C++ for Real Time Communications in the Cloud

Thiya Ramalingam
Head of Platforms
Zoom Video Communications
https://www.linkedin.com/in/thiya

About Zoom


Meetings
Online Meeting, Training & Technical Support
Chat
Cross-Platform Messaging & File Sharing
Zoom for Developers
Empower Your Apps with Video & Screen Sharing
One Consistent Enterprise Experience
Video Webinar
Marketing Events & Town Hall Meetings
H.323/SIP Connector
Bring H.323/SIP Video Systems to the Cloud
Zoom Rooms
Build Collaboration-Enabled Conference Rooms

Performance Optimizations

- Threading
- Branch prediction
- Timers
- Containers
- Object lifetimes

Threading

- Create thread pools
  - Event driven async main thread
  - One or more sync or async auxillary threads or message aware async thread pool

- Locking is expensive
- Debugging is a nightmare
- Keep the common data to minimum

Threading – HW profiling

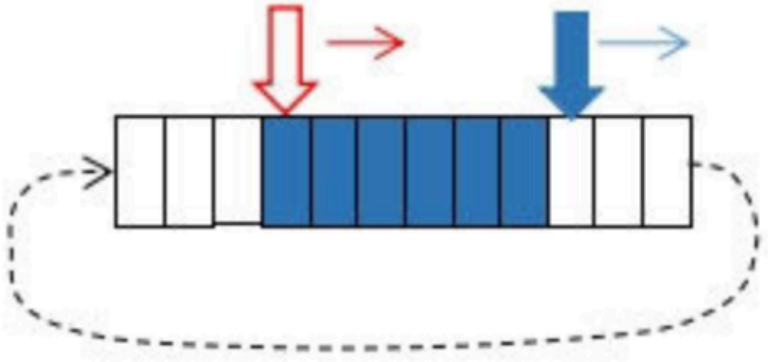
```
unsigned int hardware_concurrency()
{
   unsigned int cores = std::thread::hardware_concurrency();
   return cores ? cores: my_hardware_concurrency();
}

auto gcc_hardware_concurrency()
{
   std::ifstream cpuinfo("/proc/cpuinfo");

   return std::count(std::istream_iterator<std::string>(cpuinfo),
                     std::istream_iterator<std::string>(),
                     std::string("processor"));
}
```

Threading – CPU Affinity

```
unsigned int id = 1;
cpu_set_t cpuset;
CPU_ZERO(&cpuset);
CPU_SET(id, &cpuset);

pthread_t current_thread = pthread_self();
pthread_setaffinity_np(current_thread, sizeof(cpu_set_t),
                       &cpuset);
```

Threading– No Lock Queues

Locks are expensive – use no-lock queues for ITC



Threading - Re-ordering Impacts

Memory reordering rule: "The re-ordering shall not modify the behavior of a single threaded program"

```
int D, E;

void compute()
{
    D  = E + 2;
    E = 0;
}
```

```
8         push    rbp
9         mov     rbp, rsp
10        mov     eax, DWORD PTR E[rip]
11        add     eax, 2
12        mov     DWORD PTR D[rip], eax
13        mov     DWORD PTR E[rip], 0
14        mov     eax, 1
15        pop     rbp
```

```
6  .LFB1025:
7         mov     eax, DWORD PTR E[rip]
8         mov     DWORD PTR E[rip], 0
9         add     eax, 2
10        mov     DWORD PTR D[rip], eax
11        mov     eax, 1
12        ret
```

Threading – Memory Barriers

Ensure that the correct re-ordering is enforced.

```
int global;
int updated;

void compute()
{
    global = x;
    updated = 1;
}
```

Threading – Compiler Fencing

Use the compiler barriers as needed

```
6  int main()
7  {
8      D = E + 2;
9      asm volatile("" ::: "memory");
10     E=0;
11
12     return 1;
13 }
```

```
4  .Ltext_cold0:
5  main:
6  .LFB1025:
7          mov     eax, DWORD PTR E[rip]
8          add     eax, 2
9          mov     DWORD PTR D[rip], eax
10         mov     DWORD PTR E[rip], 0
11         mov     eax, 1
12         ret
```

Performance Optimizations

- Threading
- Branch prediction
- Timers
- Containers
- Object lifetimes

Performance – Micro-optimizations

```
int main()
{
    const auto size = 102400;
    int arr[size];
    auto count = 0;

    for (auto i=0; i<size; i++) {
        arr[i] = std::rand() % 1024;
    }

    for (auto i=0; i<size; i++) {
        for (auto j=0; j<size; j++) {
            if (arr[j] <= 128) {
                count ++;
            }
        }
    }
    std::cout << "Total = " << count << std:endl;

}
```

