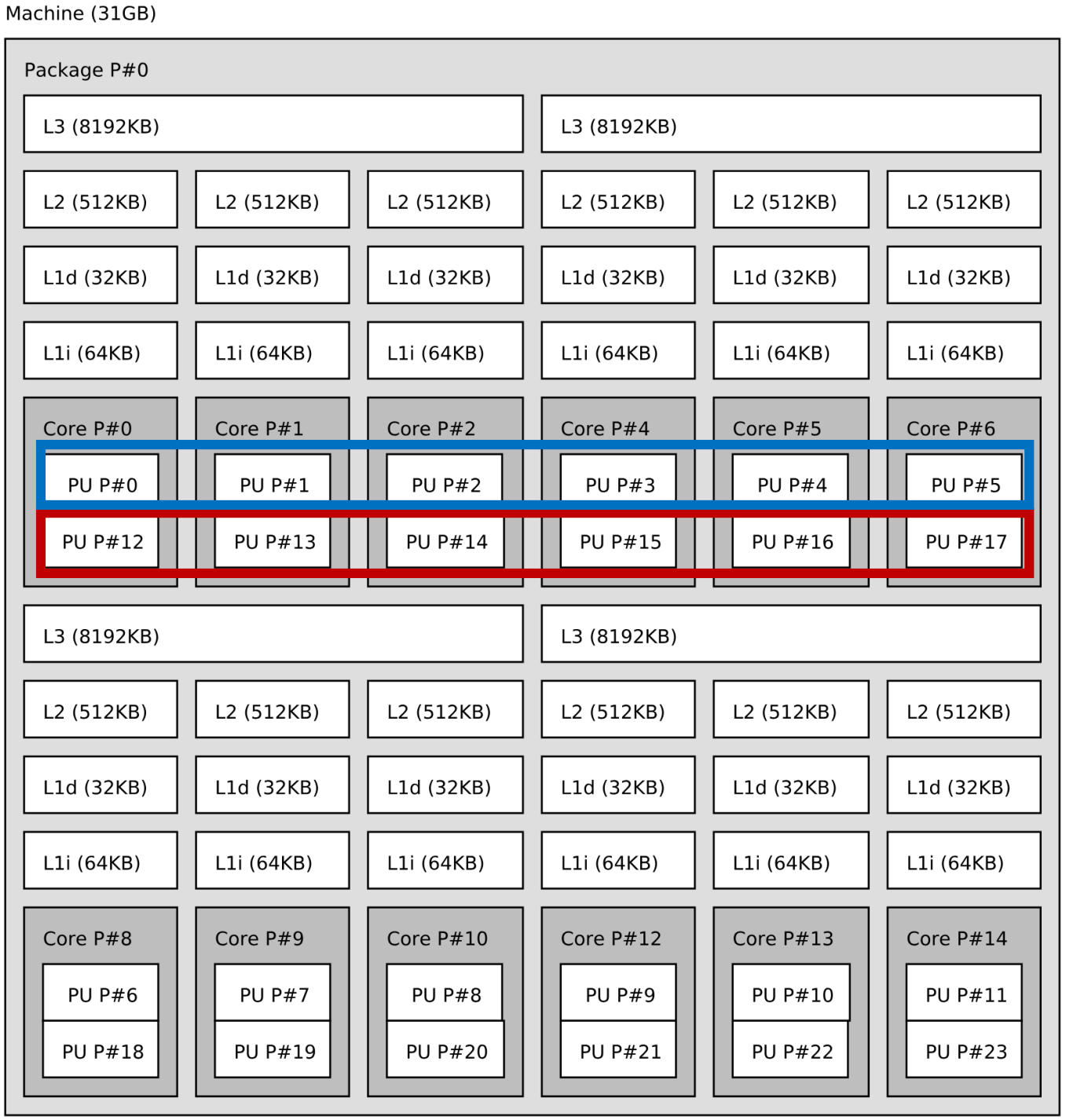
- 系統共有24個hardware threads，其中第24個是master thread，其餘的是slave threads
- 特別注意master thread和slave thread功能一模一樣
- 在這個實驗中，我們希望知道目前pthread在NUMA的架構下，對於spinlock的實現是否夠好

