Wait-free & Lock-free

Concurrency is the power in 21st century.

scc@teamt5.org

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters

Reflecting on Last Year's Challenges: Overcoming 60% Unmastered Material

TEAMT5
杜 浦 數 位 安 全

Related Works

TEAMT5 杜浦數位安全

並行和多執行緒程式設計

Back to Basics: Concurrency

Concurrency in C++: A Programmer's Overview

Smarter Cpp Atomic Smart Pointers - Efficient Concurrent Memory Management

Lock-free Atomic Shared Pointers Without a Split Reference Count? It Can Be Done!

Hazard Pointer Synchronous Reclamation Beyond Concurrency TS2

C++ atomics, from basic to advanced. What do they really do?

The speed of concurrency (is lock-free faster?)

Non-Uniform Memory Architecture (NUMA): A Nearly Unfathomable Morass of Arcana - Fedor Pikus CppNow

Wait-Free Queues With Multiple Enqueuers and Dequeuers

Outline

TEAMT5
杜 浦 數 位 安 全

Queueing Theory

Definition

Mutex

Barriers

Atomic

Use cases

- Hazard Pointer, RCU, and work stealing

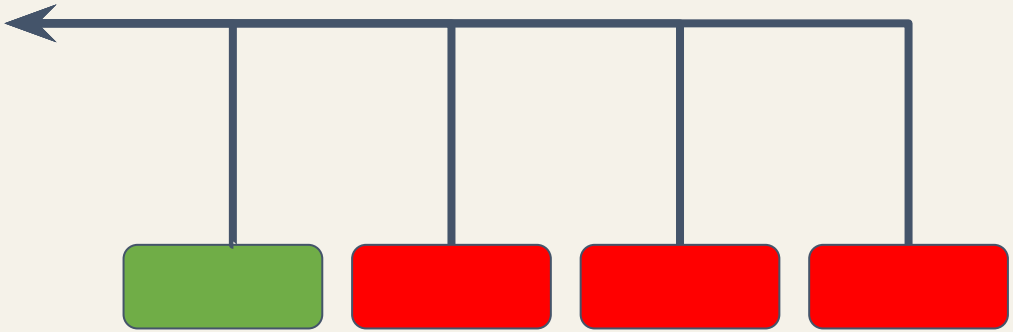
Intro. Wait-Free and Lock-Free Prog.

TEAMT5 杜浦數位安全

```
void foo() {
    static std::mutex m_;
    std::unique_lock lock(m_);
    // …
}
```

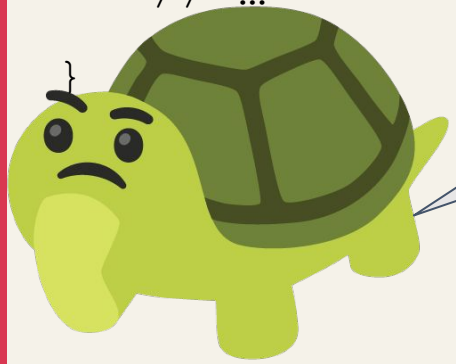
Intro. Wait-Free and Lock-Free Prog.

```
void foo() {
    static std::mutex m_;
    std::unique_lock lock(m_);
    // …
}
```

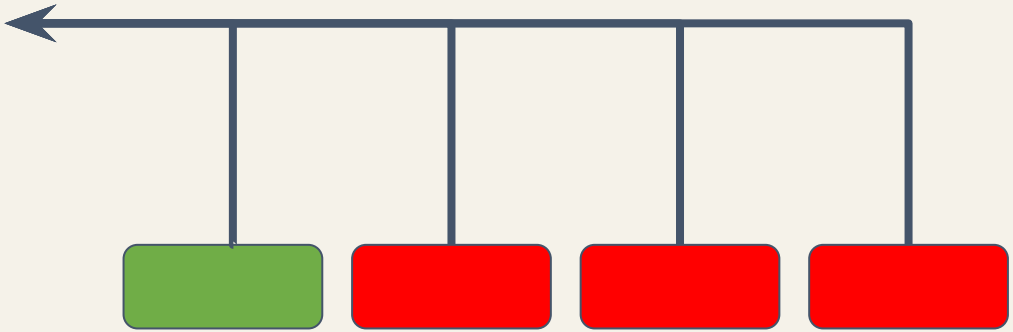


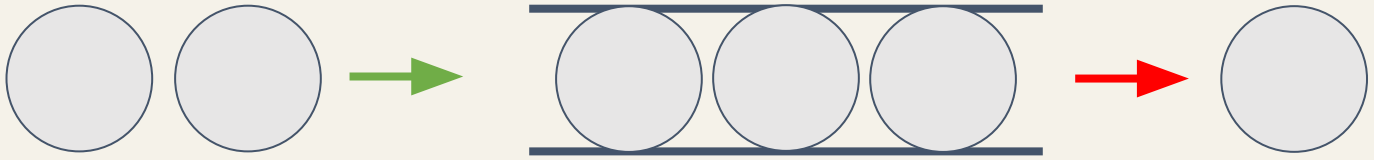
Intro. Wait-Free and Lock-Free Prog.

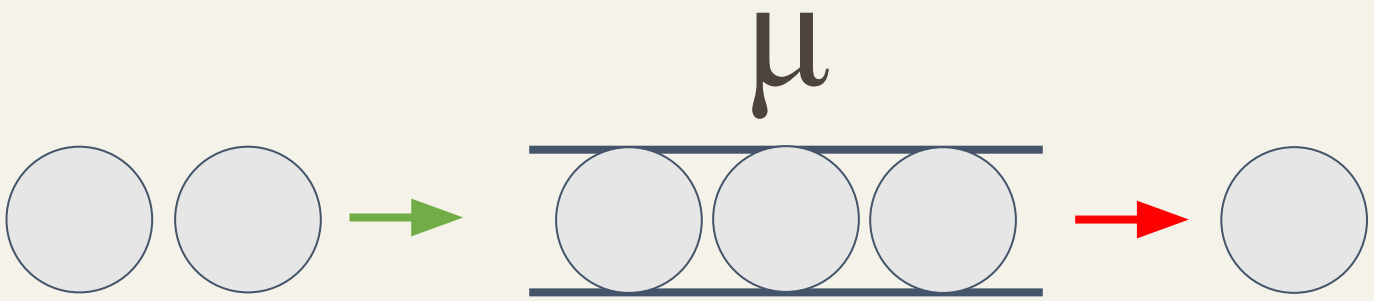
```
void foo() {
    static std::mutex m_;
    std::unique_lock lock(m_);
    // …
}
```



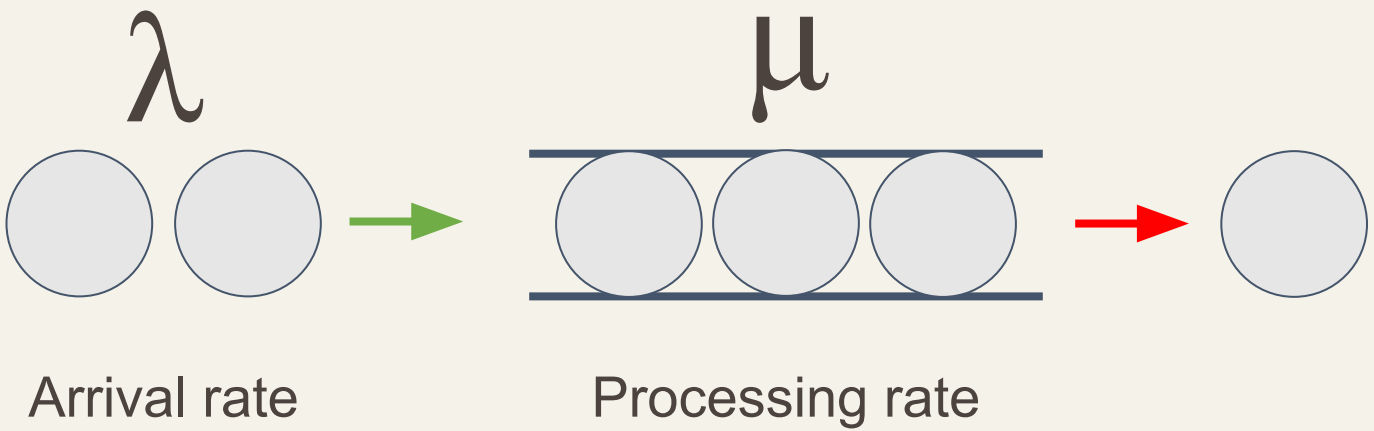
Toward Zero Starvation?

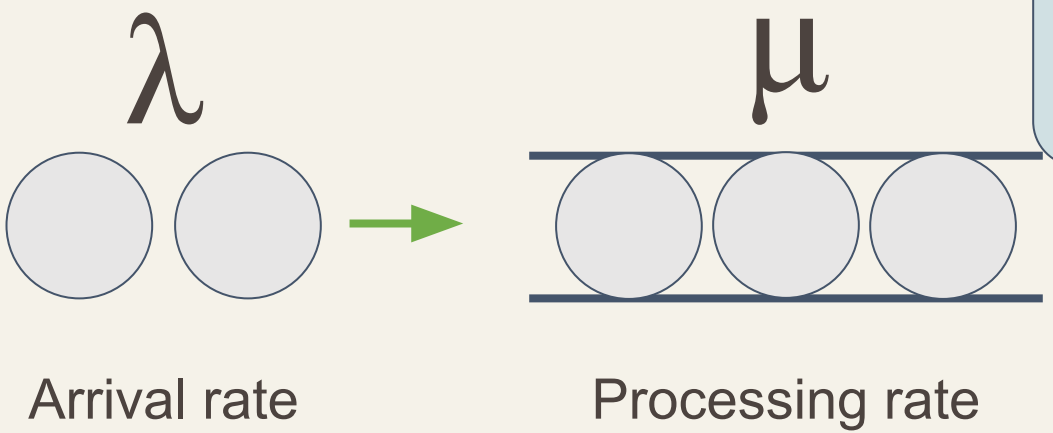




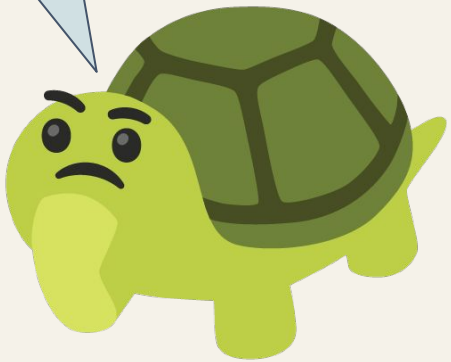


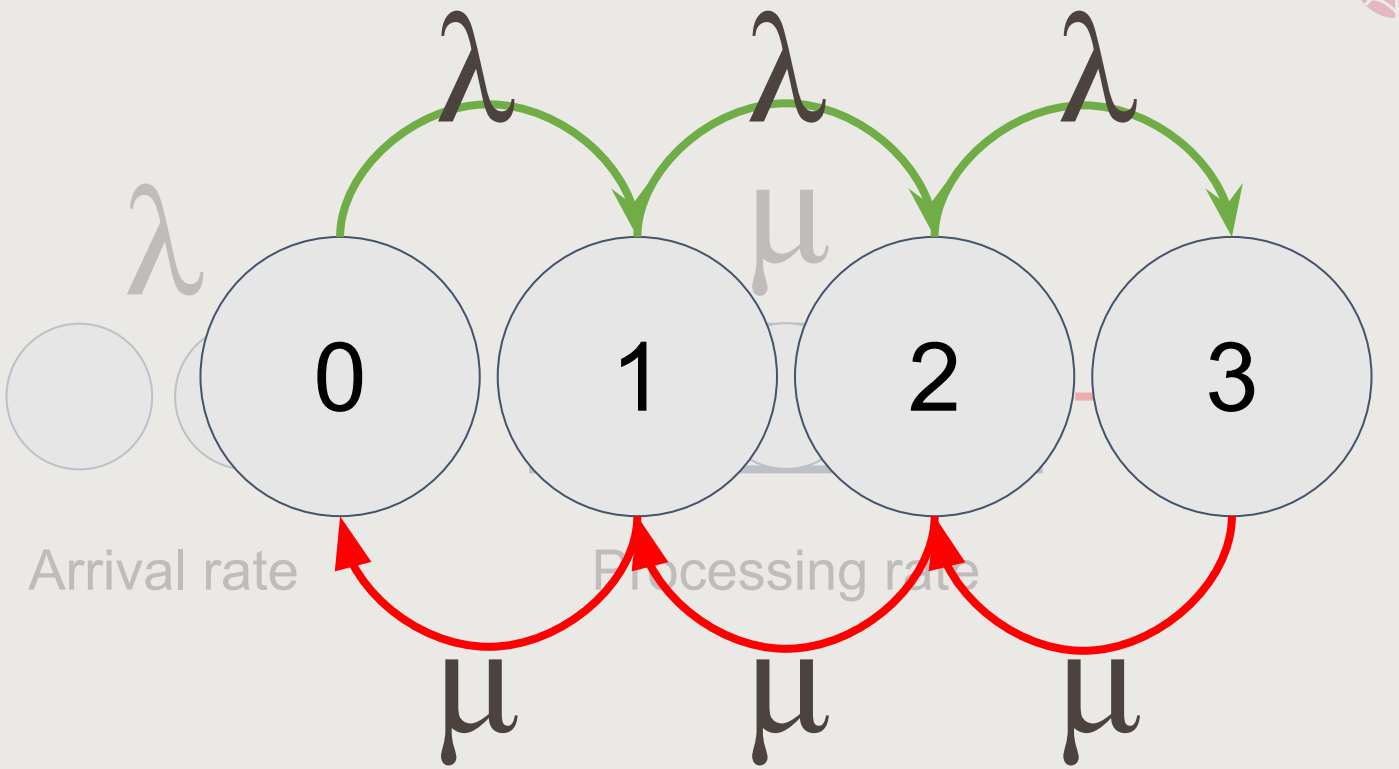
Processing rate

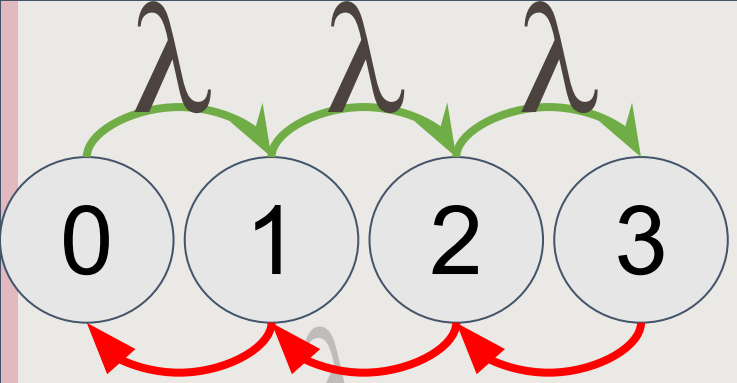

λ
μ
Arrival rate
Processing rate

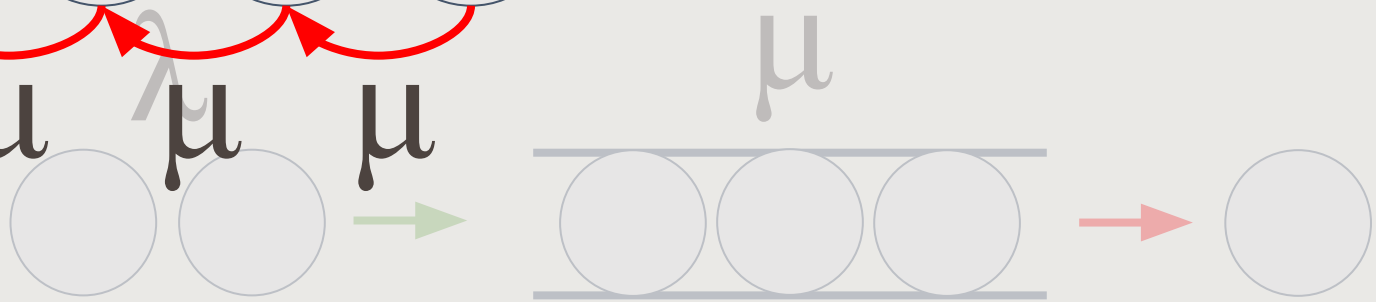


Arrival iid Poisson distribution
Processing iid exponential distribution.



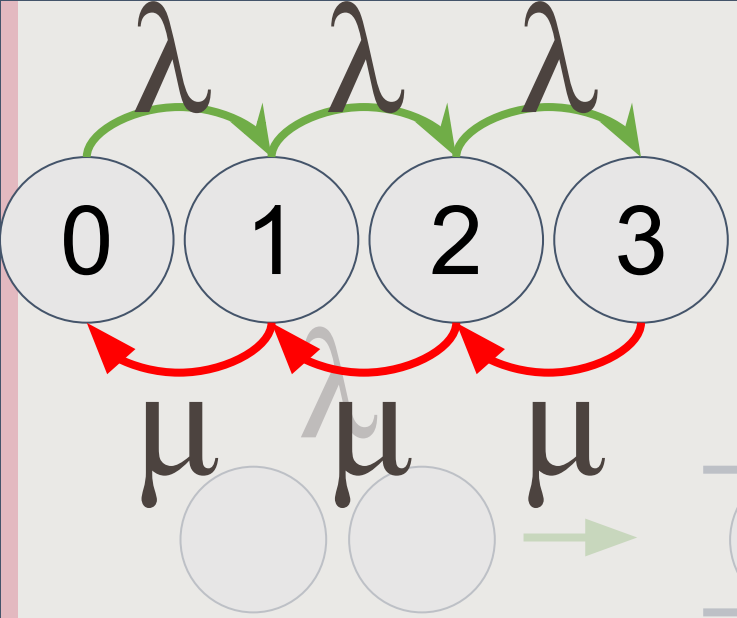

λ
λ
λ
0
1
2
3
μ
μ
μ





Arrival rate

Processing rate

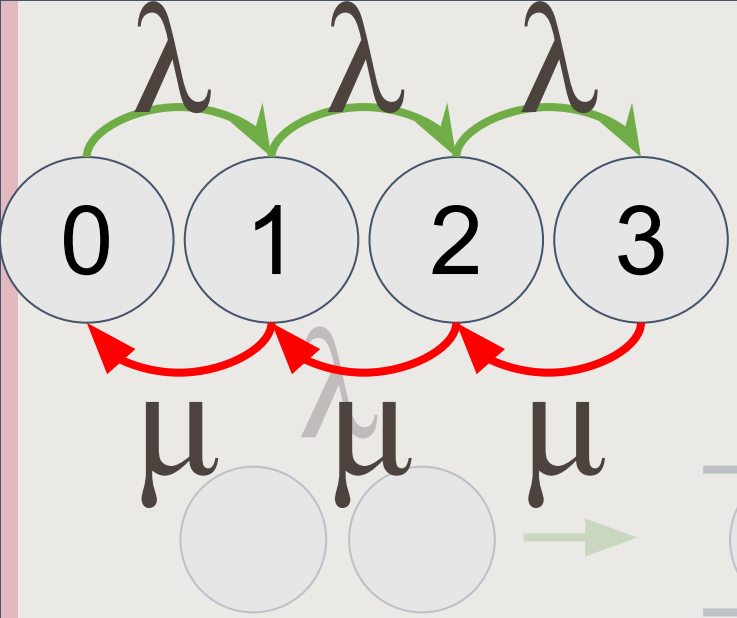


$$P_0 \cdot \lambda = P_1 \cdot \mu$$

TEAMT5
杜 浦 數 位 安 全

Arrival rate

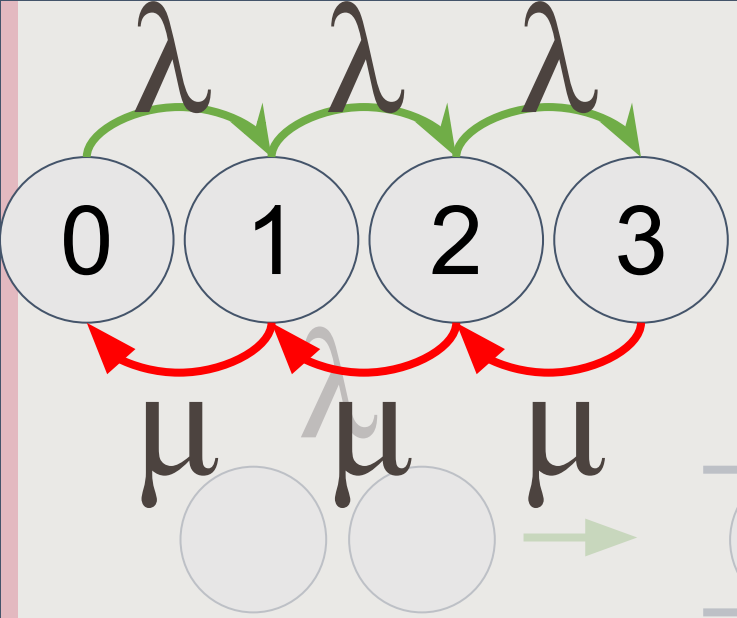
Processing rate



$$P_0 \cdot \lambda = P_1 \cdot \mu \qquad P_1 = \frac{\lambda}{\mu} \cdot P_0$$

Arrival rate

Processing rate

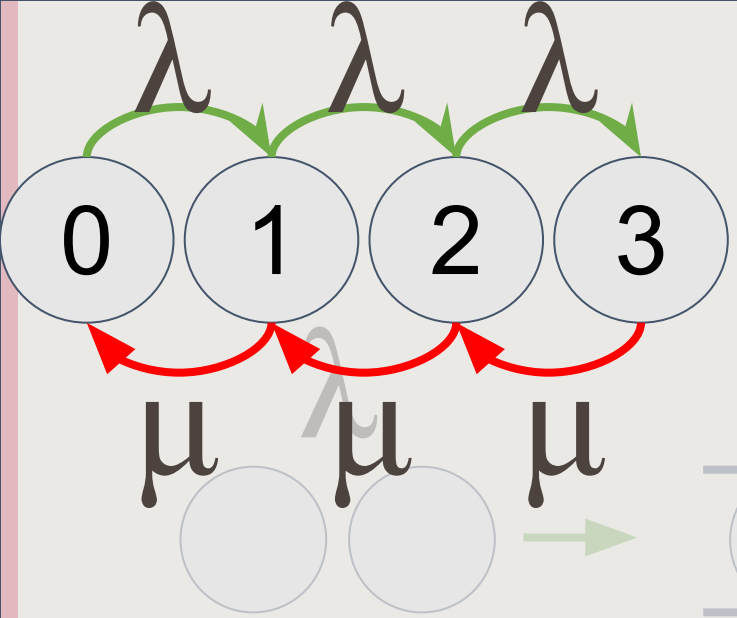


Arrival rate

Processing rate

$$P_0 \cdot \lambda = P_1 \cdot \mu \qquad P_1 = \frac{\lambda}{\mu} \cdot P_0$$

$$P_1 \cdot \lambda + P_1 \cdot \mu = P_0 \cdot \lambda + P_2 \cdot \mu$$



Arrival rate

Processing rate

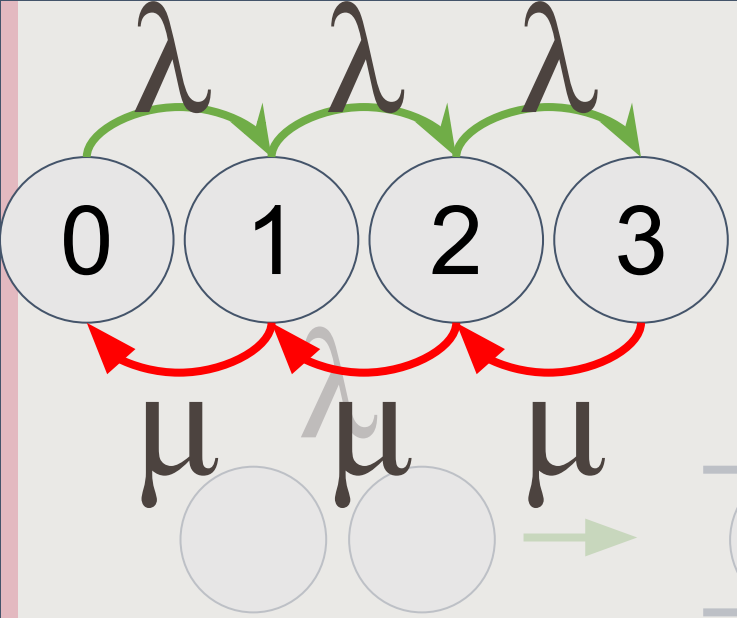
$$P_0 \cdot \lambda = P_1 \cdot \mu \qquad P_1 = \frac{\lambda}{\mu} \cdot P_0$$