實驗結果


pthread spinlock
25000000
20000000
15000000
10000000
5000000
0
PU#0
PU#1
PU#2
PU#3
PU#4
PU#5
PU#6
PU#7
PU#8
PU#9
PU#10
PU#11
PU#12
PU#13
PU#14
PU#15
PU#16
PU#17
PU#18
PU#19
PU#20
PU#21
PU#22
PU#23
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

pthread的表現

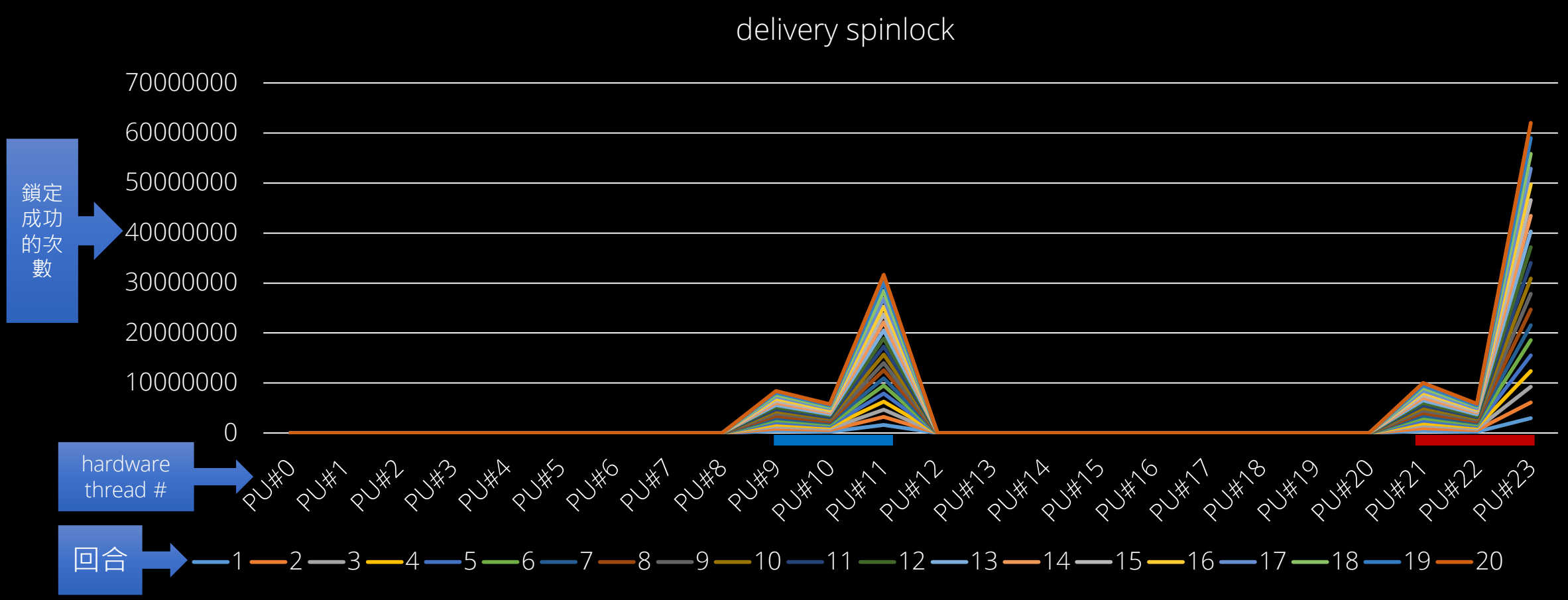
- 對照上個實驗，可以發現0~5和12~17拿到lock的機會比較大
- 請對照右圖的NUMA架構
- 明顯的其中一顆處理器拿到lock的機會比另外一顆處理器高很多
- pthread的spinlock對於NUMA的支援不夠好