$$P_1 \cdot \lambda + P_1 \cdot \mu = P_0 \cdot \lambda + P_2 \cdot \mu$$

$$P_2 = \frac{\lambda}{\mu} P_1 + \frac{1}{\mu}(\mu P_1 - \lambda P_0)$$

$$= \frac{\lambda}{\mu} P_1$$

$$= \frac{\lambda^2}{\mu^2} P_0$$



Arrival rate

Processing rate

$$P_0 \cdot \lambda = P_1 \cdot \mu$$

$$\boxed{P_1 = \frac{\lambda}{\mu} \cdot P_0}$$

$$P_1 \cdot \lambda + P_1 \cdot \mu = P_0 \cdot \lambda + P_2 \cdot \mu$$

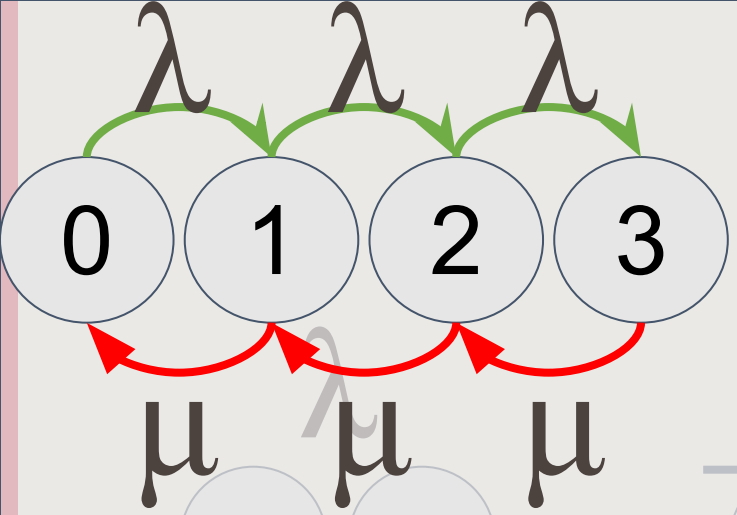
$$P_2 = \frac{\lambda}{\mu} P_1 + \frac{1}{\mu}(\mu P_1 - \lambda P_0)$$

$$= \frac{\lambda}{\mu} P_1$$

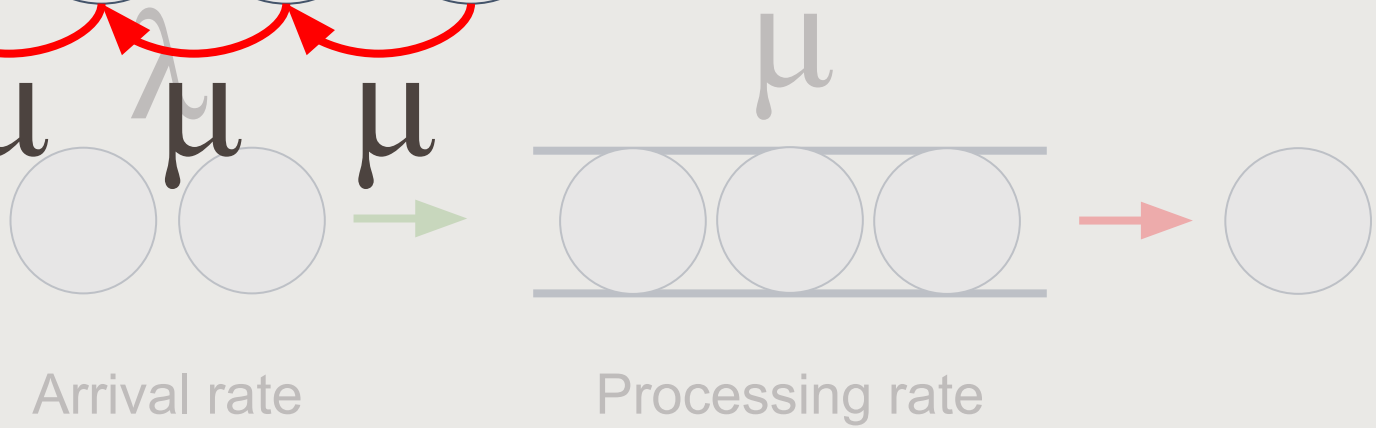
$$\boxed{= \frac{\lambda^2}{\mu^2} P_0}$$

$$P_1 = \frac{\lambda}{\mu} \cdot P_0$$

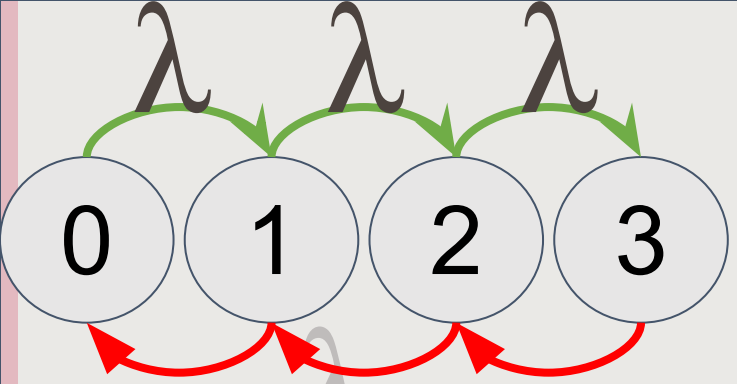
$$P_2 = \frac{\lambda^2}{\mu^2} P_0$$



$$P_n = \left(\frac{\lambda}{\mu}\right)^n \cdot P_0$$



TEAMT5 浦數位安全



$$P_n = \left(\frac{\lambda}{\mu}\right)^n \cdot P_0$$

Suppose: $n \to \infty$

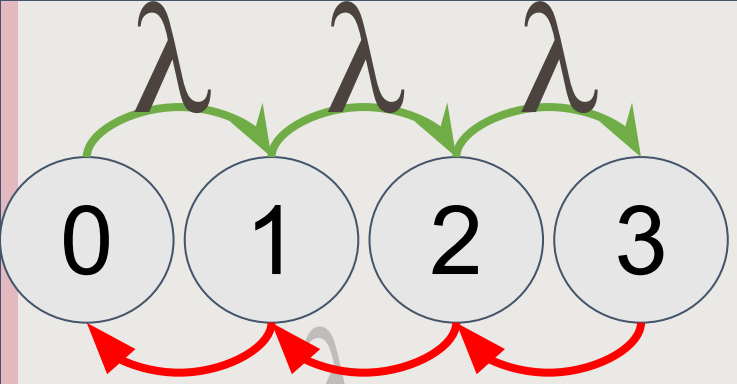
$$\lim_{N \to \infty} \Sigma_0^N P_n = 1$$

$$Let\ \rho = \frac{\lambda}{\mu}$$

Arrival rate

Processing rate

TEAMT5 浦數位安全



$$P_n = (\frac{\lambda}{\mu})^n \cdot P_0$$

Suppose: $n \to \infty$

$$\lim_{N\to\infty} \Sigma_0^N P_n = 1$$

$$Let\ \rho = \frac{\lambda}{\mu}$$

Arrival rate

Processing rate

$$P_0 \cdot \lim_{N\to\infty} \Sigma_0^N \rho^n$$

$$\to P_0 \cdot \frac{1-\rho^\infty}{1-\rho} = 1$$

$$\to P_0 = \frac{1-\rho}{1-\rho^\infty}$$

TEAMT5 浦數位安全

$$P_n = \left(\frac{\lambda}{\mu}\right)^n \cdot P_0$$

Suppose: $n \to \infty$

$$\lim_{N \to \infty} \Sigma_0^N P_n = 1$$

$$P_0 \lim_{N \to \infty} \Sigma_0^N \rho^n$$

$$\to P_0 \cdot \frac{1 - \rho^\infty}{1 - \rho} = 1$$

$$\to P_0 = \frac{1 - \rho}{1 - \rho^\infty}$$

Arrival rate

Processing rate

$$Let\ \rho = \frac{\lambda}{\mu}$$

$$P_0 = \frac{1 - \rho}{1 - \rho^\infty}$$

$$P_0 = \frac{1-\rho}{1-\rho^{\infty}}$$

$\mu >> \lambda$

P_0 $\rightarrow 0$

	Single channel	Multi channel
Notation	λ = expected arrivals per period μ = expected service done per period	λ = expected arrivals per period μ = expected service done per period c = number of channels
Probability distribution of n units in the system	$p_n = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^2$	$p_n = \frac{1}{\left(\frac{\lambda}{\mu}\right)^c \left(\frac{1}{c!}\right)\left[\frac{1}{1-p}\right] + \sum_{r=0}^{r=c-1}\left(\frac{\lambda}{\mu}\right)^r \left(\frac{1}{r!}\right)}$
Mean number of units in the system	$n_m = \frac{\lambda}{\mu - \lambda}$	$n_m = m_m + \frac{\lambda}{\mu}$
Average length of queue	$m_m = \frac{\lambda^2}{\mu(\mu - \lambda)}$	$m_m = \frac{\left(\frac{\lambda}{\mu}\right)^{c+1}}{(c-1)!\ \left(c - \frac{\lambda}{\mu}\right)^2}$
Mean time waiting for service	$n_m = \frac{\lambda}{\mu(\mu - \lambda)}$	$w_m = \frac{m_m}{\lambda}$
Mean time in the system	$t_m = \frac{1}{\mu - \lambda}$	$d_m = w_m + \frac{1}{\mu}$

http:/

It’s critical if we can wait-free.

Definition

Definitions

TEAMT5 杜浦數位安全

Lock-free terminology

Obstruction free (*Weakest guarantee*)
If all other threads are paused then any given thread will complete its operation in a bounded number of steps.

Lock free (*Most common guarantee*)
If multiple threads are operating on a data structure then after a bounded number of steps **one** of them will complete its operation.

Wait free (*Strongest guarantee*)
Every thread operating on a data structure will complete its operation in a bounded number of steps, even if other threads are also operating on the data structure.

https://www.justsoftw

Definitions

TEAMT5
杜 浦 數 位 安 全

[intro.progress] p2.2

When one or more lock-free executions run concurrently, at least one should complete.

[*Note 3*: It is difficult for some implementations to provide absolute guarantees to this effect, since repeated and particularly inopportune interference from other threads could prevent forward progress, e.g., by repeatedly stealing a cache line for unrelated purposes between load-locked and store-conditional instructions. For implementations that follow this recommendation and ensure that such effects cannot indefinitely delay progress under expected operating conditions, such anomalies can therefore safely be ignored by programmers. Outside this document, this property is sometimes termed lock-free. — *end note*]

Definitions

[intro.progress] p2.2

When one or more lock-free executions run concurrently, at least one should complete.

[*Note 3*: It is difficult for some implementations to provide absolute guarantees to this effect, since repeated and particularly inopportune interference from other threads could prevent forward progress, e.g., by repeatedly stealing a cache line for unrelated purposes between load-locked and store-conditional instructions. For implementations that follow this recommendation and ensure that such effects cannot indefinitely delay progress under expected operating conditions, such anomalies can therefore safely be ignored by programmers. Outside this document, this property is sometimes termed lock-free. — *end note*]

Wait-free did not be def. in std

Obstruction-free

If there is only one thread that is not blocked ([defns.block]) in a standard library function, a lock-free execution in that thread shall complete.

We take wait-free definition as:

Every thread operating on a data structure will complete its operation in a **bounded number of steps**, even if other threads are also operating on the data structure.

Wait-free did not be def. in std

Obstruction-free

If there is only one thread that is not blocked ([defns.block]) in a standard library function, a lock-free execution in

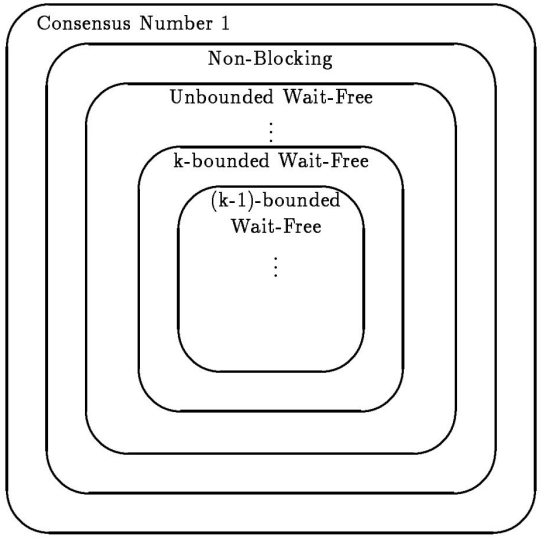
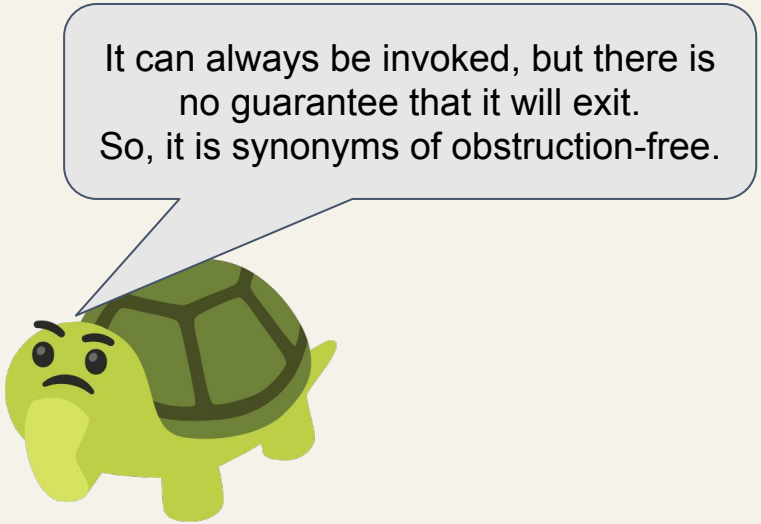


Figure 1: Hierarchy of Objects with Consensus Number 1

We take wait-free definition as:

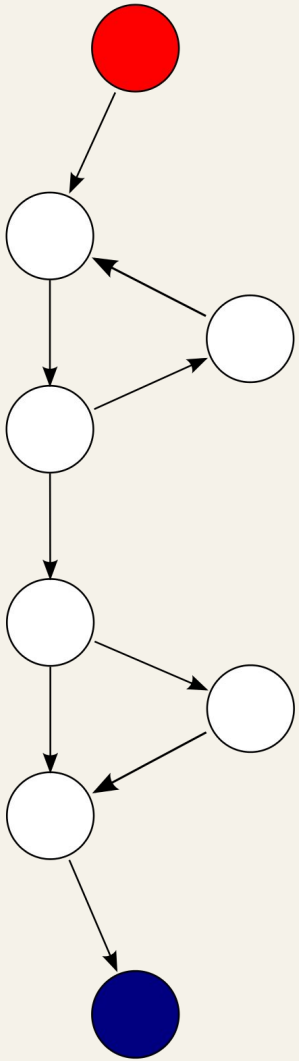
Every thread operating on a data structure will complete its operation in a **bounded number of steps**, even if other threads are also operating on the data structure.



Maurice Herlihy. 1991. Impossibility results for asynchronous PRAM (extended abstract). In Proceedings of the third annual ACM symposium on Parallel algorithms and architectures (SPAA '91). Association for Computing Machinery, New York, NY, USA, 327–336. https://doi.org/10.1145/113379.113409

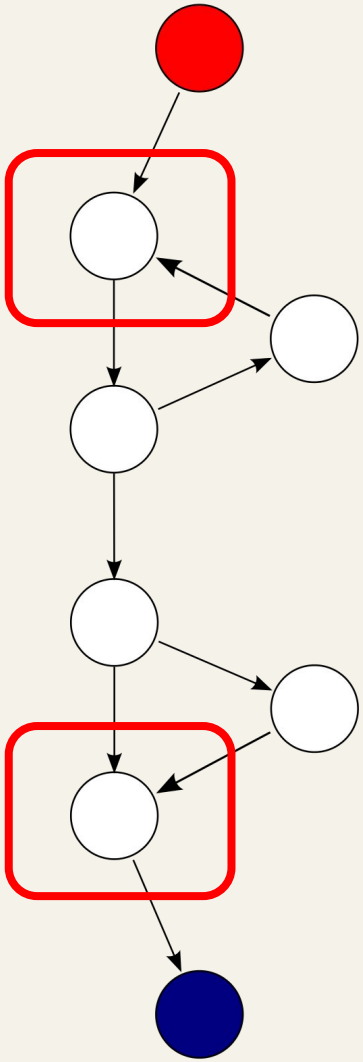
State machine

State machine



TEAMT5
杜 浦 數 位 安 全

State machine

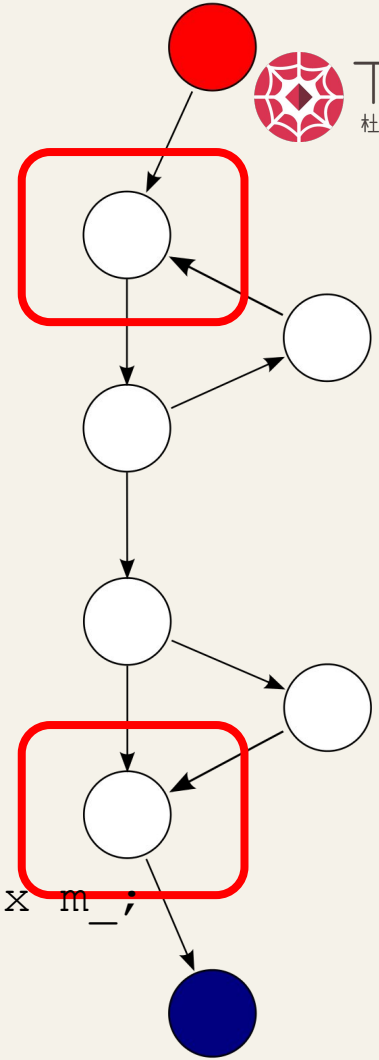


TEAMT5
杜 浦 數 位 安 全

State machine

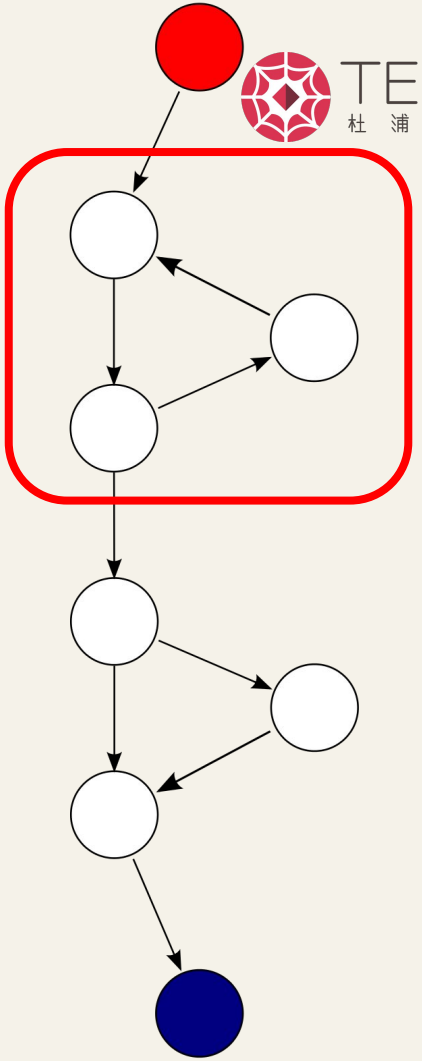
```
void update_worker() {
    // ...
    update();
}

void event_loop() {
    int status = 0;
    switch (status) {
        case UPDATE:
            update();
            break;
    };
};
```

```
void update() {
    static std::mutex m_;
    // ...
}
```



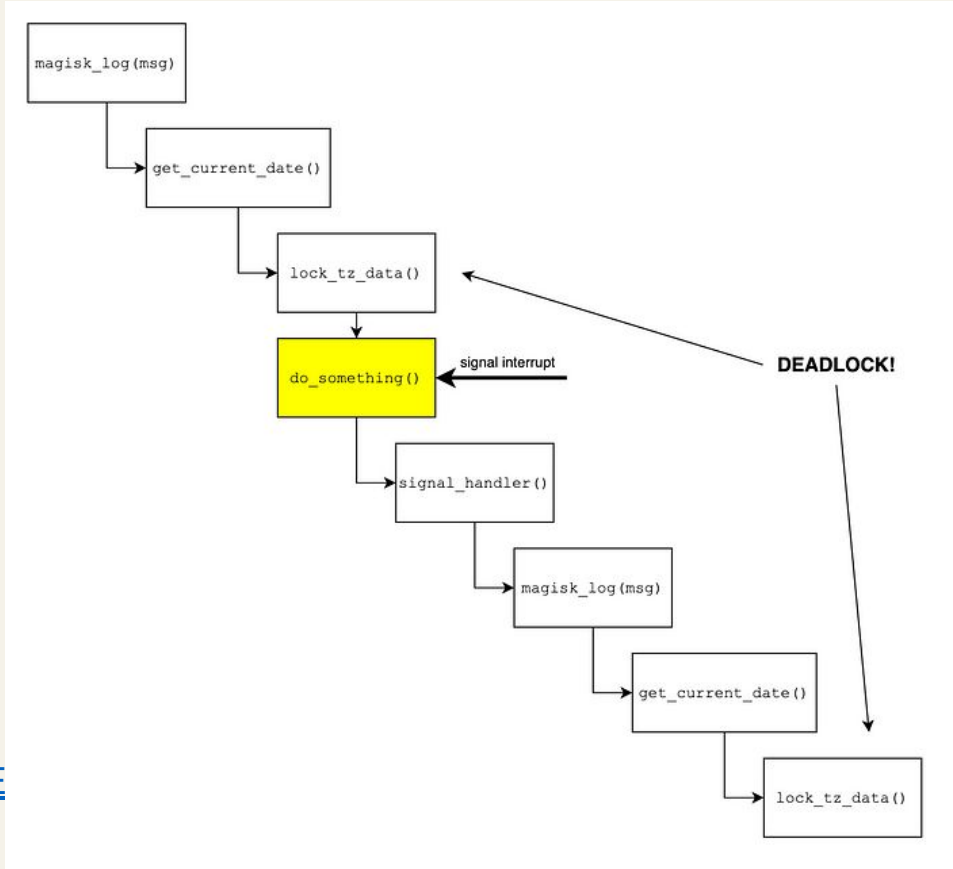
State machine

State machine helps us to figure out
if it's possible leading to a deadlock or livelock.



Magisk Deadlock

```
function magisk_log(msg):
    android_log_print(msg)
    date = get_current_date()
    date_str = format_date_string(date)
    write_file(logfile, date_str + msg)


function get_current_date():
    time = get_current_time()
    lock_tz_data()
    date = convert_to_date(time)
    unlock_tz_data()
    return date
```



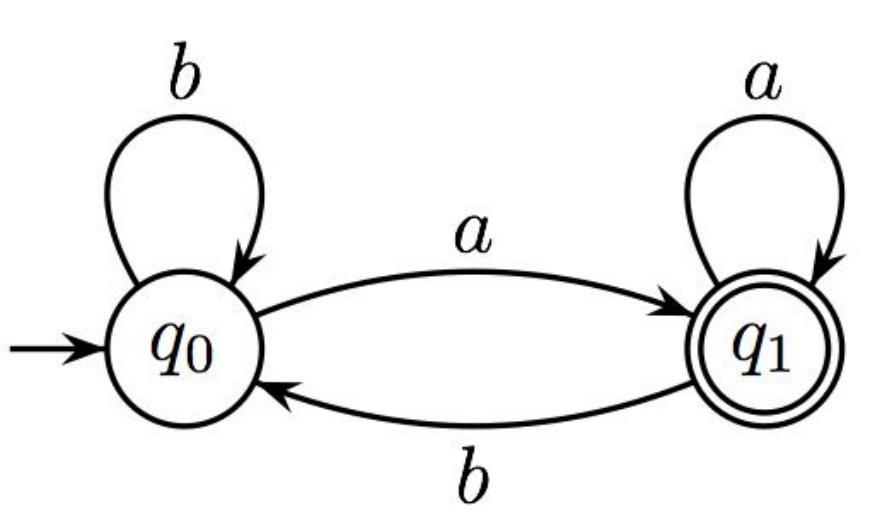
https://topjohnwu.medium.com/linux-%E7%B3%BB%E8%A8%88-logging-shenanigans-ff351e433fe6

Formal verification

Model checking

Büchi automaton

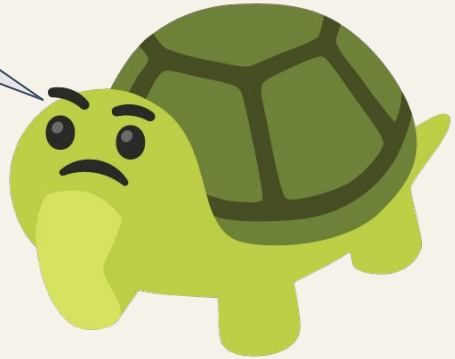
Detecting some sequence or some knots



https://en.wikipedia.org/wiki/B%C3%BCchi_automaton

Ambient isotopy

加入 TeamT5 在看的怪東西
看到睡著



https://en.wikipedia.org/wiki/Jones_polynomial

Mutex

Dekker's algorithm

```
void* p() {
    wantp = true;
    while(wantq) {
        if(turn == 2) {
            wantp = false;
            while(turn != 1){}
            wantp = true;
        }
    }
    criticalSection();
    turn = 2;
    wantp = false;
}
```

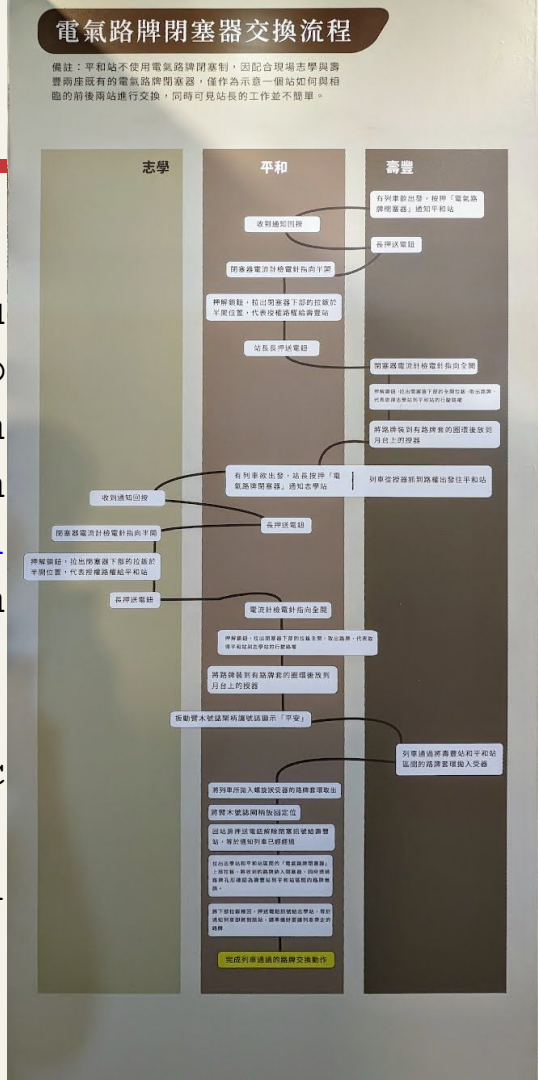
```
void* q() {
    wantq = true;
    while(wantp) {
        if(turn == 1) {
            wantq = false;
            while(turn != 2) {}
            wantq = true;
        }
    }
    criticalSection();
    turn = 1;
    wantq = false;
}
```

https://godbolt.org/z/1a9ffv1W8
https://github.com/Renjuju/Dekkers-Algorithm/blob/master/dekkers.c

Dekker's algorithm

```
    }
    criticalSection()
    turn = 2;
    wantp = false;
}
```

```
{
= tru
wantp
(turn
  wan
  whi
  wan

alSec
 1;
= fal
}
```

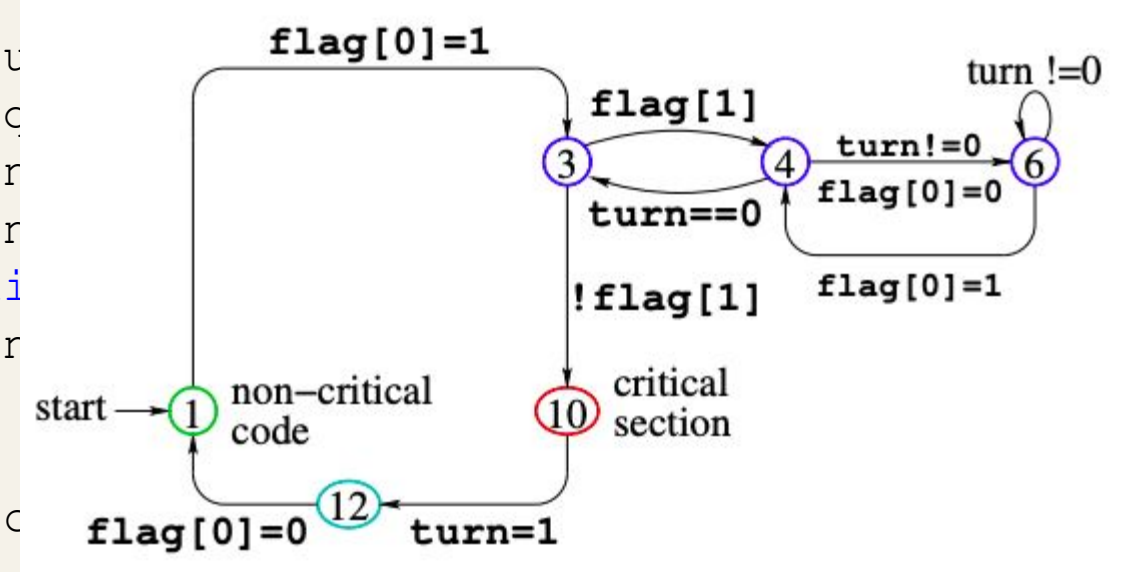


https://godbolt.org/z/1a9ffv1W8
https://github.com/Renjuju/Dekkers-Algorithm/blob/master/dekkers.c

Dekker's algorithm

```
void* p() {
    wantp = tru
    while(wantq
        if(turn
            war
            whi
            war
        }
    }
    criticalSec
    turn = 2;
    wantp = false;
}
```

```
void* q() {
                    1) {
                     false;
                    urn != 2) {}
                     true;
                    ();
    turn = 1;
    wantq = false;
}
```



https://godbolt.org/z/1a9ffv1W8
https://github.com/Renjuju/Dekkers-Algorithm/blob/master/dekkers.c

Dekker's algorithm

```
void* p() {
    wantp = true;
    while(wantq) {
        if(turn == 2) {
            wantp = false;
            while(turn != 1){}
            wantp = true;
        }
    }
    criticalSection();
    turn = 2;
    wantp = false;
}
```

```
void* q() {
    wantq = true;
    while(wantp) {
        if(turn == 1) {
            wantq = false;
            while(turn != 2) {}
            wantq = true;
        }
    }
    criticalSection();
    turn = 1;
    wantq = false;
}
```

https://godbolt.org/z/1a9ffv1W8
https://github.com/Renjuju/Dekkers-Algorithm/blob/master/dekkers.c

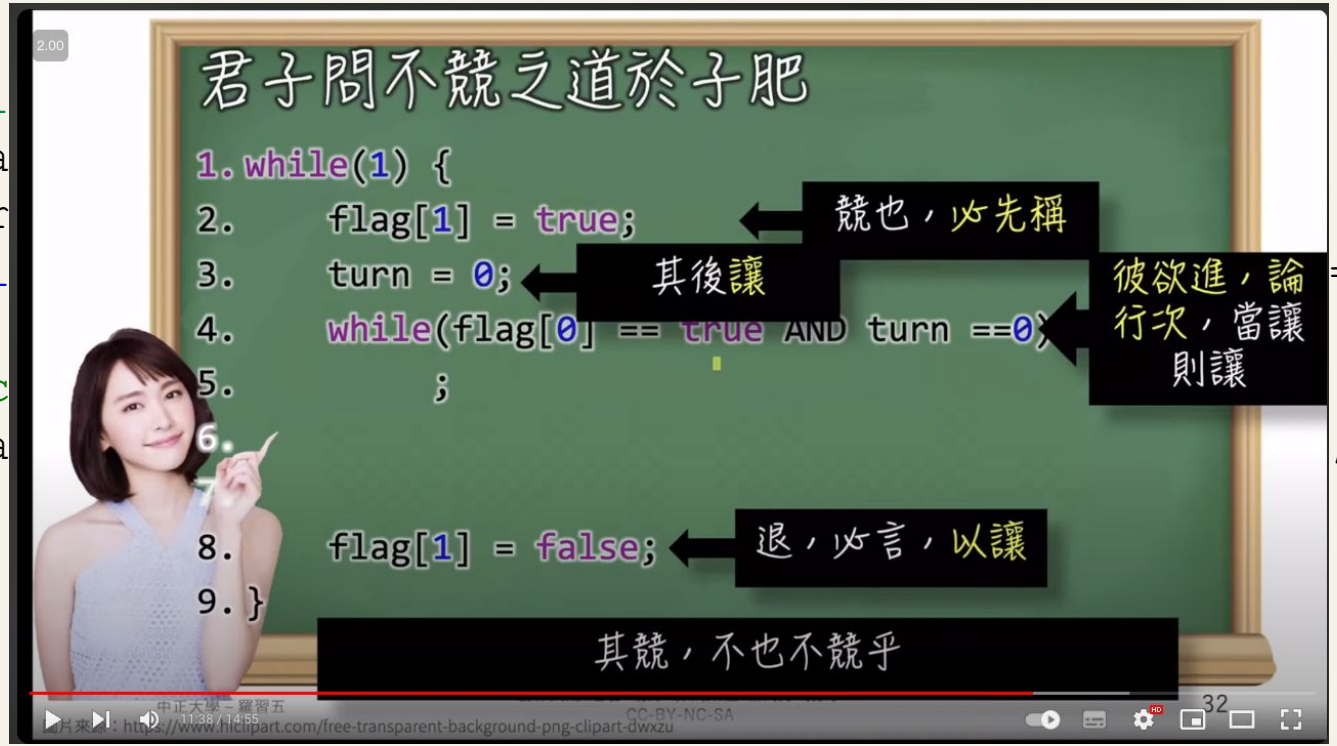
Peterson's algorithm

```
void* p() {
    while(1) {
        flag[0] = true;
        turn = 1;
        while(flag[1] == true
              AND turn == 1);
        //cs
        flag[0] = false;
    }
}
```

```
void* q() {
    while(1) {
        flag[1] = true;
        turn = 0;
        while(flag[0] == true
              AND turn == 0);
        //cs
        flag[1] = false;
    }
}
```

Peterson's algorithm

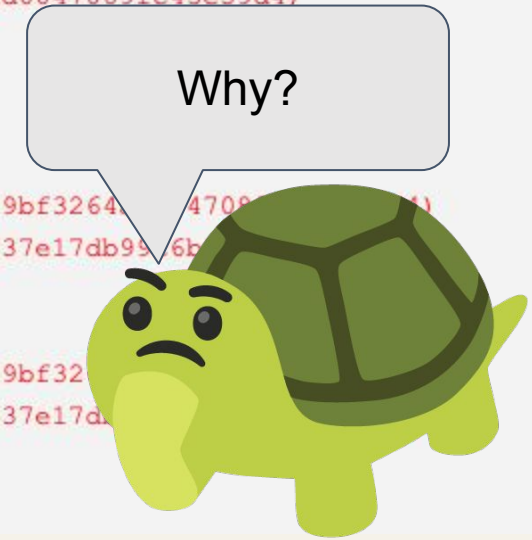
```
void* p()
    while(1
        fla
        tur
        whi                                  = true
                                             == 0);
        //c
        fla                                  ;
    }
}
```



https://www.youtube.com/watch?v=Nqk6yz1Mhdo

```
Program returned: 66
  ==================
  WARNING: ThreadSanitizer: data race (pid=1)
    Write of size 4 at 0x0000004040b8 by thread T2:
      #0 q /app/example.c:38 (output.s+0x40132a) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #1 addNumbers /app/example.c:59 (output.s+0x40145b) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #2 <null> <null> (libtsan.so.2+0x3b329) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)

    Previous read of size 4 at 0x0000004040b8 by thread T1:
      #0 p /app/example.c:22 (output.s+0x4012cc) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #1 addNumbers /app/example.c:57 (output.s+0x40142b) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #2 <null> <null> (libtsan.so.2+0x3b329) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)

    Location is global 'wantq' of size 4 at 0x0000004040b8 (output.s+0x4040b8)

    Thread T2 (tid=4, running) created by main thread at:
      #0 pthread_create <null> (libtsan.so.2+0x40cd6) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)
      #1 main /app/example.c:75 (output.s+0x40157d) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)

    Thread T1 (tid=3, running) created by main thread at:
      #0 pthread_create <null> (libtsan.so.2+0x40cd6) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)
      #1 main /app/example.c:74 (output.s+0x401562) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)

  SUMMARY: ThreadSanitizer: data race /app/example.c:38 in q
```

```
Program returned: 66
  ==================
  WARNING: ThreadSanitizer: data race (pid=1)
    Write of size 4 at 0x0000004040b8 by thread T2:
      #0 q /app/example.c:38 (output.s+0x40132a) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #1 addNumbers /app/example.c:59 (output.s+0x40145b) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #2 <null> <null> (libtsan.so.2+0x3b329) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)

    Previous read of size 4 at 0x0000004040b8 by thread T1:
      #0 p /app/example.c:22 (output.s+0x4012cc) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #1 addNumbers /app/example.c:57 (output.s+0x40142b) (BuildId: 3b063e37678775d0637e17db99c6b9ab3de91d22)
      #2 <null> <null> (libtsan.so.2+0x3b329) (BuildId: dd14181575b14129bf3264ad0047089fc45e39d4)

    Location is global 'wantq' of size 4 at 0x0000004040b8 (output.s+0x4040b8)

    Thread T2 (tid=4, running) created by main thread at:
      #0 pthread_create <null> (libtsan.so.2+0x40cd6) (BuildId: dd14181575b14129bf3264
      #1 main /app/example.c:75 (output.s+0x40157d) (BuildId: 3b063e37678775d0637e17db9

    Thread T1 (tid=3, running) created by main thread at:
      #0 pthread_create <null> (libtsan.so.2+0x40cd6) (BuildId: dd14181575b14129bf32
      #1 main /app/example.c:74 (output.s+0x401562) (BuildId: 3b063e37678775d0637e17d

  SUMMARY: ThreadSanitizer: data race /app/example.c:38 in q
```



Assumption:

關於Peterson’s solution

- G. L. Peterson於1981年提出一個完全解決race condition的「純軟體」演算法
- 這個演算法假設只有P0、P1二個task
- 對於硬體有一些重要的基本：假設「load」和「store」是atomic operations、共享記憶體的系統
- Peterson’s solution的概念可以擴展到任意多的task
- G. L. Peterson提出這個演算法後，隨後就有許多相似觀念的演算法

[第一個『純軟體』的解法] https://en.wikipedia.org/wiki/Dekker%27s_algorithm

創作共用-姓名－標示-非商業性-相同方式分享
CC-BY-NC-SA

Unfortunately

TEAMT5 杜浦數位安全

關於Peterson's solution

- G. L. Peterson於1981年提出一個完全解決race condition的「純軟體」演算法
- 這個演算法假設只有P0、P1二個task
- 對於硬體有一些重要的基本：假設「load」和「store」是atomic operations、共享記憶體的系統
- Peterson's solution的概念可以擴展到任意多的task
- G. L. Peterson提出這個演算法後，隨後就有許多相似觀念的演算法

[第一個『純軟體』的解法] https://en.wikipedia.org/wiki/Dekker%27s_algorithm

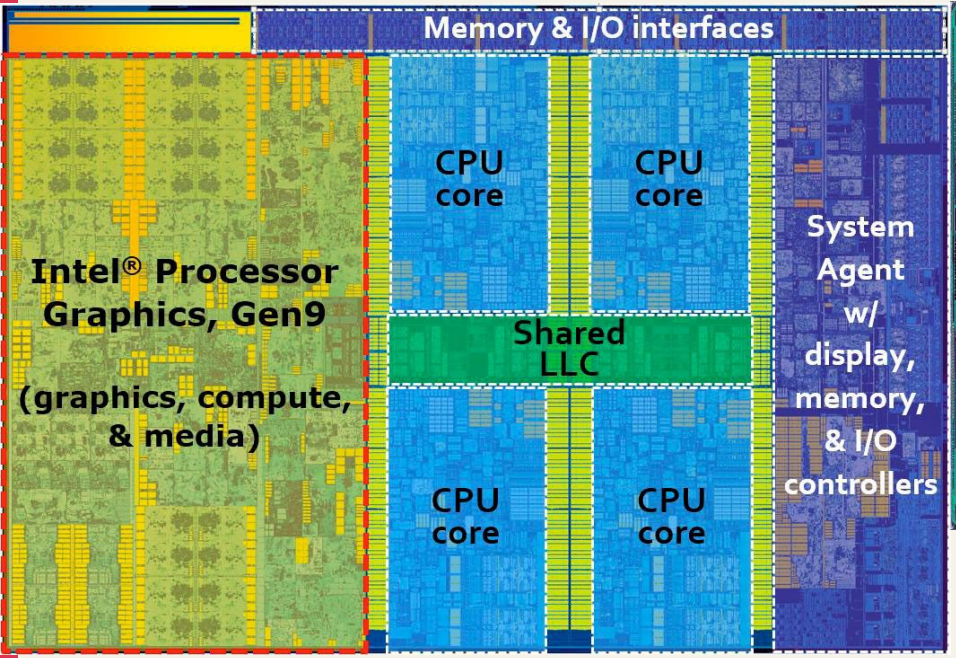
創作共用-姓名－標示-非商業性-相同方式分享
CC-BY-NC-SA

If we have atomic…

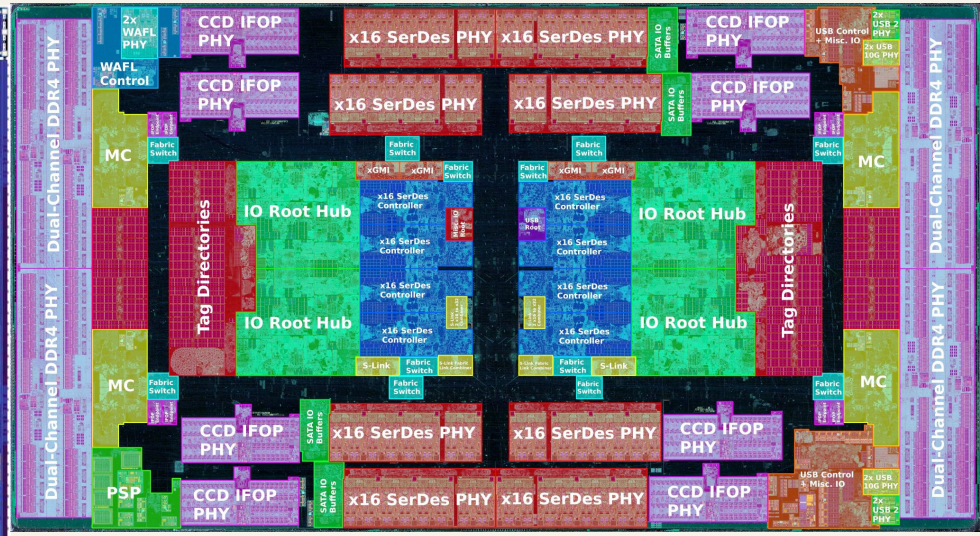
```
std::atomic<bool> lock(false); // holds true when locked
                               // holds false when unlocked
int new_line{1};
void f(int n) {
    while (std::atomic_exchange(&lock, true))
        ; // spin until acquired
    std::cout << n << (new_line++ % 80 ? "" : "\n");
    std::atomic_store(&lock, false);
}
```

https://godbolt.org/z/rKPTs7sMv

UMA and NUMA



Intel Core i7-6700K



AMD EPYC™ 7002

https://www.techpowerup.com/215333/intel-skylake-die-layout-detailed?cp=3
https://www.techpowerup.com/266287/amd-matisse-and-rome-io-controller-dies-mapped-out

Compiler Barrier

Compiler Barriers

```
#define barrier() __asm__ __volatile__("": : :"memory")
_ReadWriteBarrier() // MSVC
```

https://learn.microsoft.com/en-us/cpp/intrinsics/readwritebarrier?view=msvc-170

Compiler Barriers

TEAMT5
杜 浦 數 位 安 全

Force the compiler keeping the assembly order.

```
#define barrier() __asm__ __volatile__("": : :"memor
_ReadWriteBarrier() // MSVC
```

https://learn.microsoft.com/en-us/cpp/intrinsics/readwritebarrier?view=msvc-170

What’s output?

```
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    b = 1;
    return a + b;
}
```

What's output?

```c
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    b = 1;
    return a + b;
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR a[rip], eax
        mov     DWORD PTR b[rip], 1
        mov     edx, DWORD PTR a[rip]
        mov     eax, DWORD PTR b[rip]
        add     eax, edx
        pop     rbp
        ret
```

https://godbolt.org/z/3sbv6Yacr

What's output?

```c
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    b = 1;
    return a + b;
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR a[rip], eax
        mov     DWORD PTR b[rip], 1
        mov     edx, DWORD PTR a[rip]
        mov     eax, DWORD PTR b[rip]
        add     eax, edx
        pop     rbp
        ret
```

https://godbolt.org/z/3sbv6Yacr

What's output?

```c
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    b = 1;
    return a + b;
}
```

```asm
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR b[rip], 1
        mov     DWORD PTR a[rip], eax
        add     eax, 1
        add     rsp, 8
        ret
```