spinlock2：delivery spinlock

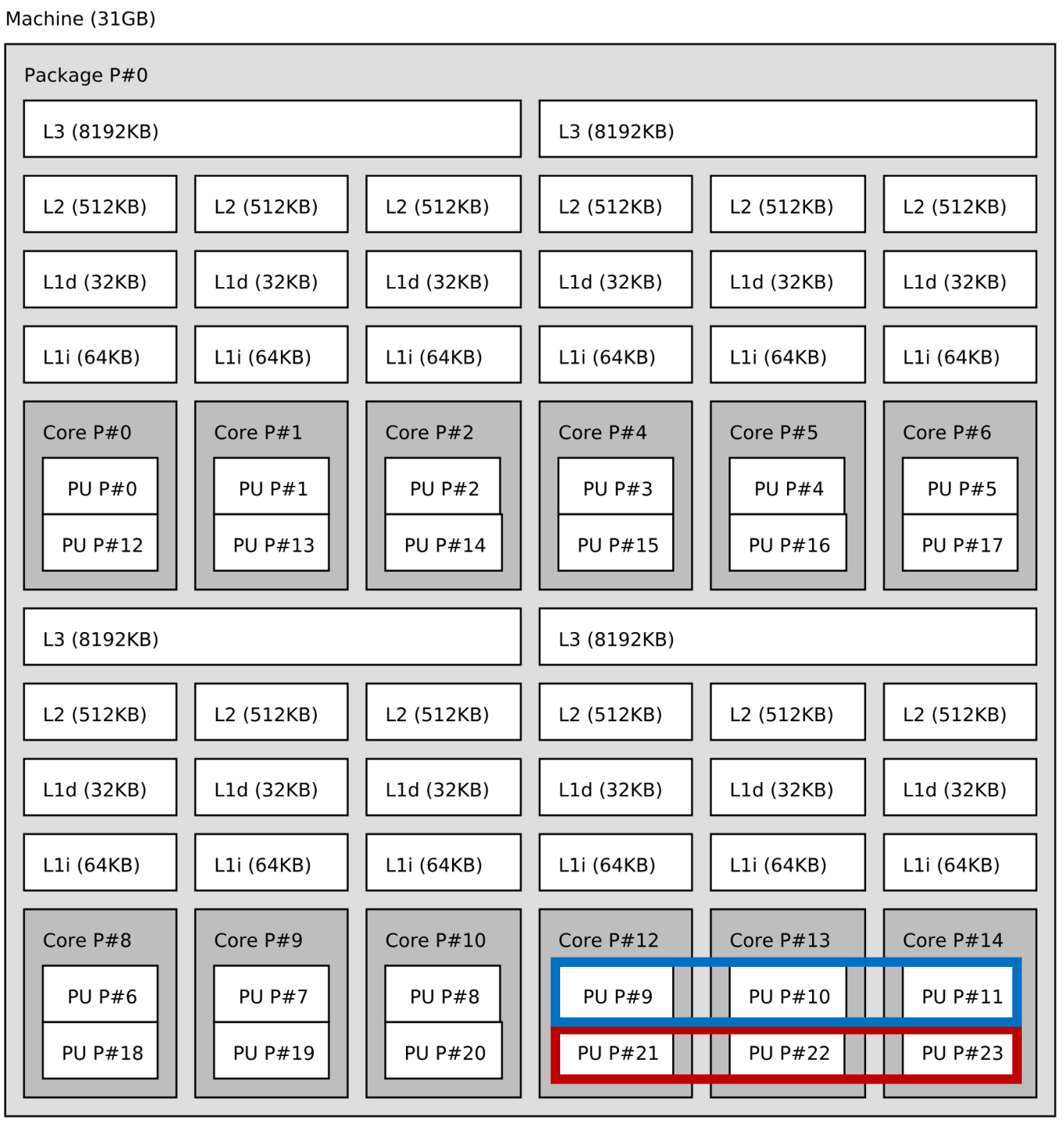
- 系統共有24個hardware threads，其中第24個是master thread，其餘的是slave threads
- Slave threads間彼此競爭CS
- Slave離開CS的時候，會先將CS交給master，隨後master再將lock解開
- 在這個實驗中，我們希望知道master thread（#23）解開lock以後，哪一顆處理器拿到lock的機會最大。

實驗結果



pthread的表現

- 對照上個實驗，可以發現21~23和9~11拿到lock的機會比較大
- 請對照右圖的NUMA架構
- 所有人拿到lock以後，都會先將lock交給23
- 23將lock交付出去時，明顯的有區域性
- Core P#14將lock交付給Core P#12的機率更高



Threadripper

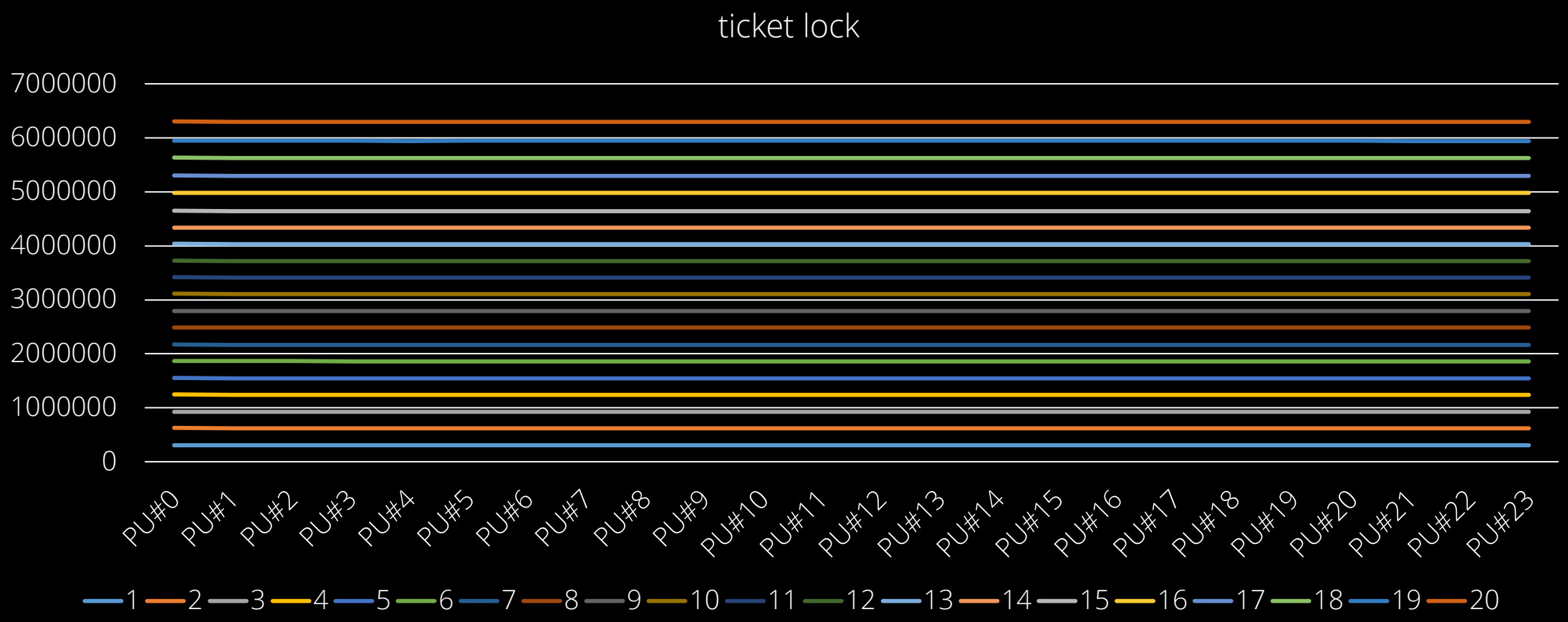
- 從左圖可以發現，Core 12與Core 14位於同一個快速交換網路上

https://www.hardwarezone.com.sg/feature-amd-ryzen-threadripper-what-you-need-know-about-amd-s-new-monster-cpu-and-x399-chipset

spinlock3：ticket lock

- 系統共有24個hardware threads
- 每個thread依照「叫號」進入CS
- 在這個實驗中，我們希望知道ticket lock除了FIFO以外，是否公平
- 此外我們也想知道ticket lock的overhead是否比較高