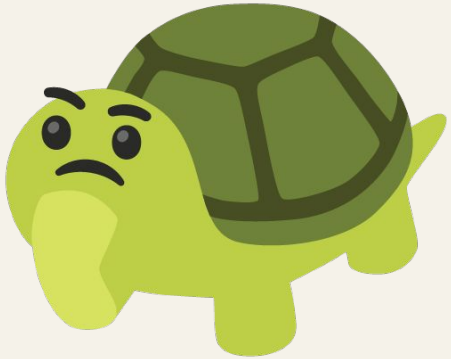
https://godbolt.org/z/3sbv6Yacr

What's output?

```c
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    b = 1;
    return a + b;
}
```

```asm
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR b[rip], 1
        mov     DWORD PTR a[rip], eax
        add     eax, 1
        rsp, 8
```

It's critical if it's a lock() call.



https://godbolt.org/z/3sbv6Yacr

What's output?

```c
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    barrier();
    b = 1;
    return a + b;
}
```

```asm
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR a[rip], eax
        mov     DWORD PTR b[rip], 1
        mov     eax, DWORD PTR a[rip]
        add     rsp, 8
        add     eax, 1
        ret
```

https://godbolt.org/z/soWco6b4c

What's output?

```
#include <stdio.h>
int a, b;
int main() {
    a = puts("");
    barrier();
    b = 1;
    return a + b;
}
```

```
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR a[rip], eax
        mov     DWORD PTR b[rip], 1
        mov     eax, DWORD PTR a[rip]
        add     rsp, 8
        add     eax, 1
        ret
```

```
#define barrier() __asm__ __volatile__("": : :"memory")
_ReadWriteBarrier() // MSVC
```

https://godbolt.org/z/soWco6b4c

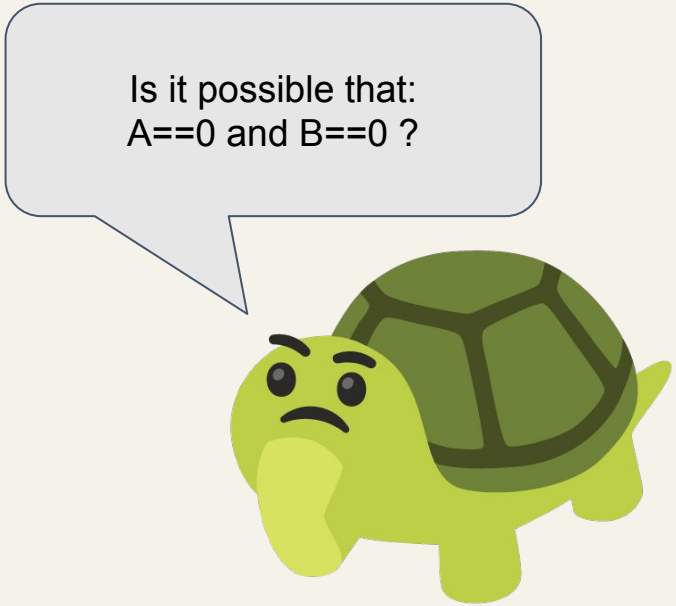
OoOE

thread 0	thread 1

`X = 1;`
`barrier();`
`A = Y;`
	`Y = 1;`
	`barrier();`
	`B = X;`

OoOE

```
static void *worker1(void *arg) {
    X = 1;
    asm volatile("" ::: "memory");
    A = Y;
}

static void *worker2(void *arg) {
    Y = 1;
    asm volatile("" ::: "memory");
    B = X;
}
```



https://godbolt.org/z/sn76dTYM9

OoOE

```
static voi
    X = 1;
    asm vol
    A = Y;
}

static voi
    Y = 1;
    asm vol
    B = X;
}
```

[2/2] /tmp/foo

1: ../LTBT-Android × 2: /tmp/foo × +

```
→ foo cc -Ofast -std=gnu11 -lpthread -Wno-all -o ooOe ooOe.c
→ foo ./ooOe
4 reorder caught in 100000 iterations.
→ foo
```

https://godbolt.org/z/sn76dTYM9

OoOE

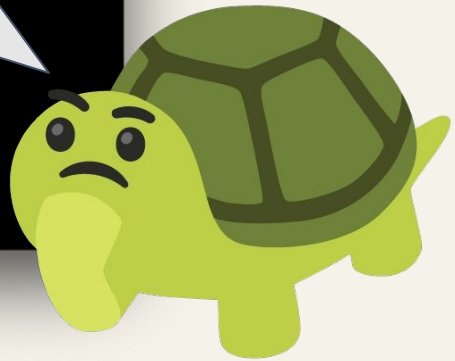
```
static voi
    X = 1;
    asm vol
    A = Y;
}

static voi
    Y = 1;
    asm vol
    B = X;
}
```

[2/2] /tmp/foo

1: ../LTBT-Android × 2: /tmp/foo × +

```
→ foo cc -Ofast -std=gnu11 -lpthread -Wno-all -o ooOe ooOe.c
→ foo ./ooOe
4 reorder caught in 100000 iterations.
→ foo
```

嘿嘿🤪
你以為 asm 順序正確，執行就會正確嗎？
CPU 會幫你 reorder 呦🥵



https://godbolt.org/z/sn76dTYM9

atomic

Atomic

原子筆跟原子沒有關係

沒有「原子操作」，只有「不可分割的操作」

不要「并发」，只有「並行」

https://en.wikipedia.org/wiki/John_Dalton

The six memory orders

TEAMT5
杜 浦 數 位 安 全

```
memory_order_relaxed
memory_order_consume
memory_order_acquire
memory_order_release
memory_order_acq_rel
memory_order_seq_cst
```

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

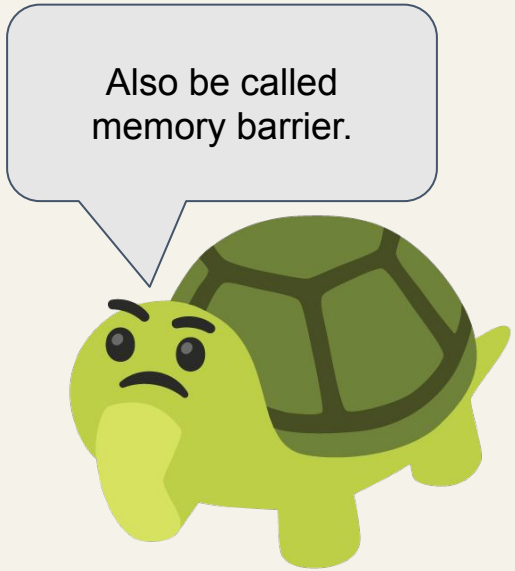
memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**



The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

The order doesn’t matter.
It only should be completed
in an atomic operation.

🌰 memory_order_relaxed

```
#ifdef RELAXED
void accumulate(int n) {
    for (int i = 0; i < n; ++i)
        accumulator.fetch_add(1, std::memory_order_relaxed);
}
#else
void accumulate(int n) {
    for (int i = 0; i < n; ++i)
        accumulator.fetch_add(1, std::memory_order_seq_cst);
}
#endif
```

memory_order_relaxed

```
#ifdef RELAXE
void accumula
    for (int i
        accumu                                  _relaxed);
}
#else
void accumula
    for (int i
        accumu                                  _seq_cst);
}
#endif
```

[2/2] /tmp/foo

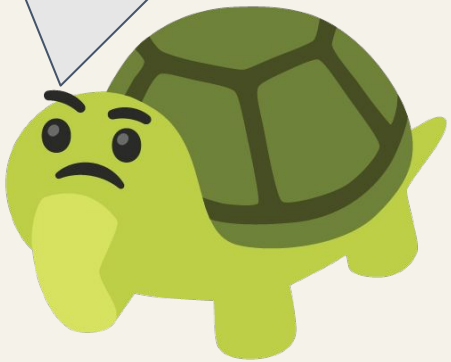
1: ../LTBT-Android 2: /tmp/foo

```
[oh-my-zsh] Would you like to update? [Y/n]
Updating Oh My Zsh
➜  foo c++ -std=c++11 -O3 -DRELAXED main.cpp -o relaxed
➜  foo time ./relaxed
Accumulated value: 20000000
./relaxed  0.27s user 0.00s system 128% cpu 0.210 total
➜  foo c++ -std=c++11 -O3 main.cpp -o seqcst
➜  foo time ./seqcst
Accumulated value: 20000000
./seqcst  0.72s user 0.00s system 122% cpu 0.589 total
➜  foo
```

memory_order_relaxed

```cpp
#include <iostream>
#include <thread>
#include <atomic>

std::atomic<int> accumulator(0);

#ifdef RELAXED
void accumulate(int n) {
   for (int i = 0; i < n; ++i)
       accumulator.fetch_add(1, std::memory_order_relaxed);
}
#else
void accumulate(int n) {
   for (int i = 0; i < n; ++i)
       accumulator.fetch_add(1, std::memory_order_seq_cst);
}
#endif

int main()
{
   constexpr int iterations = 10000000ul;

   // Create two threads that run the accumulate function
   std::thread t1(accumulate, iterations);
   std::thread t2(accumulate, iterations);

   // Wait for both threads to finish
   t1.join();
   t2.join();

   // Output the result
   std::cout << "Accumulated value: " << accumulator.load(std::memory_order_relaxed) << std::endl;

   return 0;
}
```

Source code here.
You should test your code on **ARM**-based machines.



https://preshing.com/20120930/weak-vs-strong-memory-models/

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

Similar with acquire.
Only guarantee the
dependency chain.

memory_order_consume

TEAMT5 杜浦數位安全

```
void producer() {
    std::string* p = new std::string("Hello");
    data = 42;
    ptr.store(p, std::memory_order_release);
}
void consumer() {
    std::string* p2;
    while (!(p2 = ptr.load(std::memory_order_consume)))
        ;
// never fires: *p2 carries dependency from ptr
    assert(*p2 == "Hello");
// may or may not fire: data does not carry dependency from ptr
    assert(data == 42);
}
```

```
std::atomic<std::string*> ptr;

int data;
```

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

Release the lock.
Make everyone observable.

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

For all **store** cannot reorder before the fence.

Release the lock.
Make everyone observable.

The six memory orders

TEAMT5 杜浦數位安全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

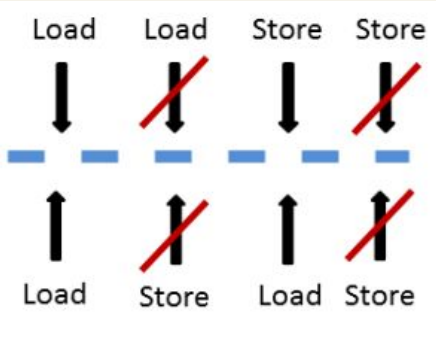
memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

For all **store** cannot reorder before the fence.

Release the lock.
Make everyone observable.



The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

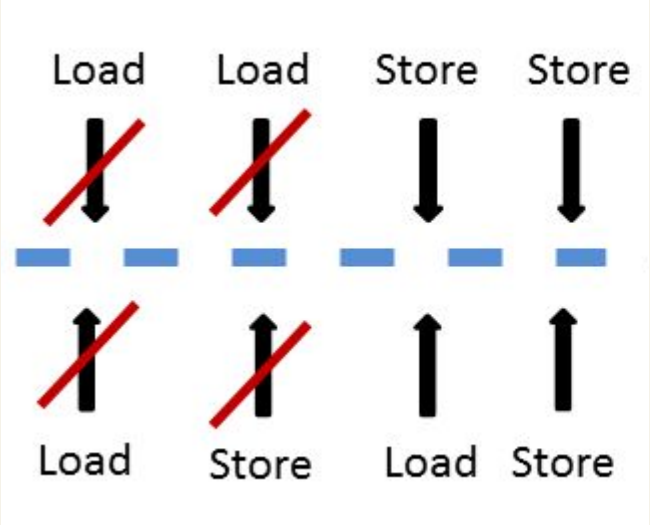
memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

Inversion of release.
So, **no loads** are permitted after the fence.



The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

Acquire + Release

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

All loads not happen after the fence.
All store not happen before the fence.

Acquire + Release

The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

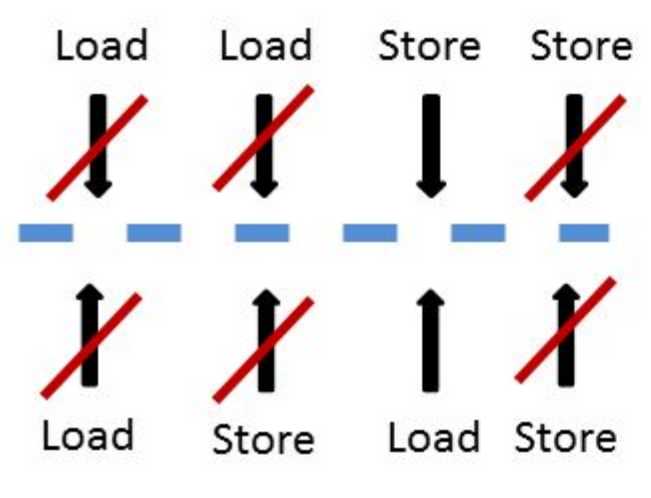
memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

All loads not happen after the fence.
All store not happen before the fence.

Acquire + Releas



The six memory orders

TEAMT5
杜 浦 數 位 安 全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

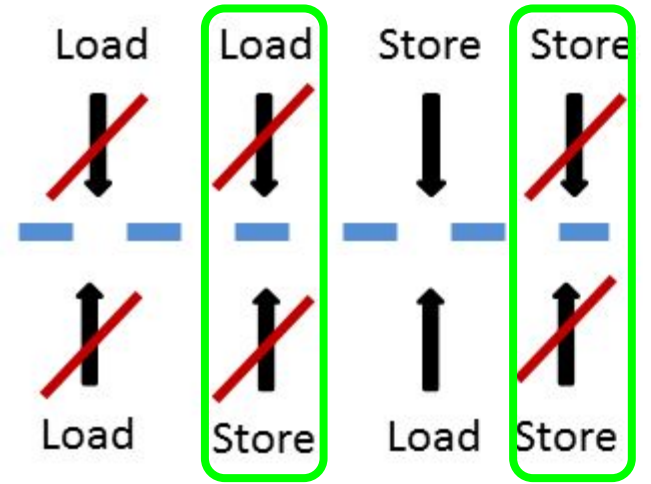
memory_order_**acq_rel**

memory_order_**seq_cst**

All loads not happen after the fence.
All store not happen before the fence.

Acquire + Releas

Release



The six memory orders

TEAMT5 杜浦數位安全

memory_order_**relaxed**

memory_order_**consume**

memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

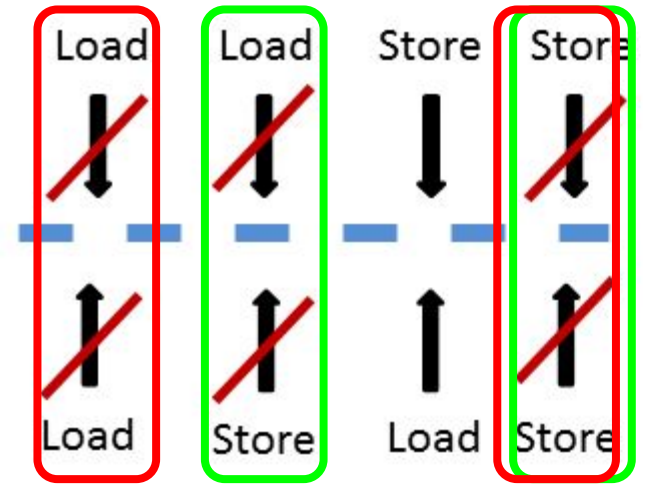
memory_order_**seq_cst**

All loads not happen after the fence.
All store not happen before the fence.

Acquire + Releas

Acquire

Release



The six memory orders

TEAMT5 杜浦數位安全

memory_order_**relaxed**

memory_order_**consume**

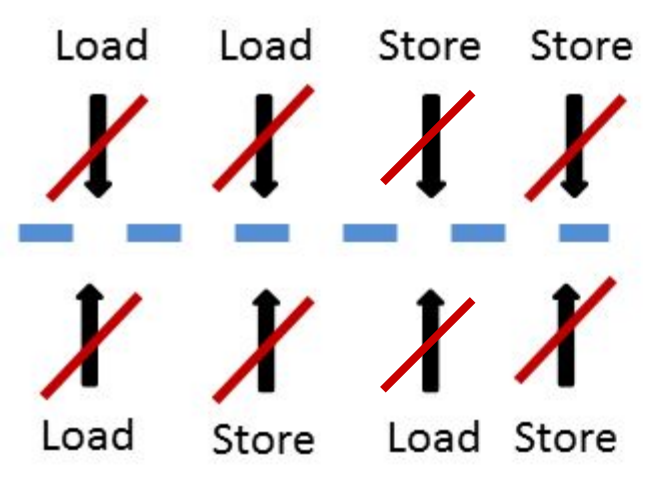
memory_order_**acquire**

memory_order_**release**

memory_order_**acq_rel**

memory_order_**seq_cst**

Any reorder is not permitted.

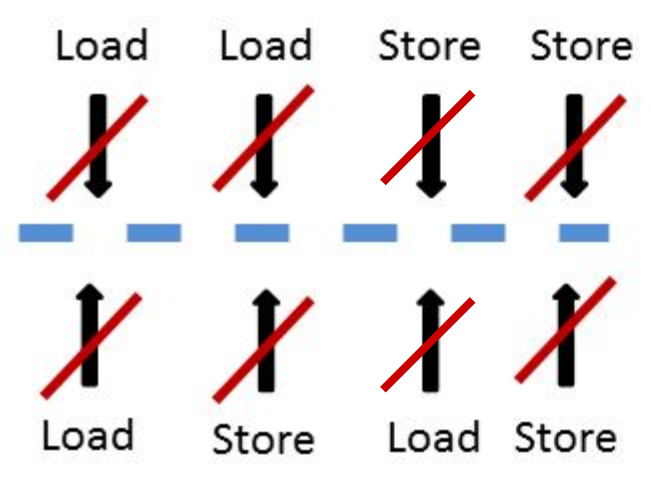


The six memory orders

The MFENCE instruction establishes a memory fence for both loads and stores. The processor ensures that no load or store after MFENCE will become globally visible until all loads and stores before MFENCE are globally visible.[1] Note that the sequences LFENCE;SFENCE and SFENCE;LFENCE are not equivalent to MFENCE because neither ensures that older stores are globally observed prior to younger loads.

```
memory_order_consume
memory_order_acquire
memory_order_release
memory_order_acq_rel
memory_order_seq_cst
```

Any reorder is not permitted.



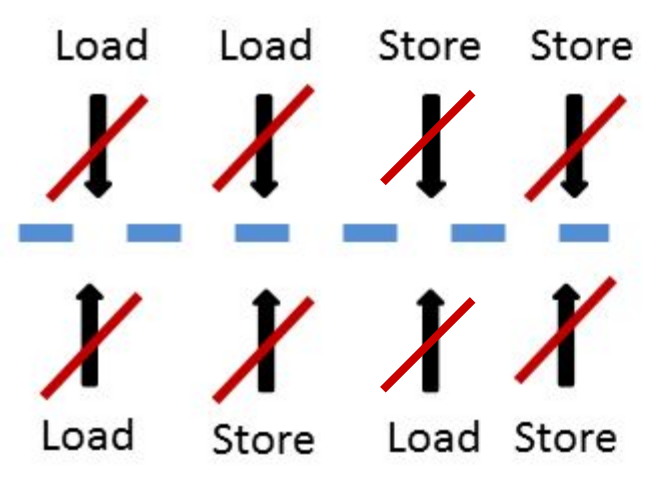
The six memory orders

The MFENCE instruction establishes a memory fence for both loads and stores. The processor ensures that no load or store after MFENCE will become globally visible until all loads and stores before MFENCE are globally visible.[1] Note that the sequences LFENCE;SFENCE and SFENCE;LFENCE are not equivalent to MFENCE because neither ensures that older stores are globally observed prior to younger loads.

One of many approach is:
Flush to L3 cache.

Any reorder is not permitted.

```
mem rder_acq_rel
me _seq_cst
```



Memory order in ASM

TEAMT5 杜浦數位安全

x86

- LFENCE (load fence) -> acquire
- SFENSE (store fence) -> release
- MFENCE (memory fence) -> sequence consistence

arm

- DSB (Data Synchronization Barrier)
- STLEX/LDAEX Load-Acquire and Store-Release Register Exclusive.
- STREX/LDREX Load and Store Register Exclusive.

Cache line - LFENCE

TEAMT5 杜浦數位安全

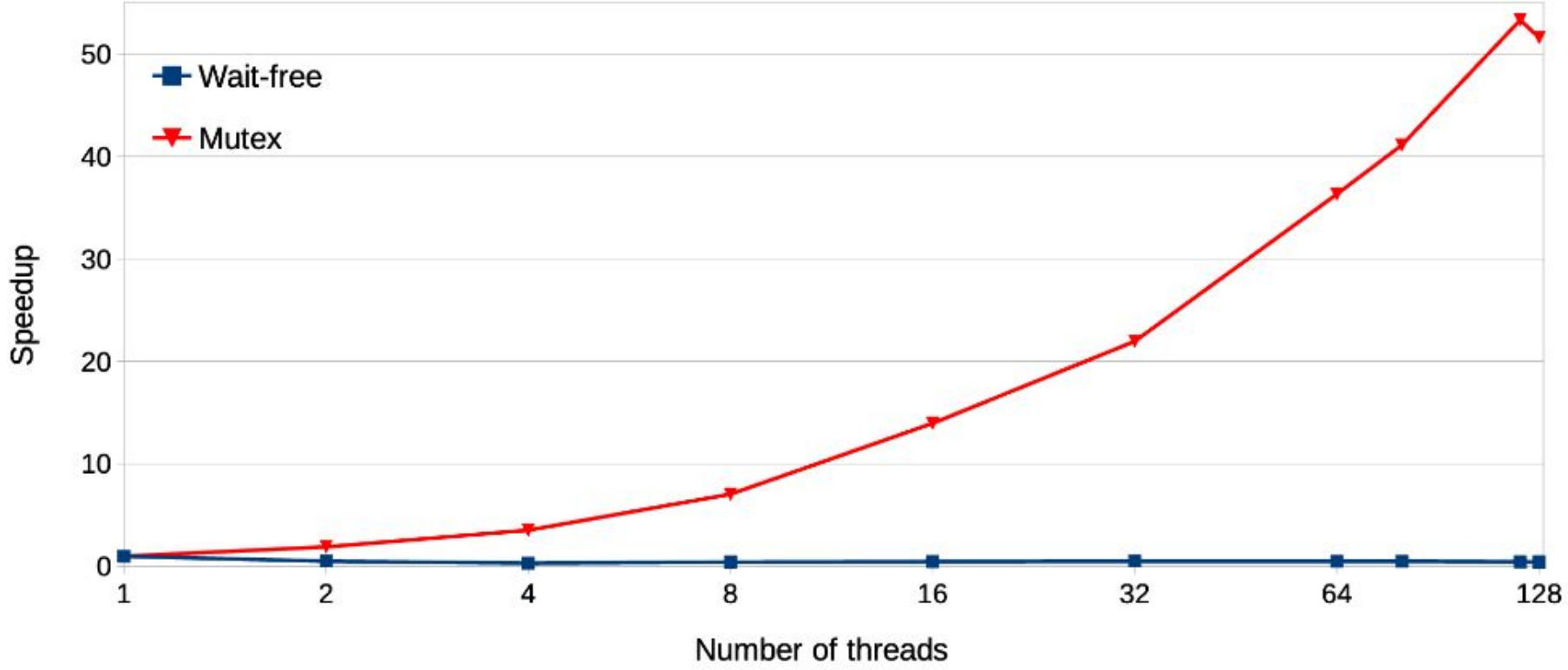
Performs a serializing operation on all load-from-memory instructions that were issued prior the LFENCE instruc- tion. Specifically, LFENCE does not execute until all prior instructions have completed locally, and no later instruc- tion begins execution until LFENCE completes. In particular, an instruction that loads from memory and that precedes an LFENCE receives data from memory prior to completion of the LFENCE. (An LFENCE that follows an instruction that stores to memory ~~might complete **before** the data~~ being stored have become globally visible.) In~~structions following~~ an LFENCE may be fetched from memory before the LFENCE, but they will not execute (even speculatively) until the LFENCE completes.

Intel® 64 and IA-32 Architectures Software Developer's Manual

Is atomic faster than mutex?

No, it depends.

Lock-free means “fast”



Mentor

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Lock-free means “fast”

```
std::atomic<unsigned long> sum;
```

- Program A:

```
void do_work(size_t N, unsigned long* a) {
  for (size_t i = 0; i < N; ++i) sum += a[i];
}
```

- Program B:

```
unsigned long sum(0); std::mutex M;
void do_work(size_t N, unsigned long* a) {
  unsigned long s = 0;
  for (size_t i = 0; i < N; ++i) s += a[i];
  std::lock_guard<std::mutex> L(M); sum += s;
}
```


Mentor A Siemens Business

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Lock-free means “fast”

```
std::atomic<unsigned long> sum;
```

- Program A:

```
void do_work(size_t N, unsigned long* a) {
  for (size_t i = 0; i < N; ++i) sum += a[i];
}
```

- Program B:

```
unsigned long sum(0); std::mutex M;
void do_work(size_t N, unsigned long* a) {
  unsigned long s = 0;
  for (size_t i = 0; i < N; ++i) s += a[i];
  std::lock_guard<std::mutex> L(M); sum += s;
}
```


Mentor A Siemens Business

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Lock-free means "fast"

```
std::atomic<unsigned long> sum;
```

- Program A:

```
void do_work(size_t N, unsigned long* a) {
  for (size_t i = 0; i < N; ++i) sum += a[i];
}
```

- Program B:

```
unsigned long sum(0); std::mutex M;
void do_work(size_t N, unsigned long* a) {
  unsigned long s = 0;
  for (size_t i = 0; i < N; ++i) s += a[i];
  std::lock_guard<std::mutex> L(M); sum += s;
}
```

Acquire

Release

128 thread:
128 Acquire
128 Release

Mentor A Siemens Business

CppCon 2017: Fedor Pikus "C++ atomics, from basic to advanced. What do they really do?"

Lock-free means “fast”

```
std::atomic<unsigned long> sum;
```

- Program A:

```
void do_work(size_t N, unsigned long* a) {
  for (size_t i = 0; i < N; ++i) sum += a[i];
}
```

Default sequence consistence

- Program B:

```
unsigned long sum(0); std::mutex M;
void do_work(size_t N, unsigned long* a) {
  unsigned long s = 0;
  for (size_t i = 0; i < N; ++i) s += a[i];
  std::lock_guard<std::mutex> L(M); sum += s;
}
```


Mentor A Siemens Business

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Lock-free means “fast”

std::atomic<unsigned long>

- Program A:

```
void do_work(size_t N, unsigned long* a) {
  for (size_t i = 0; i < N; ++i) sum += a[i];
}
```

- Program B:

```
unsigned long sum(0); std::mutex M;
void do_work(size_t N, unsigned long* a) {
  unsigned long s = 0;
  for (size_t i = 0; i < N; ++i) s += a[i];
  std::lock_guard<std::mutex> L(M); sum += s;
}
```

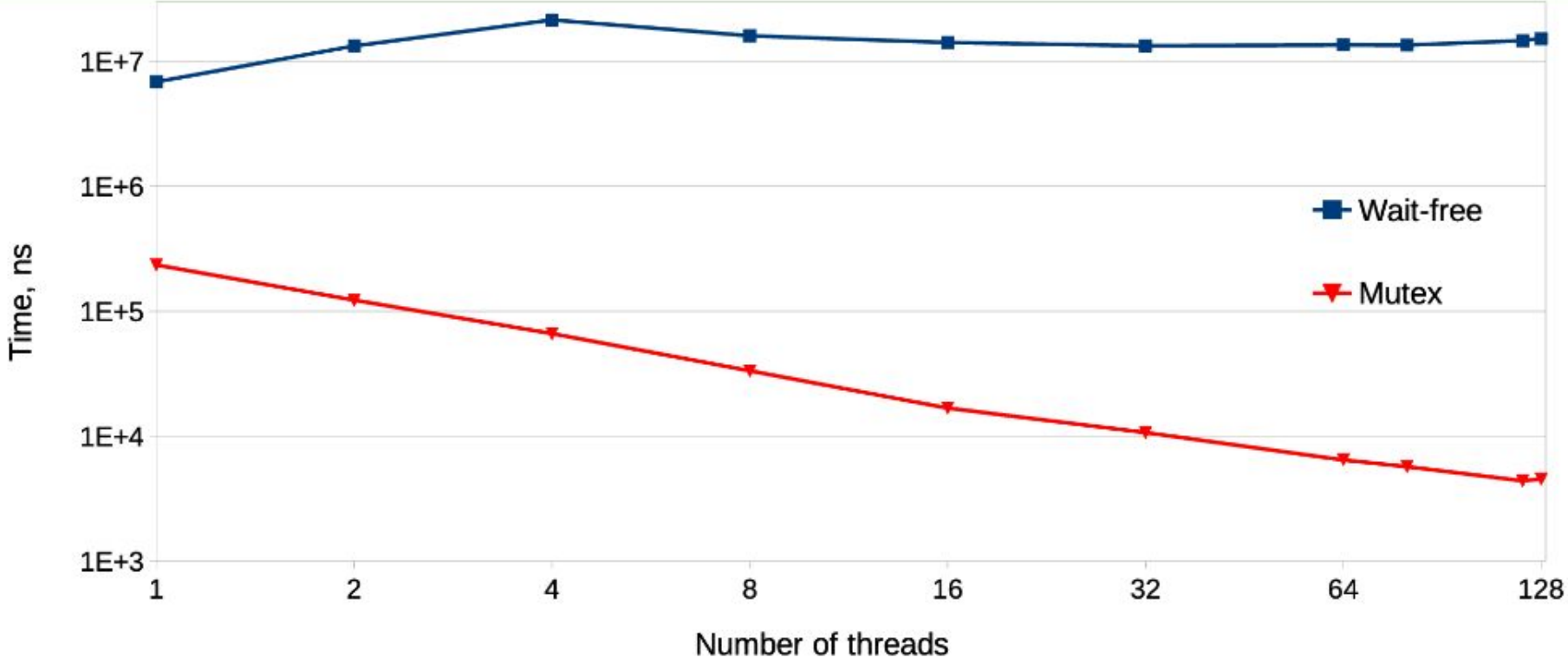
128 thread:
128**N** **seq_cst**

Default sequence consistence

Mentor A Siemens Business

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Is lock-free faster?



Mentor A Siemens Business

CppCon 2017: Fedor Pikus “C++ atomics, from basic to advanced. What do they really do?”

Try

TEAMT5 杜浦數位安全

[2/2] /tmp/foo

1: ../LTBT-Android × 2: /tmp/foo ×

```
→ foo c++ -std=c++11 -O3 atomic_cppcon17.cpp -o atomic_cppcon17; time ./atomic_cppcon17
sum = 49999995000000
./atomic_cppcon17  1.66s user 0.02s system 569% cpu 0.295 total
→ foo
```

relaxed

[2/2] /tmp/foo

1: ../LTBT-Android × 2: /tmp/foo ×

```
→ foo c++ -std=c++11 -O3 atomic_cppcon17.cpp -o atomic_cppcon17; time ./atomic_cppcon17
sum = 49999995000000
./atomic_cppcon17  3.04s user 0.02s system 651% cpu 0.469 total
→ foo
```

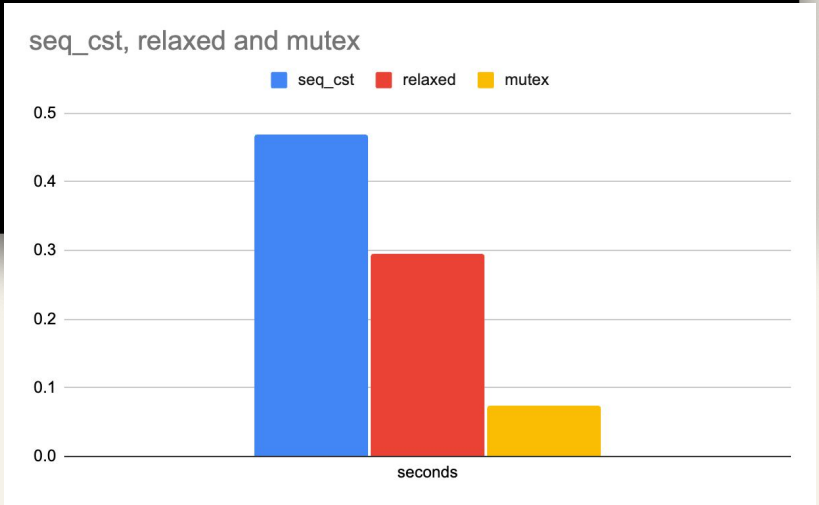
seq_cst

[2/2] /tmp/foo

1: ../LTBT-Android × 2: /tmp/foo ×

```
→ foo c++ -DMUTEX -std=c++11 -O3 atomic_cppcon17.cpp -o atomic_cppcon17; time ./atomic_cppcon17
sum = 49999995000000
./atomic_cppcon17  0.01s user 0.01s system 31% cpu 0.074 total
→ foo
```

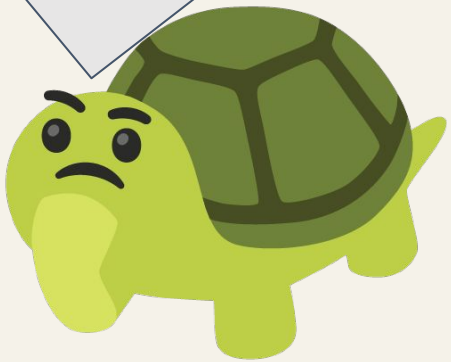
mutex



https://godbolt.org/z/d8z35Ee6s

ABA problem

The issue of ABA in the context of Compare-And-Swap (CAS) operations is a **fundamental** problem encountered by beginners.

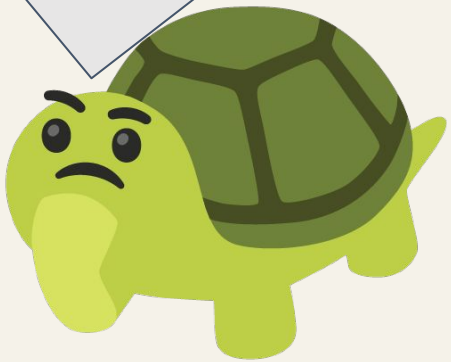


ABA problem

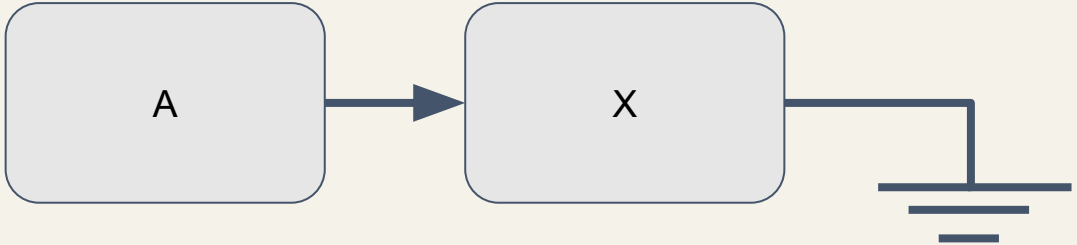
TEAMT5
杜 浦 數 位 安 全

The issue of ABA in the context of Compare-And-Swap (CAS) operations is a **fundamental** problem encountered by beginners.

One of the workaround is our hazard pointer.

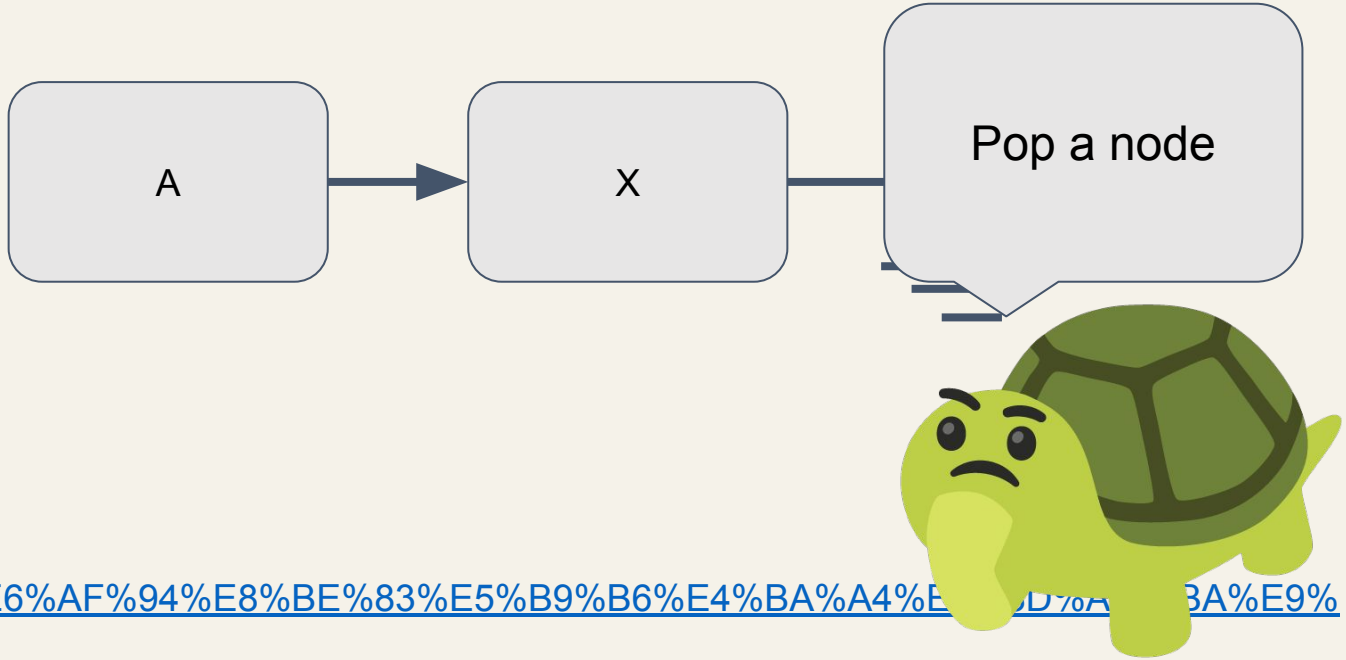


ABA problem



https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E6%8D%A2#ABA%E9%97%AE%E9%A2%98

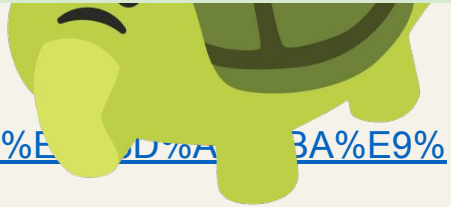
ABA problem



https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E[illegible]D%A[illegible]3A%E9%97%AE%E9%A2%98

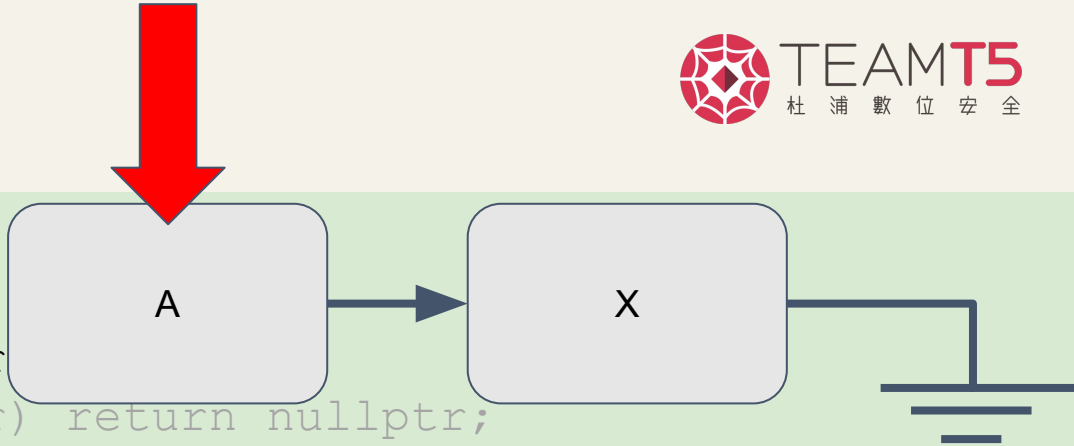
ABA problem

```
Obj* Pop() {
    while (1) {
        Obj* ret_ptr = top_ptr;
        if (ret_ptr == nullptr) return nullptr;
        Obj* next_ptr = ret_ptr->next;
        if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
            return ret_ptr;
        }
    }
}
```

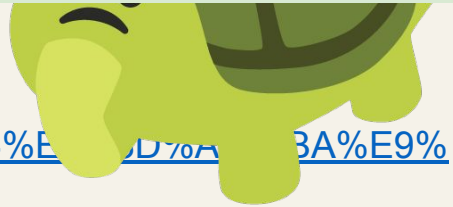


https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E[illegible]D%A[illegible]3A%E9%97%AE%E9%A2%98

ABA problem

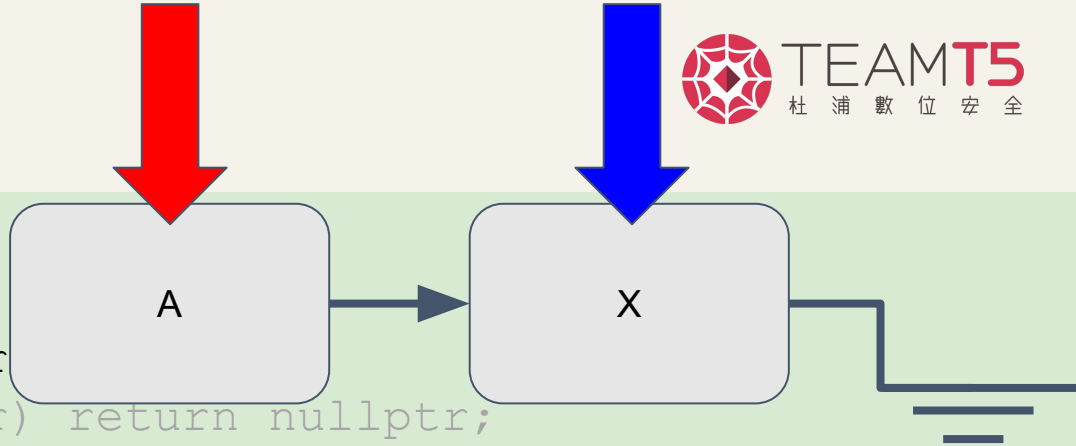



```
Obj* Pop() {
    while (1) {
        Obj* ret_ptr = top_ptr
        if (ret_ptr == nullptr) return nullptr;
        Obj* next_ptr = ret_ptr->next;
        if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
            return ret_ptr;
        }
    }
}
```

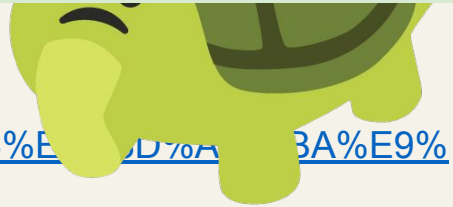


https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E[illegible]D%A[illegible]3A%E9%97%AE%E9%A2%98

ABA problem

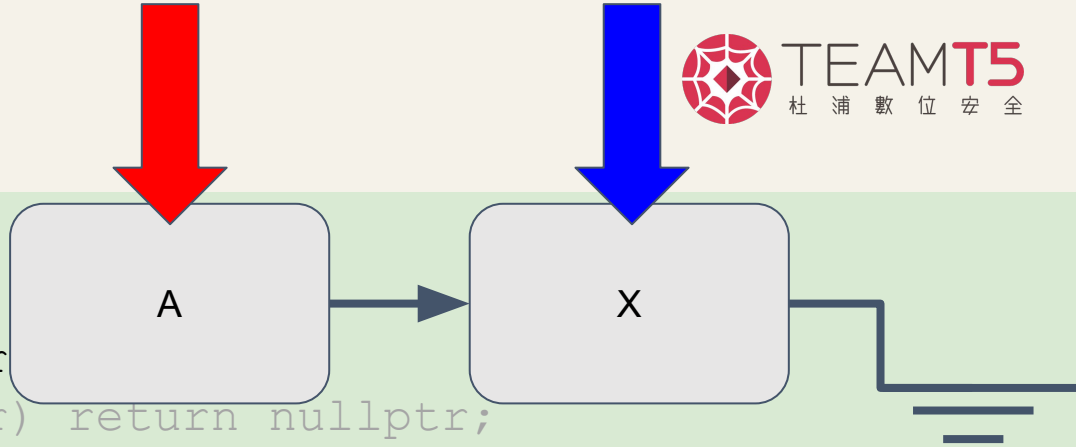



```
Obj* Pop() {
    while (1) {
        Obj* ret_ptr = top_ptr
        if (ret_ptr == nullptr) return nullptr;
        Obj* next_ptr = ret_ptr->next;
        if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
            return ret_ptr;
        }
    }
}
```



https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E[illegible]D%A[illegible]3A%E9%97%AE%E9%A2%98

ABA problem




```
Obj* Pop() {
    while (1) {
        Obj* ret_ptr = top_ptr
        if (ret_ptr == nullptr) return nullptr;
        Obj* next_ptr = ret_ptr->next;
        if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
            return ret_ptr;
        }
    }
}
```

If **top_ptr** == **ret_ptr**
Then **top_ptr** = **next_ptr**

https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E[illegible]%A[illegible]3A%E9%97%AE%E9%A2%98