實驗結果



第20回的鎖定成功次數

- 從右方的表格可以看出，ticket lock的鎖定很有效率，為pthread spinlock的82.5%
- ticket lock維持了高效率，並且於NUMA機器上保證了公平性

名稱	鎖定成功次數
pthread spinlock	183,261,937
ticket lock	151,155,527
delivery lock	124,228,069

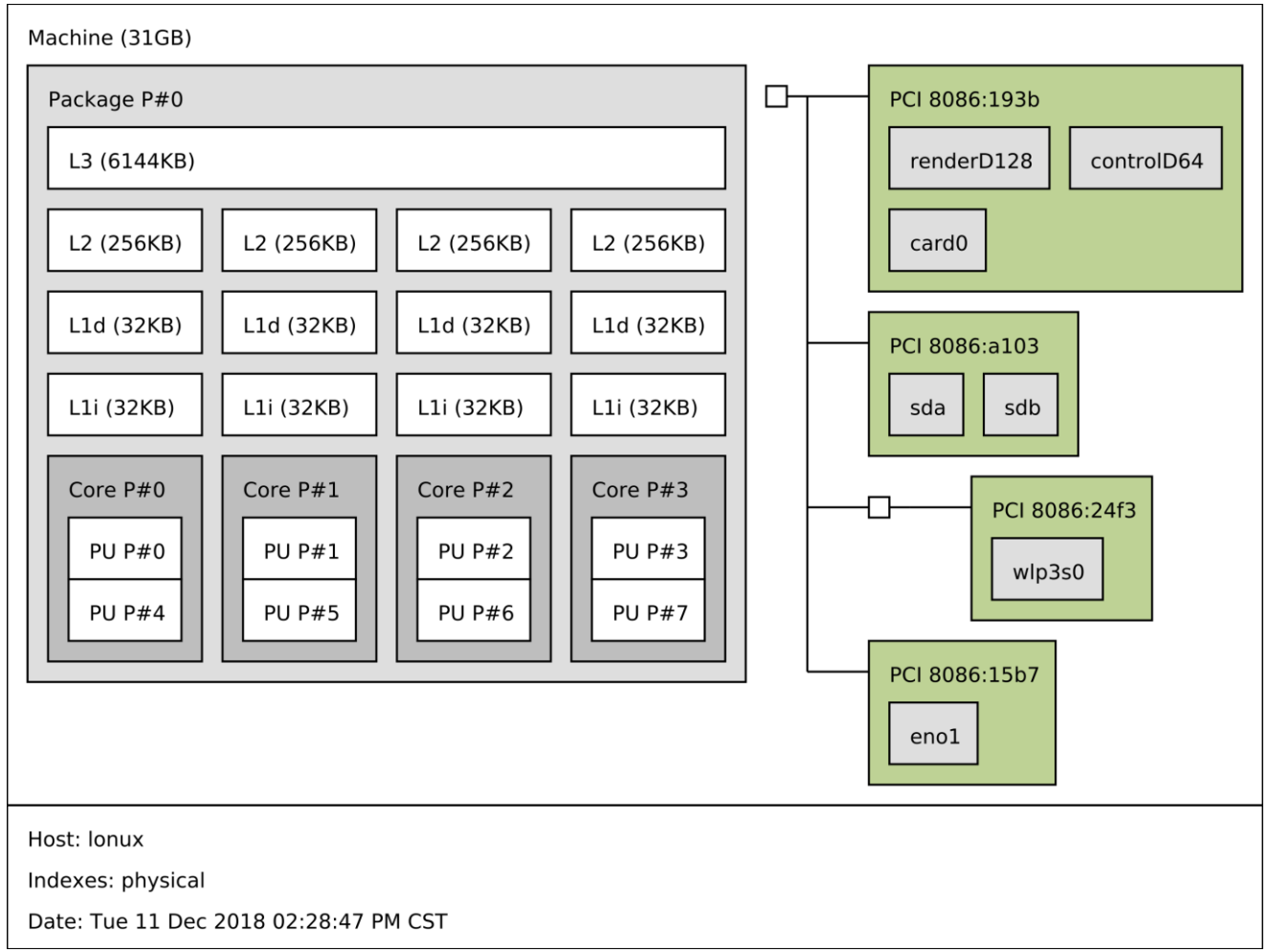
自行實作

取得硬體架構的指令

- lstopo --of svg > x86-64.svg

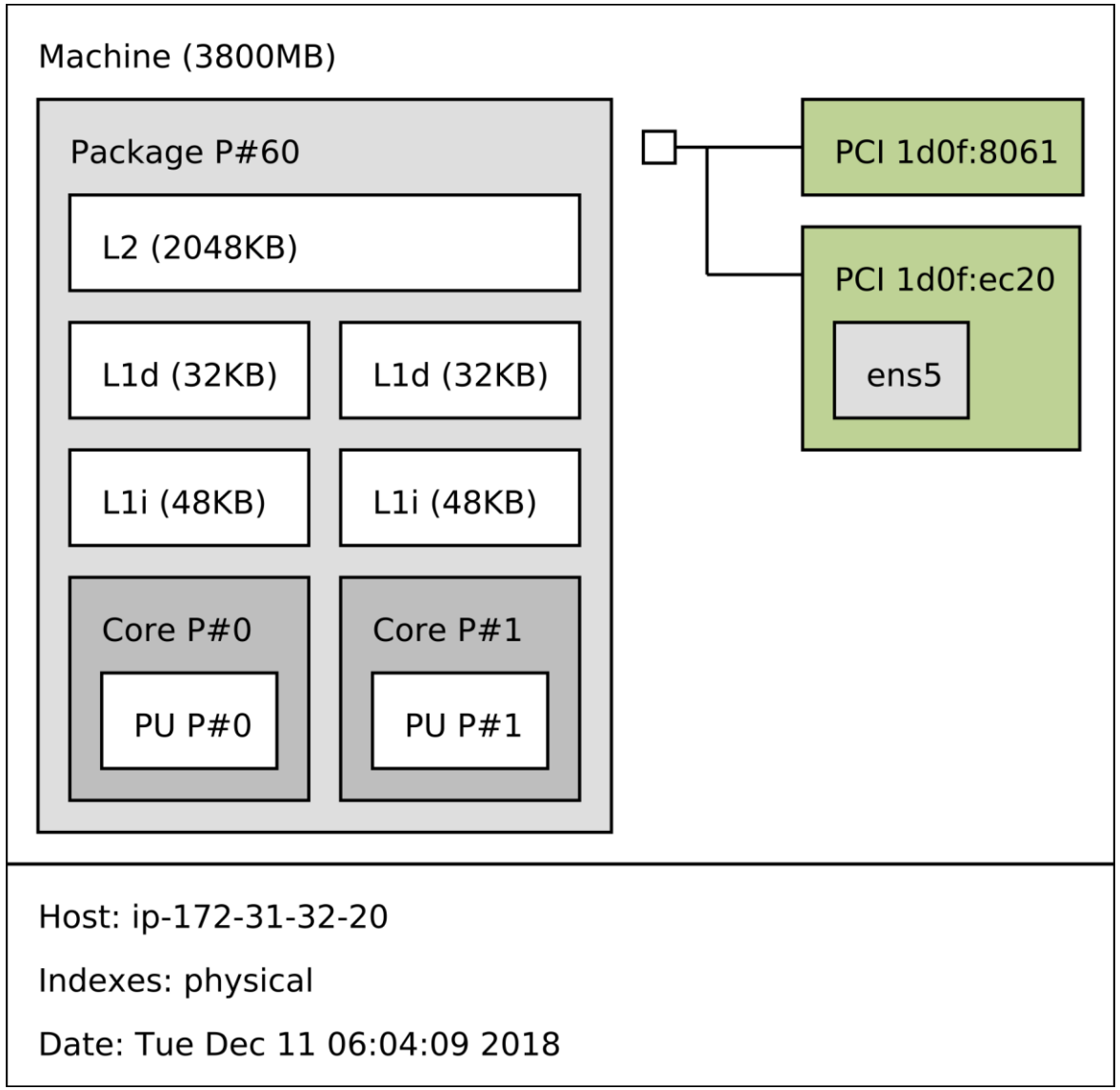
Lonux topology

- 使用ssh登入下列伺服器
- 140.123.97.157
- lonux.cs.ccu.edu.tw



ARM topology

- 使用ssh登入下列伺服器
- 18.212.120.210
- ec2-18-212-120-210.compute-1.amazonaws.com



關於AWS伺服器

- 「18.212.120.210」這一台伺服器只有二顆處理器，大家可以在這個伺服器上面進行開發。
- 上述伺服器，由於CPU的數量太少，因此無法看出spinlock於many core的情況下的影響
- 由於AWS伺服器的租金不便宜，如果「想玩玩看」的同學請填寫下列表格，只要超過10位同學想玩，我就會購買這台many core伺服器 (只要「想玩」即可，不一定要寫到報告內或者任何產出)