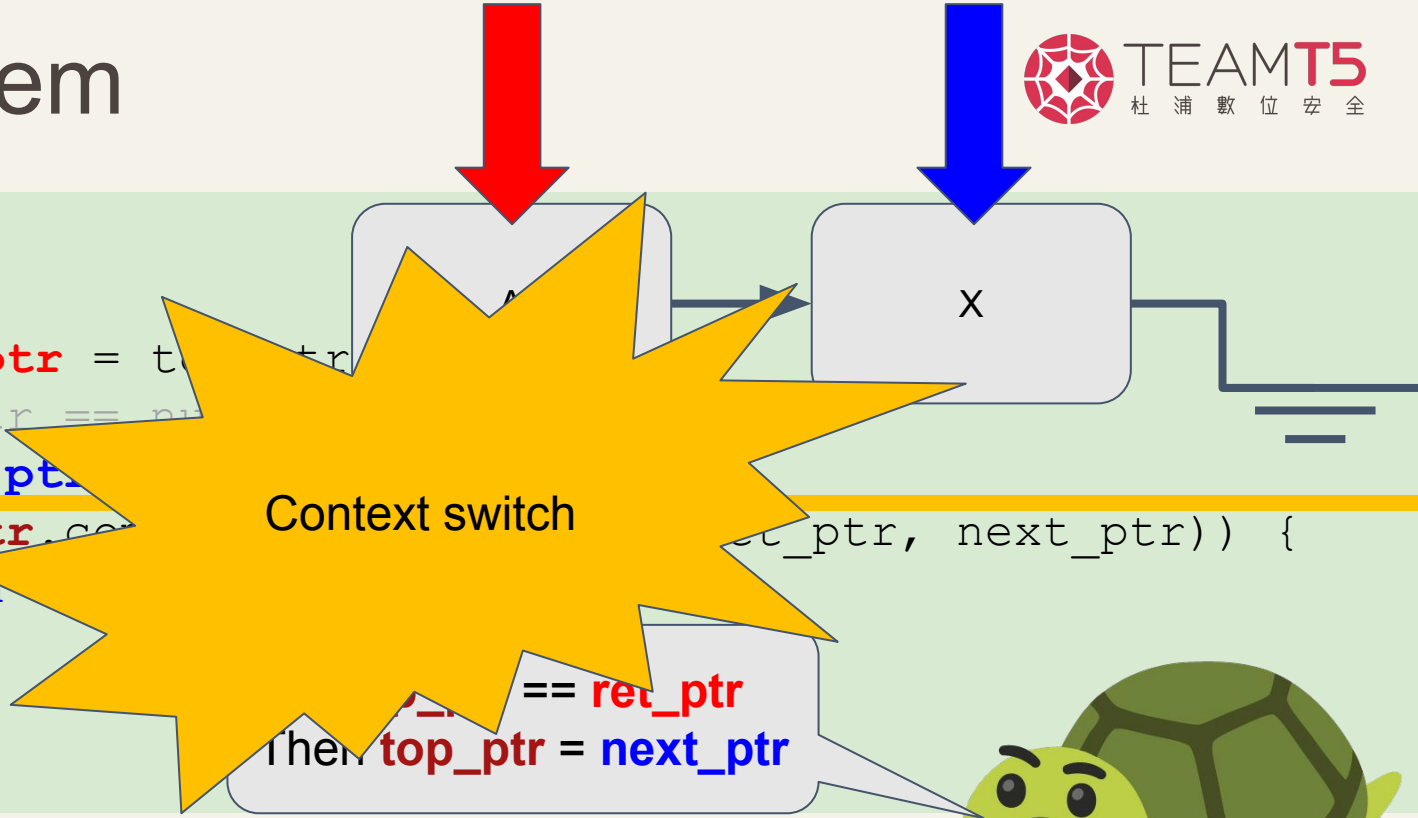
AB

```
Obj
    _pt
    if (top_pt                    ret_ptr, next_ptr)) {
        retur
    }
}
}
```

https://zh.wikipedia.org BE%83%E5%B9%B6%E4%BA%A4%E D% 3A%E9%97%AE%E9%A2%98

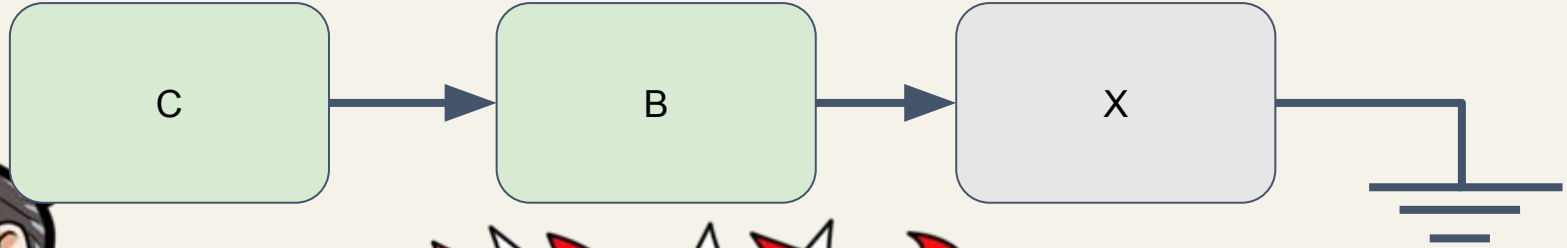
ABA problem

```
Obj* Pop() {
    while (1) {
        Obj* ret_ptr = t
        if (ret_ptr ==
        Obj* next pt
        if (top_ptr.co                    et_ptr, next_ptr)) {
            return
        }
    }
}
```



https://zh.wikipedia.org/zh-tw/%E6%AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E
D%A
3A%E9%97%AE%E9%A2%98

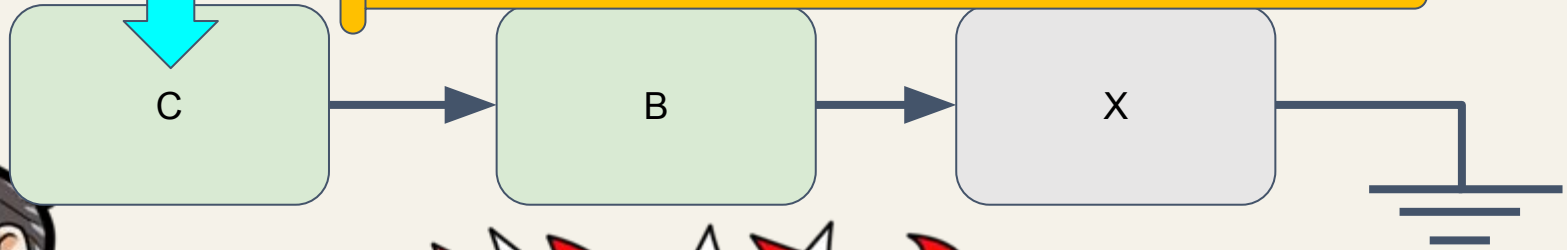
ABA problem



h[illegible]AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E6%8D%A2#ABA%E9%97[illegible]

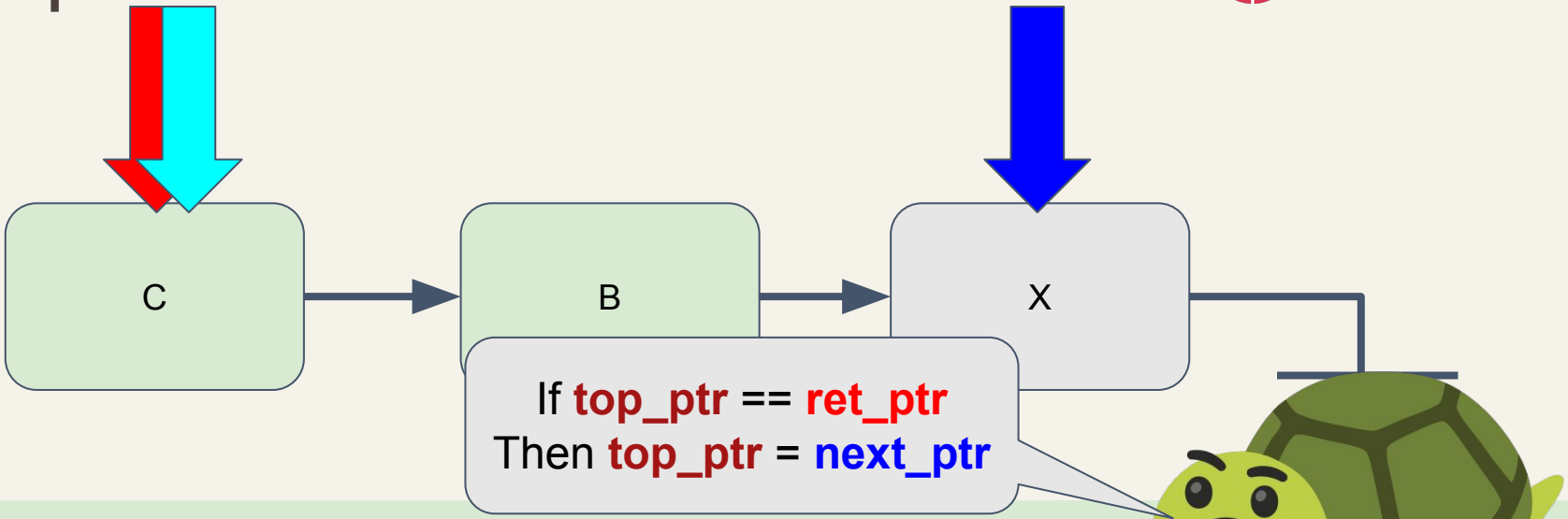
ABA problem

Memory arena: **Reuse** from the free list.



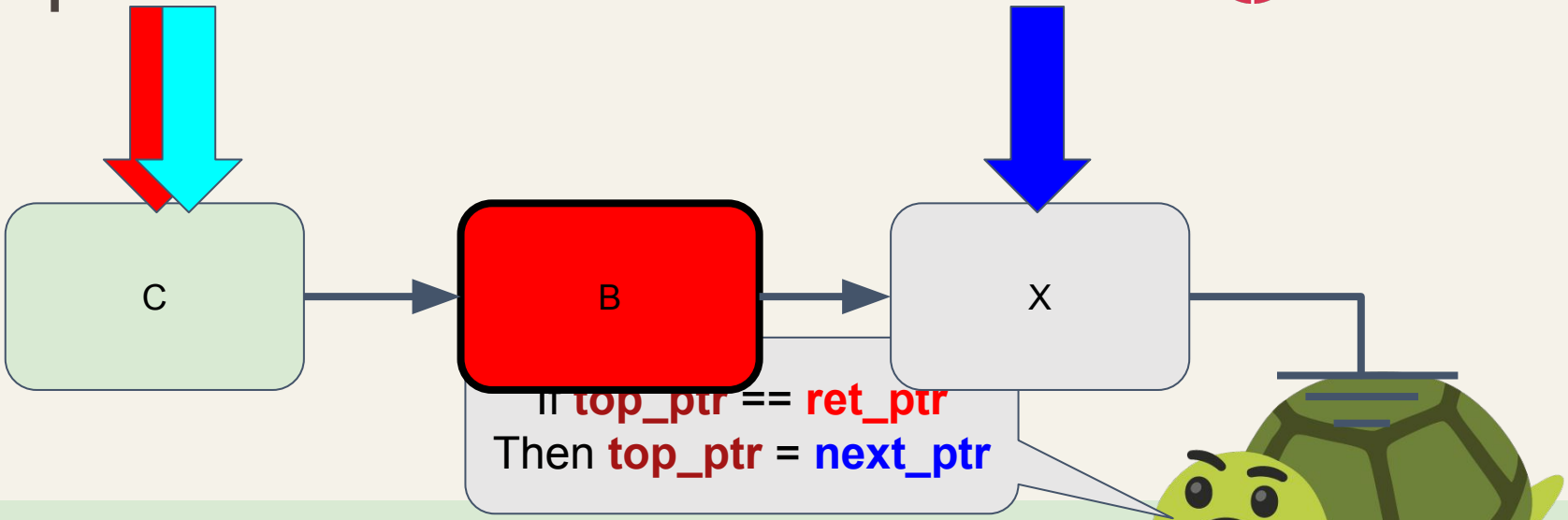
h[illegible]AF%94%E8%BE%83%E5%B9%B6%E4%BA%A4%E6%8D%A2#ABA%E9%97[illegible]%[illegible]

ABA problem




```
Obj* Pop() {
   while (1) {
       Obj* ret_ptr = top_ptr;
       if (ret_ptr == nullptr) return nullptr;
       Obj* next_ptr = ret_ptr->next;
           if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
          return ret_ptr;
       }
   }
}
```

ABA problem




```
Obj* Pop() {
   while (1) {
       Obj* ret_ptr = top_ptr;
       if (ret_ptr == nullptr) return nullptr;
       Obj* next_ptr = ret_ptr->next;
           if (top_ptr.compare_exchange_weak(ret_ptr, next_ptr)) {
          return ret_ptr;
       }
   }
}
```

ABA problem

TEAMT5
杜 浦 數 位 安 全

Solutions:

- Locks
- Tags (another metadata)
- Hazard pointer
- RCU

Use cases

Hazard pointer

TEAMT5
杜 浦 數 位 安 全

C++26

Basically reference counting

Extremely WRRM (Write-Rare-Read-Many)

- Typically, single writer.

Hazard pointer

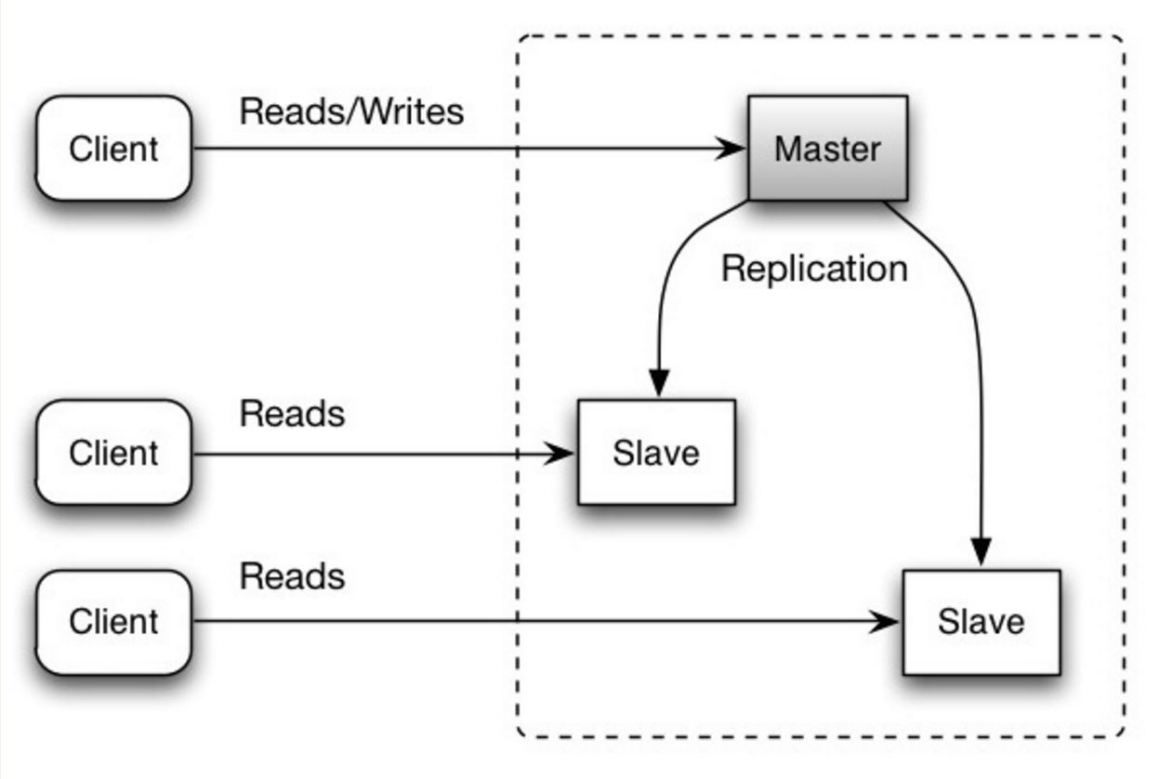
C++26

Basically reference counting

Extremely WRRM (Write-Rare-Read-Many)

- Typically, single writer.

Hazard pointer



Similar with
Master-Slave replication

https://github.com/donnemartin/system-design-primer?tab=readme-ov-file#master-master-replication

Hazard pointer (folly)

```
void push(T val) {
    auto node = new
Node(val, head());

    while
(!cas_head(node->next_,
node)) {/*try again*/;}
}
```

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)

```
void push(T val) {
    auto node = new
Node(val, head());

    while
(!cas_head(node->next_,
node)) {/*try again*/;}
}
```

Almost single writer.

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)

```
void push(T val) {
    auto node = new
Node(val, head());


    while
(!cas_head(node->next_,
node)) {/*try again*/;}
}
```

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

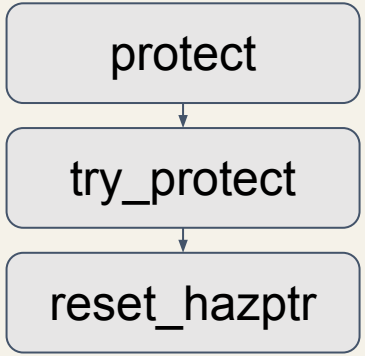
Retire to read-only.

Hazard pointer (folly)

```
void push(T val) {
    auto node = new
Node(val, head());

    while
(!cas_head(node->next_,
node)) {/*try again*/;}
}
```

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

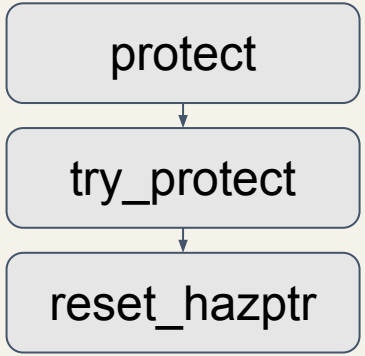
Hazard pointer




```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
```

```
FOLLY_ALWAYS_INLINE void reset_hazptr(const void* p = nullptr)
noexcept {
   hazptr_.store(p, std::memory_order_release);
 }
```

Hazard pointer

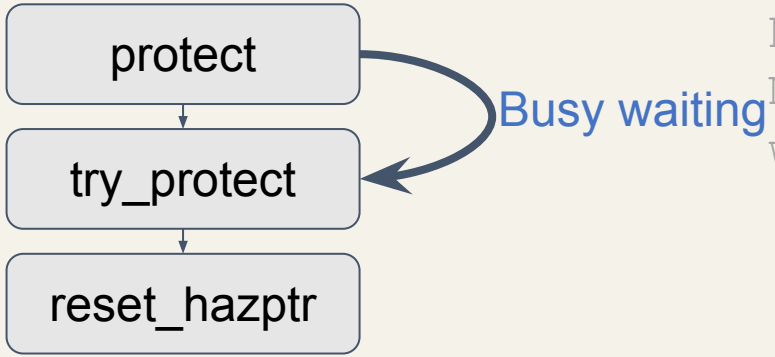



```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node
            retu
        auto nex
        if (cas_
```

Copy the pointer to protect zone

```
FOLLY_ALWAYS_INLINE void reset_hazptr(const void* p =
noexcept {
    hazptr_.store(p, std::memory_order_release);
}
```

Hazard pointer

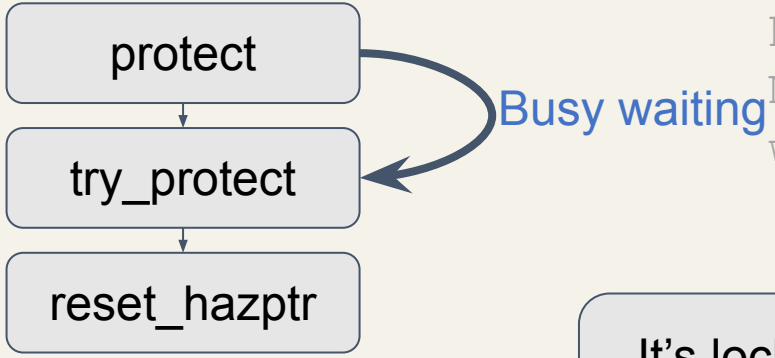



```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node
            retu
        auto nex
        if (cas_
```

Copy the pointer to protect zone

```
FOLLY_ALWAYS_INLINE void reset_hazptr(const void* p =
noexcept {
    hazptr_.store(p, std::memory_order_release);
}
```

Hazard pointer




```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node
            retu
        to nex
        cas_head(node,
```

It's lock-free not wait-free.

Copy the pointer to protect zone

```
FOLLY_ALWAYS_INLINE void reset_hazptr(const void* p =
noexcept {
    hazptr_.store(p, std::memory_order_release);
}
```

Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
```

```
FOLLY_ALWAYS_INLINE void reset_hazptr(const void* p = nullptr)
noexcept {
  hazptr_.store(p, std::memory_order_release);
 }
```

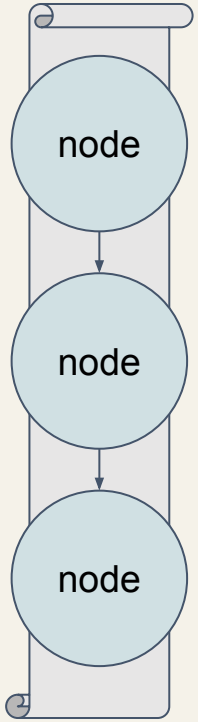
```
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)

TEAMT5
杜 浦 數 位 安 全

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

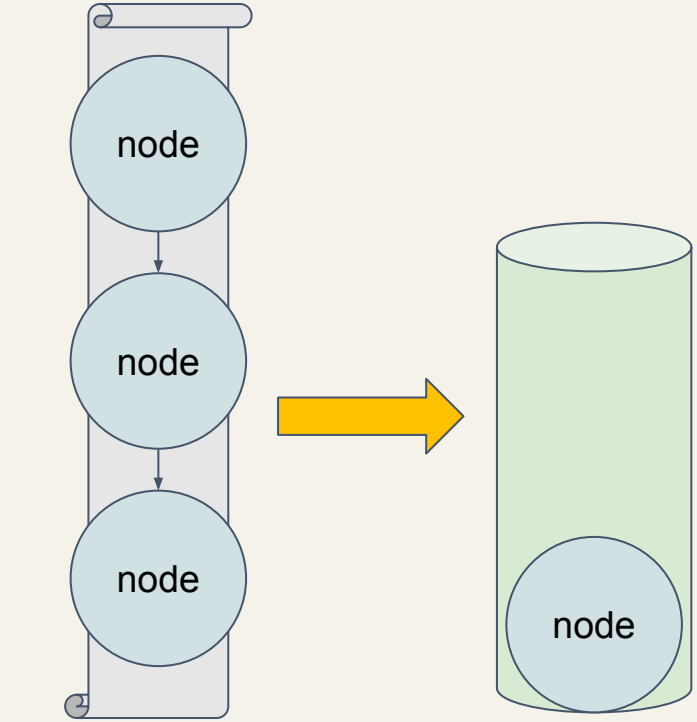
Hazard pointer (folly)



Hazptr_obj_list

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

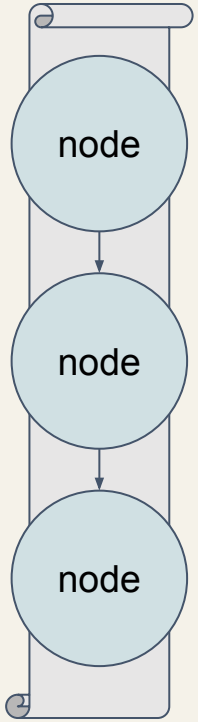
Hazard pointer (folly)

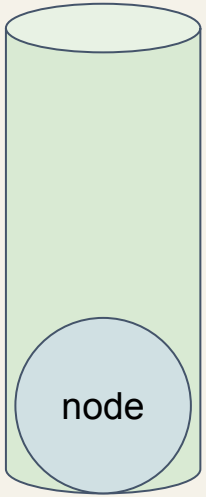


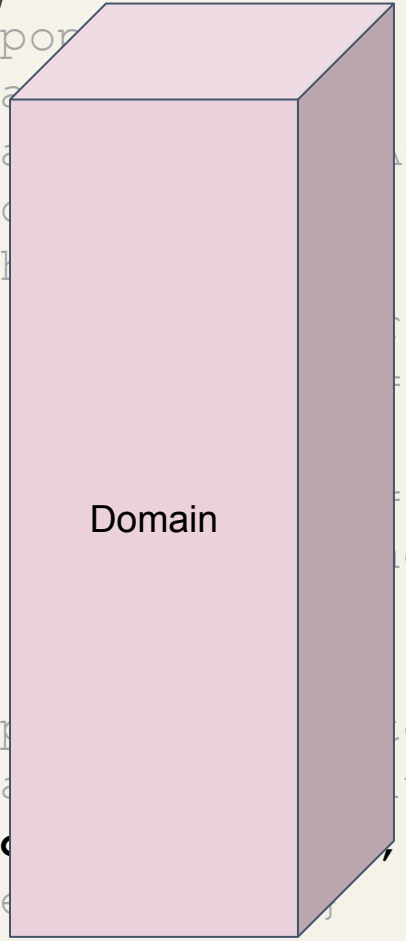
hazptr_obj_list hazptr_obj_cohort

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
    Node* node;
    while (true) {
        node = hptr.protect(head_);
        if (node == nullptr)
            return false;
        auto next = node->next();
        if (cas_head(node, next))
            break;
    }
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)