  - https://goo.gl/SpGDZd，填寫時間到12/12:02:00
  - 😎just for fun😎

關於ARM的atomic operation實現（1）

```
50              my_ticket = atomic_fetch_add(next_ticket, 1);
   0x0000000000000b8c <+12>:    ldr     x0, [sp, #8]
   0x0000000000000b90 <+16>:    ldaxr   w1, [x0]
   0x0000000000000b94 <+20>:    add     w2, w1, #0x1
   0x0000000000000b98 <+24>:    stlxr   w3, w2, [x0]
   0x0000000000000b9c <+28>:    cbnz    w3, 0xb90 <ticketLock_acquire+16>
   0x0000000000000ba0 <+32>:    str     w1, [sp, #28]

51              atomic_fetch_add_explicit(next_ticket, 1, memory_order_relaxed);
   0x0000000000000ba4 <+36>:    ldr     x0, [sp, #8]
   0x0000000000000ba8 <+40>:    ldxr    w1, [x0]
   0x0000000000000bac <+44>:    add     w1, w1, #0x1
   0x0000000000000bb0 <+48>:    stxr    w2, w1, [x0]
   0x0000000000000bb4 <+52>:    cbnz    w2, 0xba8 <ticketLock_acquire+40>
```

LDXR

- 這一個指令會和下一個指令（STXR）構成atomic operation
- 細節可以參考上課的投影片

http://www.keil.com/support/man/docs/armclang_asm/armclang_asm_pge1427897418390.htm

LDXR

Home » A64 Data Transfer Instructions » LDXR

17.75 LDXR

Load Exclusive Register.

Syntax

```
LDXR  Wt , [ Xn|SP {,#0}] ; 32-bit
```

```
LDXR  Xt , [ Xn|SP {,#0}] ; 64-bit
```

Where:

`Wt`
Is the 32-bit name of the general-purpose register to be transferred.

`Xt`
Is the 64-bit name of the general-purpose register to be transferred.

`Xn|SP`
Is the 64-bit name of the general-purpose base register or stack pointer.