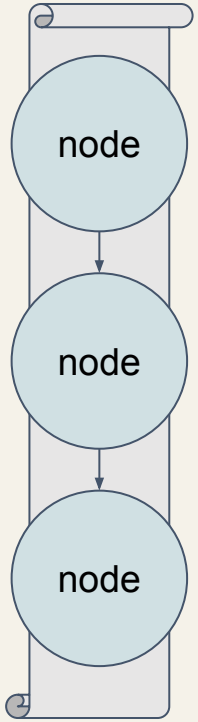


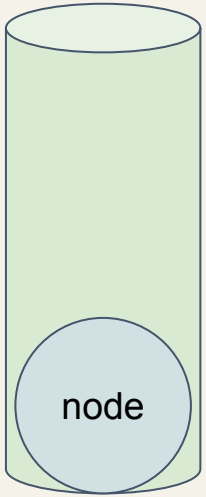


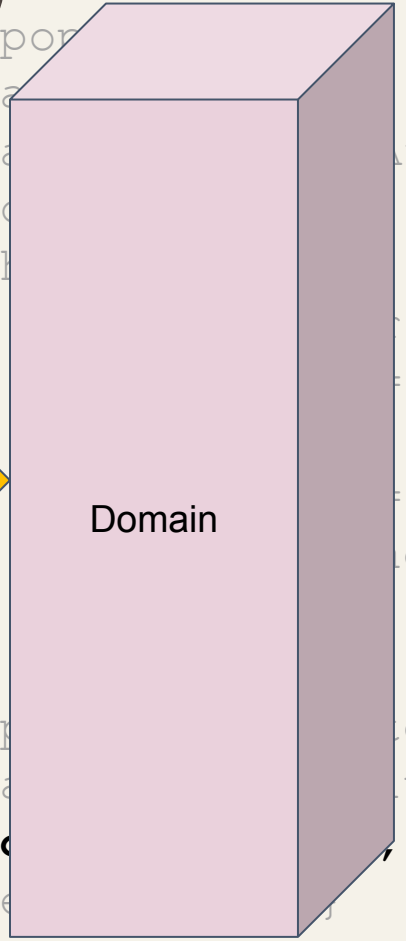
hazptr_obj_list

hazptr_obj_cohort

Hazard pointer (folly)







hazptr_obj_list

hazptr_obj_cohort

```
bool por
ha                    Atom> h;
ha                    tom>& hptr = h[0];
No
wh

                      .protect(head_);
                      = nullptr)
                      false;
                      = node->next();
                      d(node, next))

}
hp                    ection();
va                    ue();
no
re
```

Hazard pointer (folly)

TEAMT5 杜浦數位安全

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& domain = default_hazptr_domain<Atom>())
{
  pre_retire(std::move(deleter));
  set_reclaim();
  this->push_obj(domain); // defined in hazptr_obj
}
```

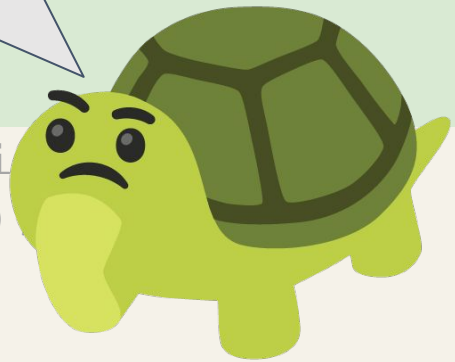
```
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& domain = default_
{
    pre_retire(std::move(deleter));
    set_reclaim();
    this->push_obj(domain); // defined in hazptr_
}
```

Delay deletion.
Set up deleting function.



```
hptr.reset_protecti
val = node->value()
node->retire();
return true;}
```

Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& domain = default_hazptr_domain<Atom>())
{
  pre_retire(std::move(deleter));
  set_reclaim();
  this->push_obj(domain); // defined in hazptr_obj
}
```

Delete directly if domain is not exist. Domain could be TLS, global or user-defined.

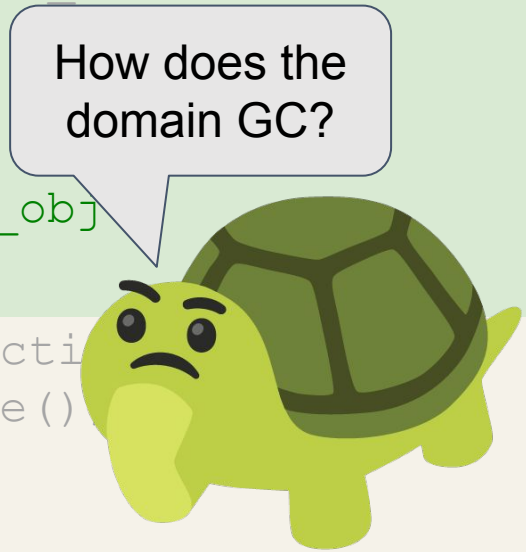
```
tection();
lue();
;
```

Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& domain = default_hazptr_domain<Atom>())
{
  pre_retire(std::move(deleter));
  set_reclaim();
  this->push_obj(domain); // defined in hazptr_obj
}
```

```
    hptr.reset_protecti
    val = node->value()
    node->retire();
    return true;}
```



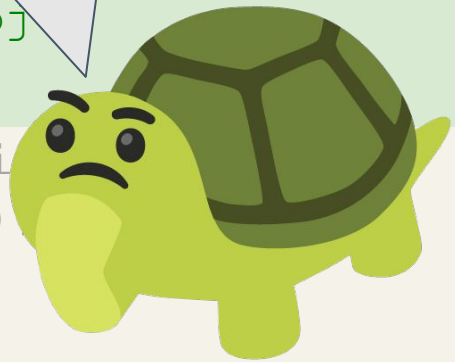
Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& domain = default_hazptr_domain<Atom>())
{
  pre_retire(std::move(deleter));
  set_reclaim();
  this->push_obj(domain);
}
```

1. Cleanup
2. Push to domain and timeup

How does the domain GC?



```
    hptr.reset_protection();
    val = node->value();
    node->retire();
    return true;}
```

Hazard pointer (folly)

```
bool pop(T& val) {
    hazptr_local<1, Atom> h;
    hazptr_holder<Atom>& hptr = h[0];
```

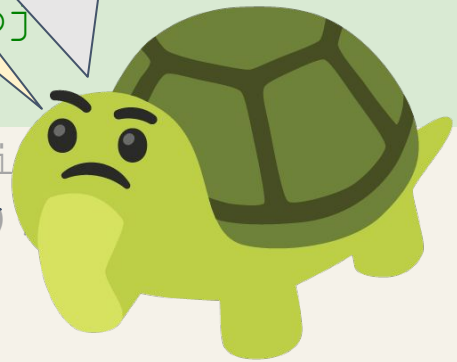
```
void retire(
    D deleter = {},
    hazptr_domain<Atom>& do                    ptr_domain<Atom>())
{
  pre_retire(std::move(deleter))
  set_reclaim();
  this->push_obj(doma                          bj
}
```

```
hptr.reset_protecti
val = node->value()
node->retire();
return true;}
```

Where is the reference count?

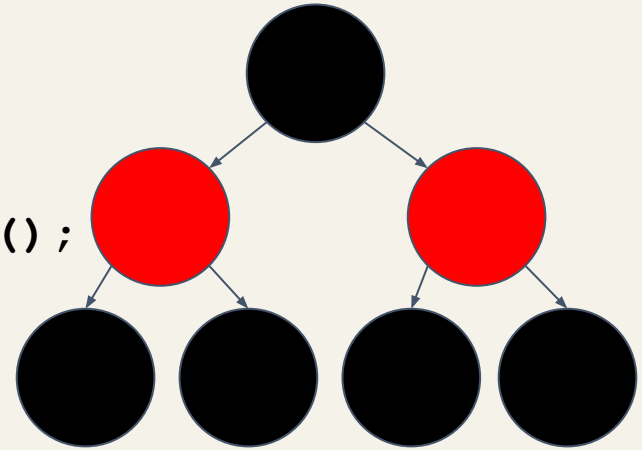
How does the domain GC?

1. Cleanup
2. Push to domain and timeup



Hazard object linking API (folly)

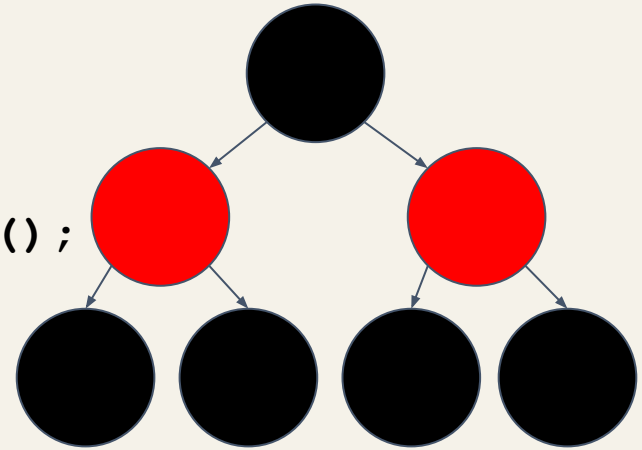
```
template <typename T, template <typename> class Atom, typename D>
class hazptr_obj_base_linked : public hazptr_obj_linked<Atom>,
                               public hazptr_deleter<T, D> {
   using Stack = std::stack<hazptr_obj_base_linked<T, Atom, D>*>;

   public:
   void retire();
   void unlink();
   void unlink_and_reclaim_unchecked();
   // ... private
}; // hazptr_obj_base_linked
```



Hazard object linking API (folly)

```
template <typename T, template <typename> class Atom, typename D>
class hazptr_obj_base_linked : public hazptr_obj_linked<Atom>,
                               public hazptr_deleter<T, D> {
    usi                    l::stack<hazptr_obj_base_linked<T, Atom, D>*>;

    pub
    voi
    voi                    nlink();
                           ink_and_reclaim_unchecked();
                           vate
                           _obj_base_linked
```

Yes, it has to be a DAG.

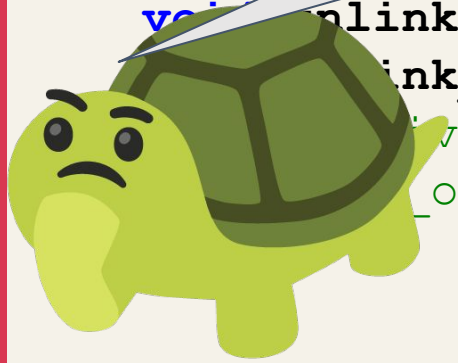


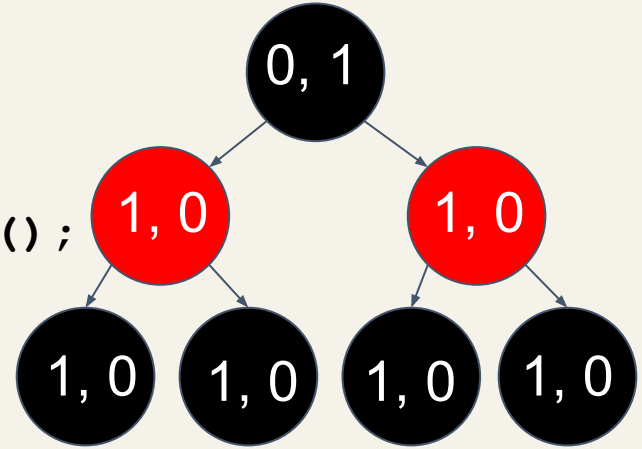
```
template <template <typename> class Atom>
class hazptr_obj_linked : public hazptr_obj<Atom> {
 using Count = uint64_t;

 static constexpr Count kRef = Count{1};
 static constexpr Count kLink = Count{1} << 32;
 static constexpr Count kRefMask = kLink - Count{1};
 static constexpr Count kLinkMask = ~kRefMask;

 Atom<Count> count_{0};

public:
 void acquire_link() noexcept { count_inc(kLink); }
 void acquire_link_safe() noexcept { count_inc_safe(kLink); }
 void acquire_ref() noexcept { count_inc(kRef); }
 void acquire_ref_safe() noexcept { count_inc_safe(kRef); }
}
```

Hazard object linking API (folly)

```
template <typename T, template <typename> class Atom, typename D>
class hazptr_obj_base_linked : public hazptr_obj_linked<Atom>,
                               public hazptr_deleter<T, D> {
    usi                    l::stack<hazptr_obj_base_linked<T, Atom, D>*>;

    pub
    voi
    voi                    nlink();
                           ink_and_reclaim_unchecked();
                           vate
                           _obj_base_linked
```

Yes, it has to be a DAG.





RCU

rcu_read_lock();

rcu_read_unlock();

synchronize_rcu();

```
void foo_read(void) {
    rcu_read_lock();
    foo *fp = global_foo;
    if (fp)
        do_something(fp->a, fp->b, fp->c);
    rcu_read_unlock();
}
void foo_update(foo *new_fp) {
    spin_lock(&foo_mutex);
    foo *old_fp = global_foo;
    global_foo = new_fp;
    spin_unlock(&foo_mutex);
    synchronize_rcu();
    kfree(old_fp);
}
```

RCU

```
void foo_read(void) {
    rcu_read_lock();
    foo *fp = global_foo;
    if (fp)
        do_something(fp->a, fp->b, fp->c);
    rcu_read_unlock();
}
v    foo_update(foo *new_fp) {
     in lock(&foo_mutex);
                 global_foo;
                 ew_fp;
                 oo_mutex);
                 u();
    kfree(old_fp);
}
```

rcu_read_lock();

rcu_read_unlock();

synchronize_rcu();

Prohibit removal when reading.

RCU

```
void foo_read(void) {
    rcu_read_lock();
    foo *fp = global_foo;
    if (fp)
        do_something(fp->a, fp->b, fp->c);
    rcu_read_unlock();
```

```
e(foo *new_fp) {
foo_mutex);
= global_foo;
glo  _foo = new_fp;
spi_unlock(&foo_mutex);
synchronize_rcu();
kfree(old_fp);
}
```

rcu_read_lock();

rcu_read_unlock();

synchronize_rcu();

Truly remove when everyone not in CS.

RCU

```
void foo_read(void) {
    rcu_read_lock();
    foo *fp = global_foo;
    if (fp)
        do_something(fp->a, fp->b, fp->c);
    rcu_read_unlock();
```

```
    glo    _foo = new_
    spi_unlock(&foo_    utex);
    synchronize_rcu();
    kfree(old_fp);
}
```

rcu_read_lock();

rcu_read_unlock();

synchronize_rcu();

Truly remove when everyone not in CS.

If any other thread in the CS. Keep it in the grace period.

RCU

TEAMT5 杜浦數位安全

rcu_read_lock();

rcu_read_unlock();

synchronize_rcu();

```
void
                    mething(fp->a, fp->b, fp->c);
                    lock();
                    e(f
                    foo
                    =
                    glo    _fo    = new_
                    spi_unlock(&foo_    utex);
                    synchronize_rcu();
                    kfree(old_fp);
}
```

Blocking until release.

Truly remove wh
everyone not in C

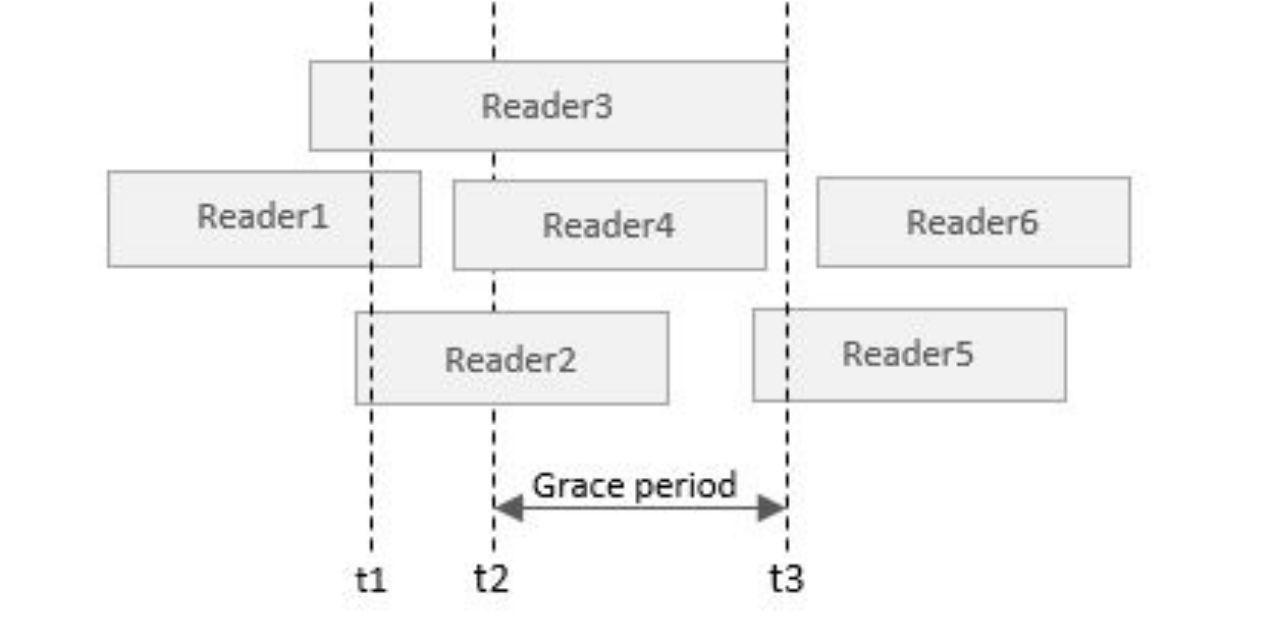
If any other thread in the CS. Keep it in the grace period.

RCU

rcu_read_lock();

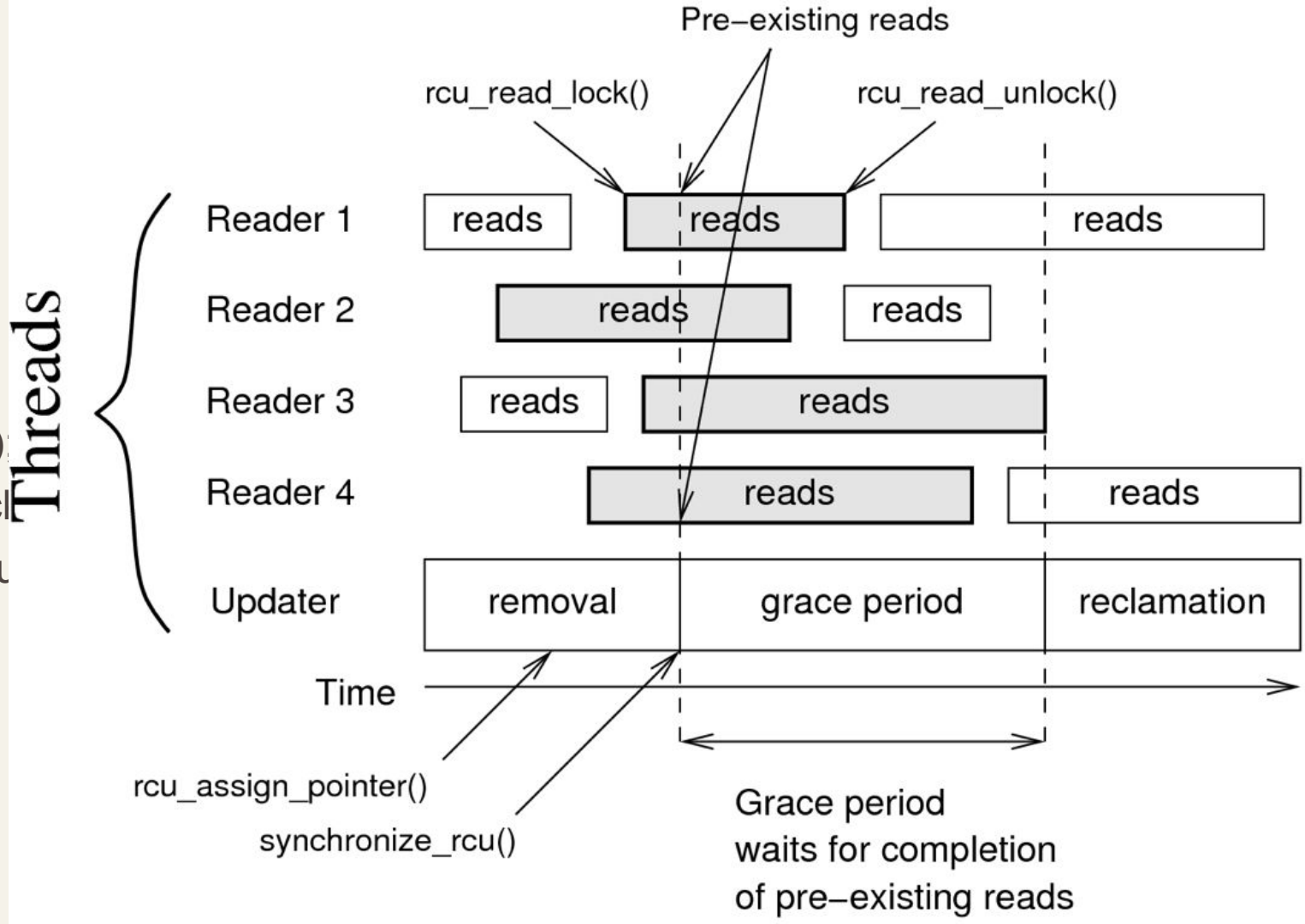
rcu_read_unlock();

synchronize_rcu();