Usage

Load Exclusive Register derives an address from a base register value, loads a 32-bit word or a 64-bit doubleword from memory, and writes it to a register. The memory access is atomic. The PE marks the physical address being accessed as an exclusive access. This exclusive access mark is checked by Store Exclusive instructions. See *Synchronization and semaphores* in the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*. For information about memory accesses see *Load/Store addressing modes* in the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*.

STXR

- 這裡要特別注意的是，如果是exclusive written，那麼Ws的值會是0
- 換句話說，如果這道指令更新了記憶體（而且一定是exclusive written），Ws會是0

17.131 STXR

Store Exclusive Register.

Syntax

```
STXR  Ws ,  Wt , [ Xn|SP {,#0}] ; 32-bit
```

```
STXR  Ws ,  Xt , [ Xn|SP {,#0}] ; 64-bit
```

Where:

`Wt`
Is the 32-bit name of the general-purpose register to be transferred.

`Xt`
Is the 64-bit name of the general-purpose register to be transferred.

`Ws`
Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written. The value returned is.

`0`
If the operation updates memory.

`1`
If the operation fails to update memory.

`Xn|SP`
Is the 64-bit name of the general-purpose base register or stack pointer.

LDXR + STXR

- 這二道指令合起來保證“atomic operation”，但memory order是最寬鬆的*memory_order_relaxed*
- 詳細的說明，請參考上課投影片，最後面的memory order小節

關於ARM的atomic operation實現（2）

```
52          atomic_fetch_add_explicit(next_ticket, 1, memory_order_consume);
   0x0000000000000bb8 <+56>: ldr    x0, [sp, #8]
   0x0000000000000bbc <+60>: ldaxr  w1, [x0]
   0x0000000000000bc0 <+64>: add    w1, w1, #0x1
   0x0000000000000bc4 <+68>: stxr   w2, w1, [x0]
   0x0000000000000bc8 <+72>: cbnz   w2, 0xbbc <ticketLock_acquire+60>

53          atomic_fetch_add_explicit(next_ticket, 1, memory_order_acquire);
   0x0000000000000bcc <+76>: ldr    x0, [sp, #8]
   0x0000000000000bd0 <+80>: ldaxr  w1, [x0]
   0x0000000000000bd4 <+84>: add    w1, w1, #0x1
   0x0000000000000bd8 <+88>: stxr   w2, w1, [x0]
   0x0000000000000bdc <+92>: cbnz   w2, 0xbd0 <ticketLock_acquire+80>
```

關於ARM的atomic operation實現（3）

```
54          atomic_fetch_add_explicit(next_ticket, 1, memory_order_release);
   0x0000000000000be0 <+96>:  ldr     x0, [sp, #8]
   0x0000000000000be4 <+100>: ldxr    w1, [x0]
   0x0000000000000be8 <+104>: add     w1, w1, #0x1
   0x0000000000000bec <+108>: stlxr   w2, w1, [x0]
   0x0000000000000bf0 <+112>: cbnz    w2, 0xbe4 <ticketLock_acquire+100>

55          atomic_fetch_add_explicit(next_ticket, 1, memory_order_acq_rel);
   0x0000000000000bf4 <+116>: ldr     x0, [sp, #8]
   0x0000000000000bf8 <+120>: ldaxr   w1, [x0]
   0x0000000000000bfc <+124>: add     w1, w1, #0x1
   0x0000000000000c00 <+128>: stlxr   w2, w1, [x0]
   0x0000000000000c04 <+132>: cbnz    w2, 0xbf8 <ticketLock_acquire+120>
```

關於ARM的atomic operation實現（４）

```
56          atomic_fetch_add_explicit(next_ticket, 1, memory_order_seq_cst);
   0x000000000000c08 <+136>:ldr    x0, [sp, #8]
   0x000000000000c0c <+140>:ldaxr  w1, [x0]
   0x000000000000c10 <+144>:add    w1, w1, #0x1
   0x000000000000c14 <+148>:stlxr  w2, w1, [x0]
   0x000000000000c18 <+152>:cbnz   w2, 0xc0c <ticketLock_acquire+140>
```