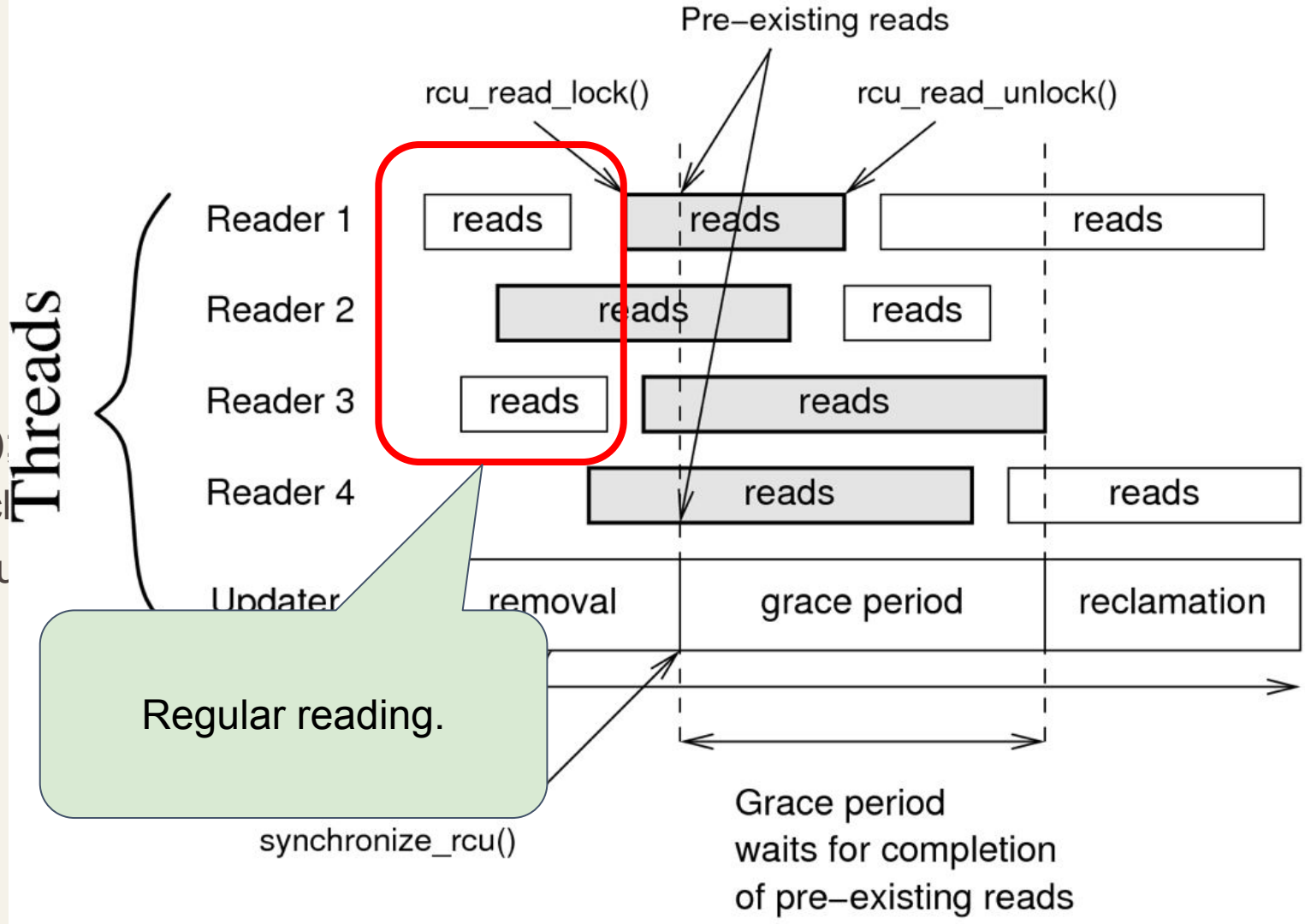
RCU

rcu_read_lock()
rcu_read_unlock
synchronize_rcu



RCU

rcu_read_lock()
rcu_read_unlock
synchronize_rcu



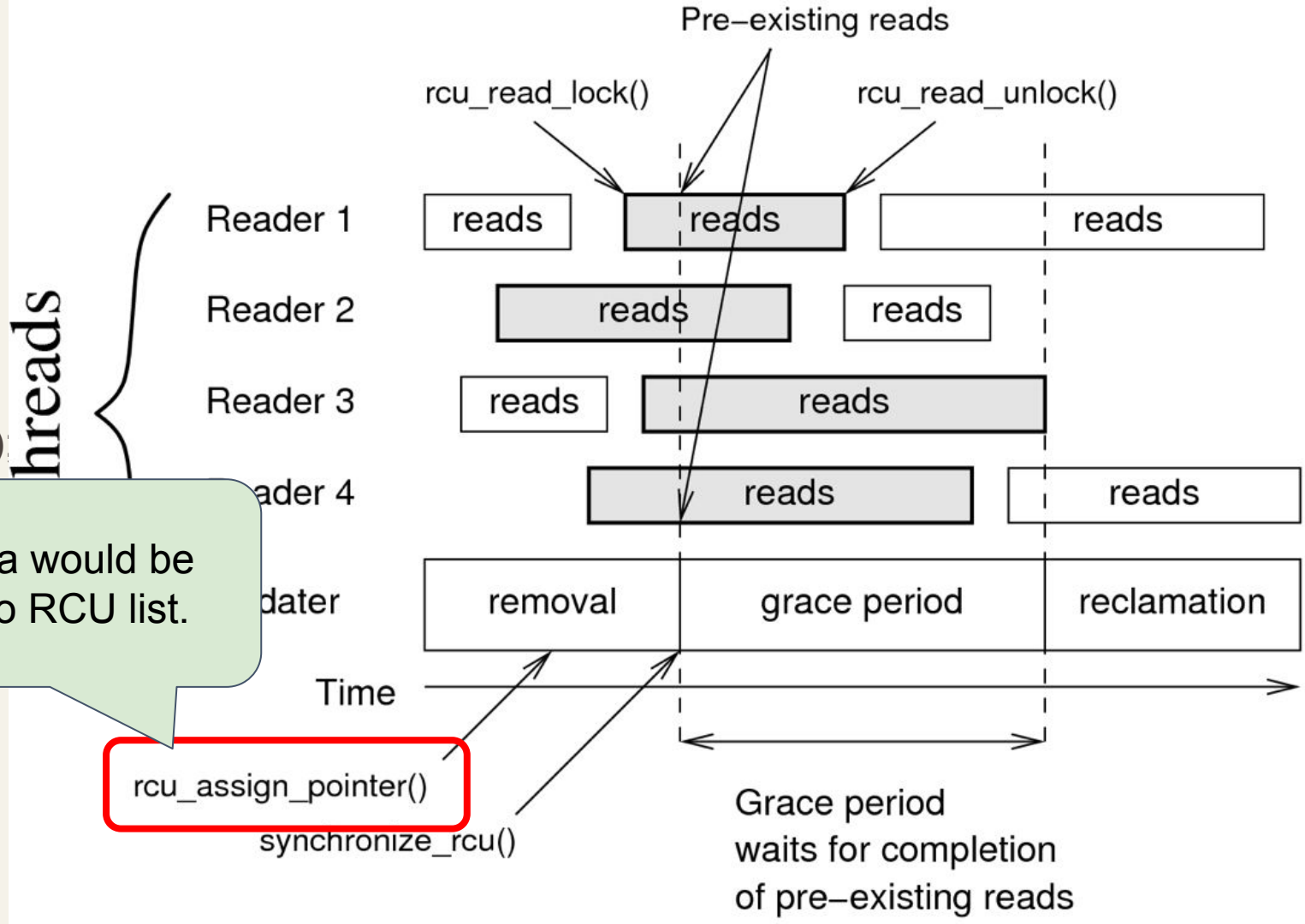
RCU

rcu_read_lock()

rc

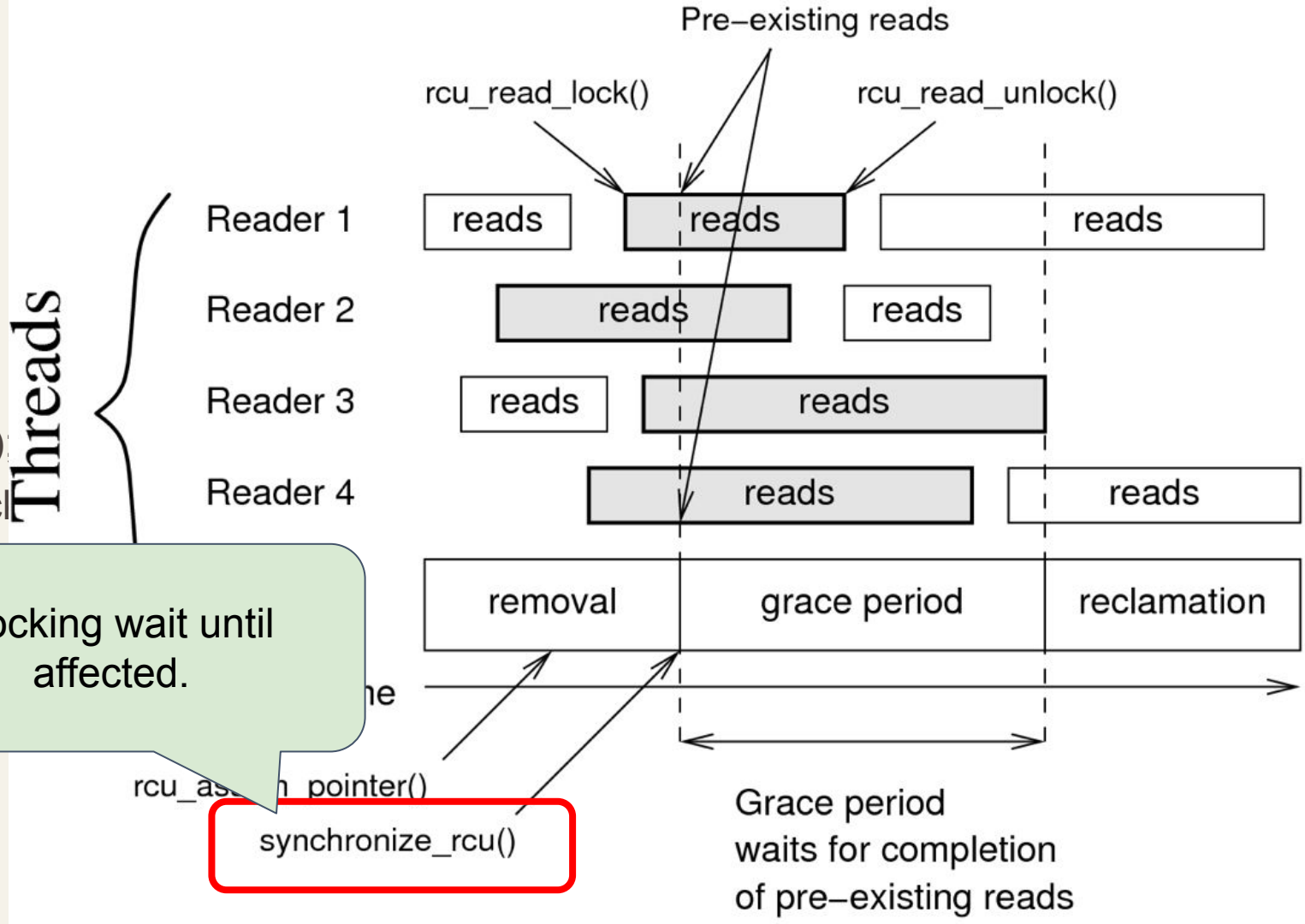
sy

Some data would be inserted to RCU list.



RCU

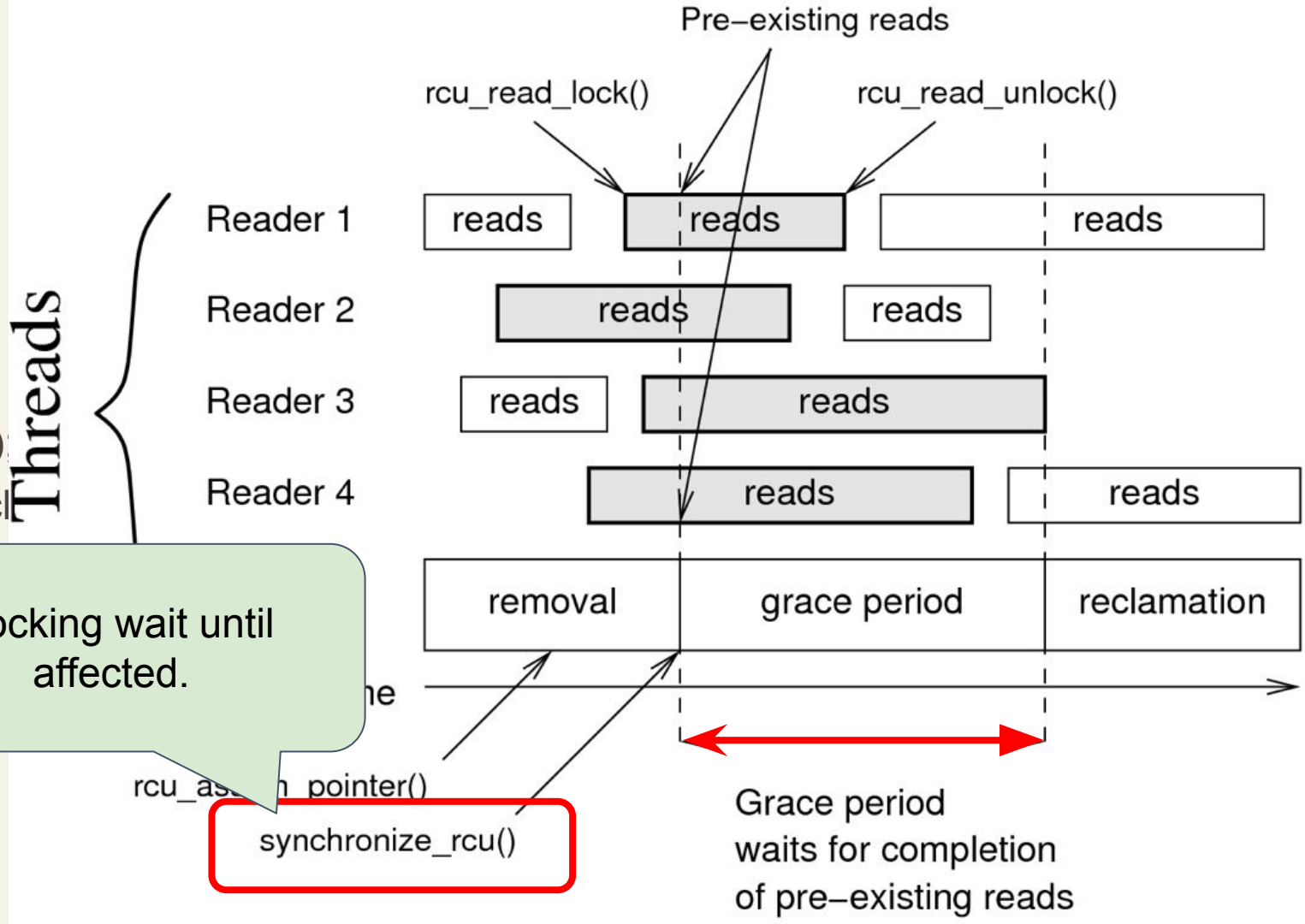
rcu_read_lock()
rcu_read_unloc
synchro



Blocking wait until affected.

RCU

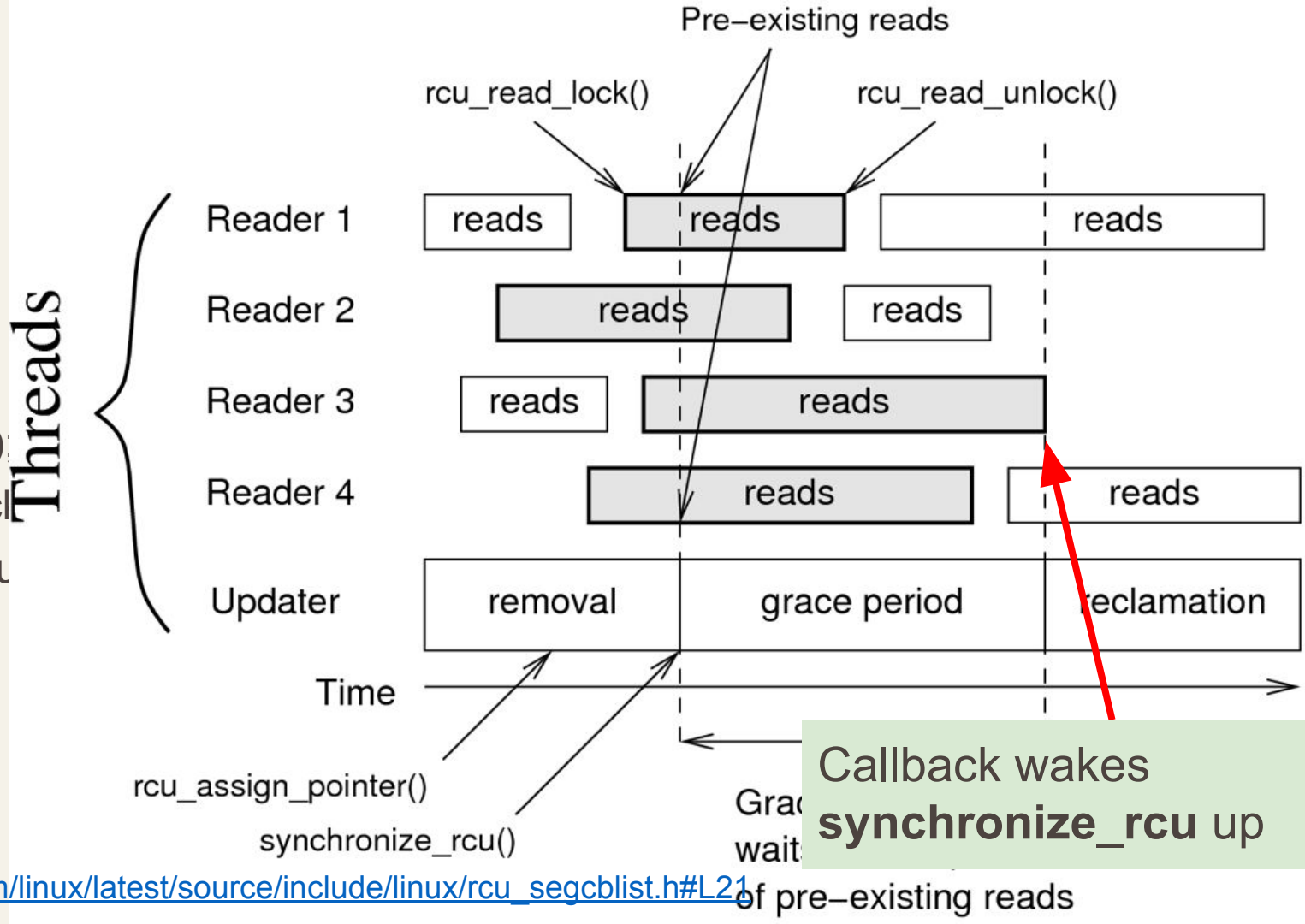
rcu_read_lock()
rcu_read_unloc
synchro



Blocking wait until affected.

RCU

rcu_read_lock()
rcu_read_unlock
synchronize_rcu

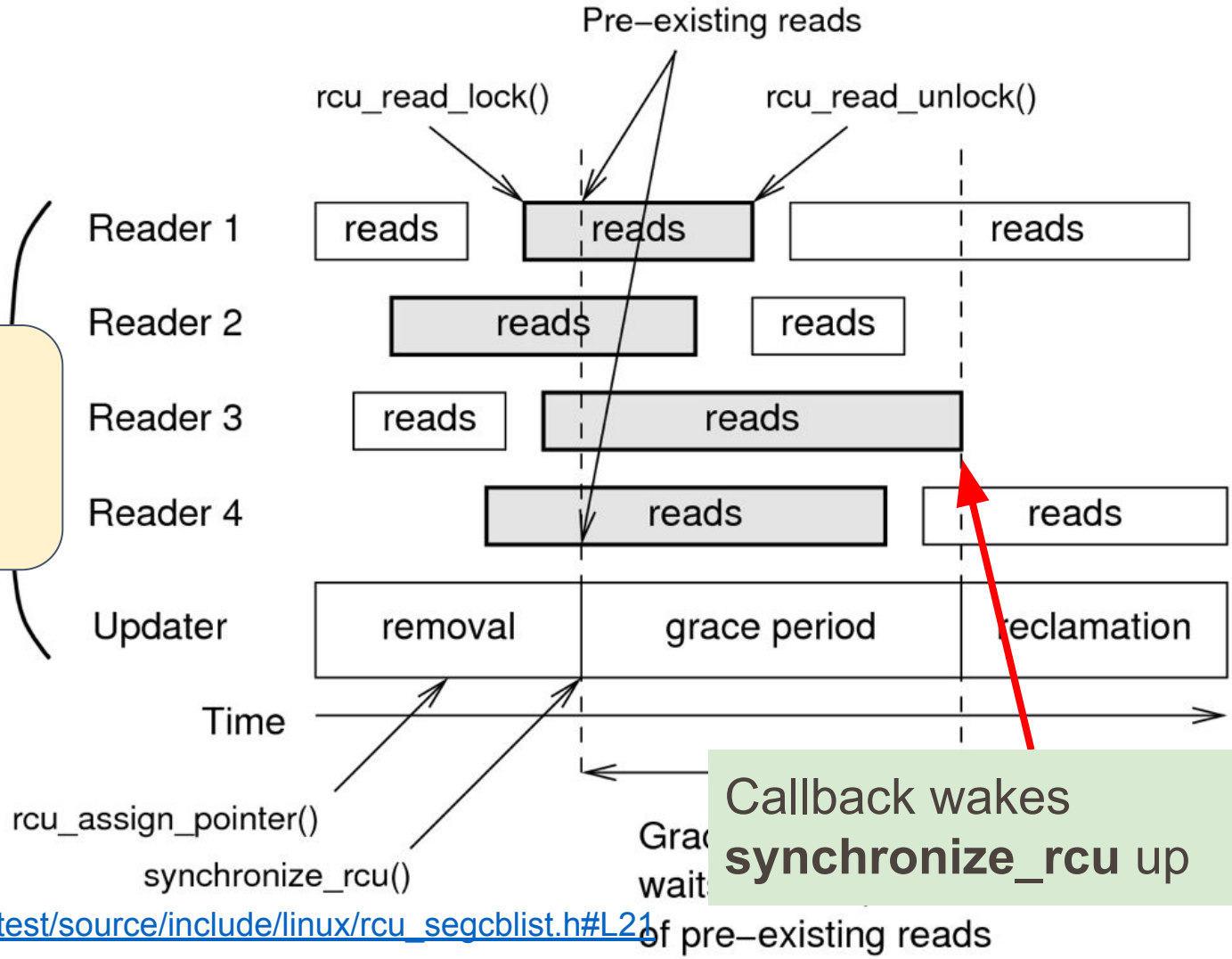


Callback wakes **synchronize_rcu** up

https://elixir.bootlin.com/linux/latest/source/include/linux/rcu_segcblist.h#L21

RCU

All RCUs share a global RCU_manager.
Once it is cleaned, all retired nodes are released.



Callback wakes **synchronize_rcu** up

https://elixir.bootlin.com/linux/latest/source/include/linux/rcu_segcblist.h#L21

Hazard Pointers vs RCU

TEAMT5 杜浦數位安全

	RCU	Hazard Pointers
Readers Progress	wait-free population oblivious	lock-free (sometimes wait-free)
Reclaimer Progress	blocking	wait-free bounded linearly
Readers Synchronization per node	none	2 seq-cst load 1 seq-cst store
Read Synchronization overhead	2 seq-cst loads 2 seq-cst stores	none
Max Nodes to be deleted	O(1)	O(threads x hps)
Ease of deployment	very easy	hard

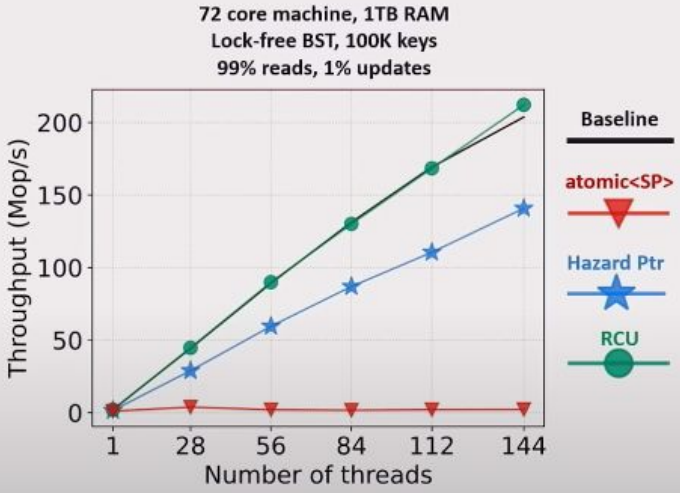
https://concurrencyfreaks.blogspot.com/2016/08/hazard-pointers-vs-rcu.html

Cppcon 2022 The C++ Conference
September 12th-16th

Daniel Anderson

Smarter Atomic Smart Pointers: Efficient Concurrent Memory Management for Everybody

RCU: Actual performance



Carnegie Mellon University

Video Sponsorship Provided By:
think-cell

https://www.youtube.com/watch?v=OS7Asaa6zmY

CC-BY-NC-SA

Thank you for your attention.