Execution time

1.6s

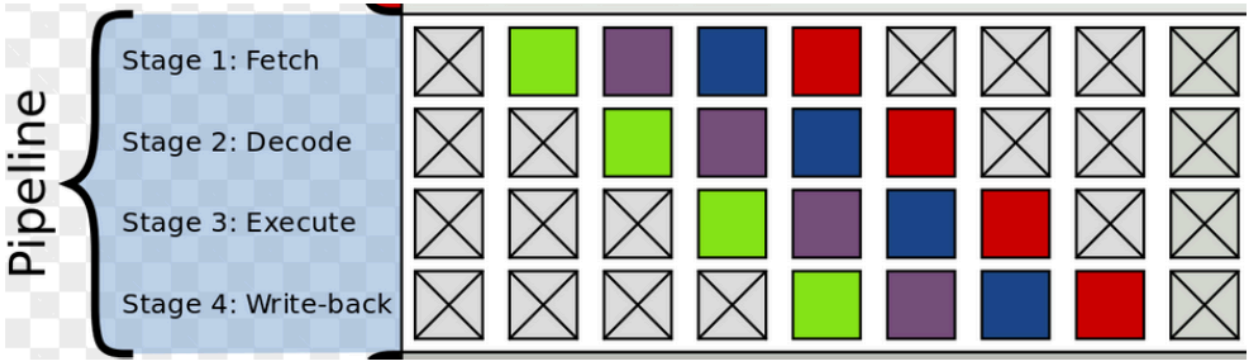
Performance – Micro-optimizations

```
int main()
{
    const auto size = 102400;
    int arr[size];
    auto count = 0;

    for (auto i=0; i<size; i++) {
        arr[i] = std::rand() % 1024;
    }

    std::sort(arr, arr+size);

    for (auto i=0; i<size; i++) {
        for (auto j=0; j<size; j++) {
            if (arr[j] <= 128) {
                count ++;
            }
        }
    }
    std::cout << "Total = " << count << std:endl;

}
```

Execution time

0.06s

Branch Prediction




```
for (auto i=0; i < size; i++) {
    arr[i] = std::rand() % 1024;
}

// work load
for (auto i = 0; i < size; i++) {
for (auto j = i; j < size; j++) {
    if (arr[j] <= 128) {
      count++;
```

```
34   mov DWORD PTR [rbp-16], eax
35 .L7:
36   cmp DWORD PTR [rbp-16], 102399
37   jg .L5
38   mov eax, DWORD PTR [rbp-16]
39   cdqe
40   mov eax, DWORD PTR [rbp-409632+rax*4]
41   cmp eax, 128
42   jg .L6
43   add DWORD PTR [rbp-4], 1
```

Performance Optimizations

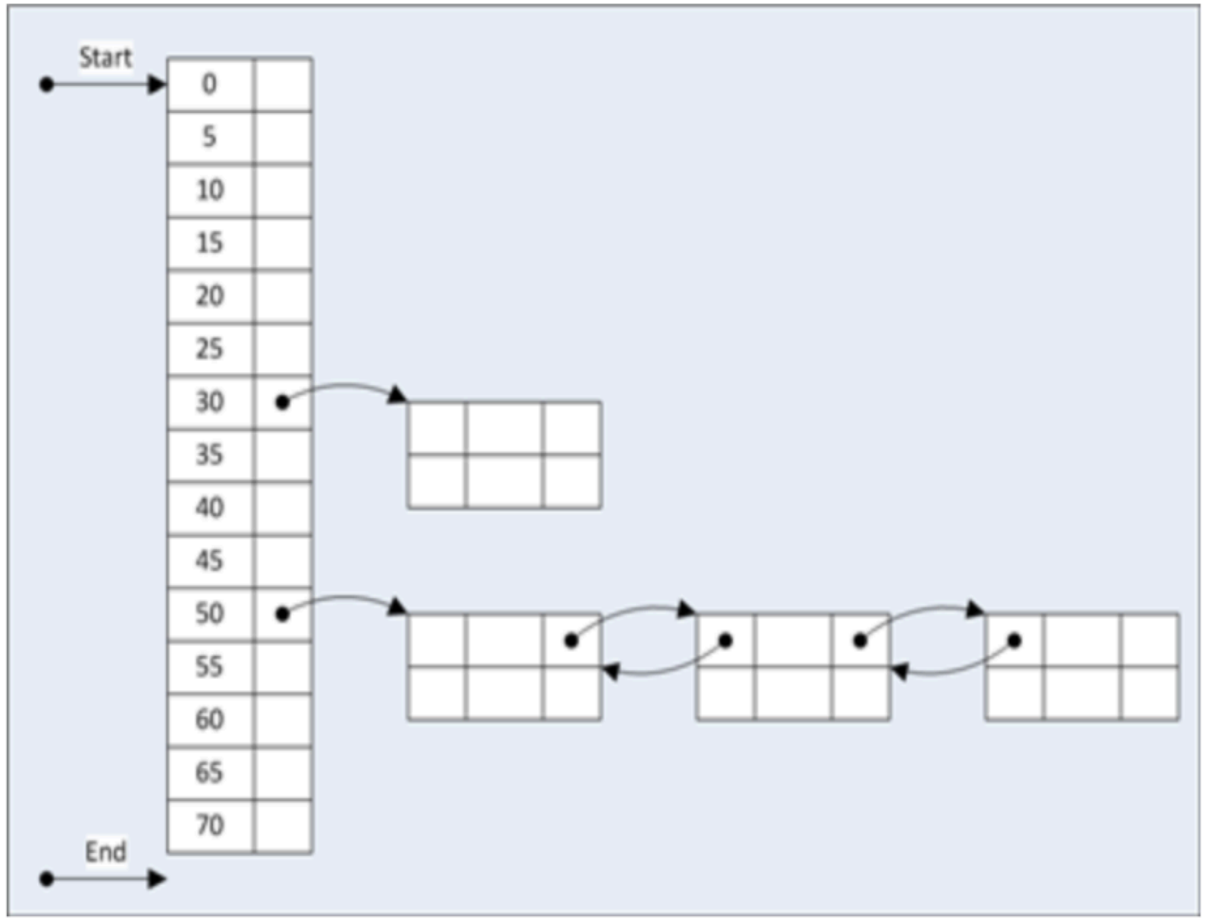
- Threading
- Branch prediction
- Timers
- Containers
- Object lifetimes

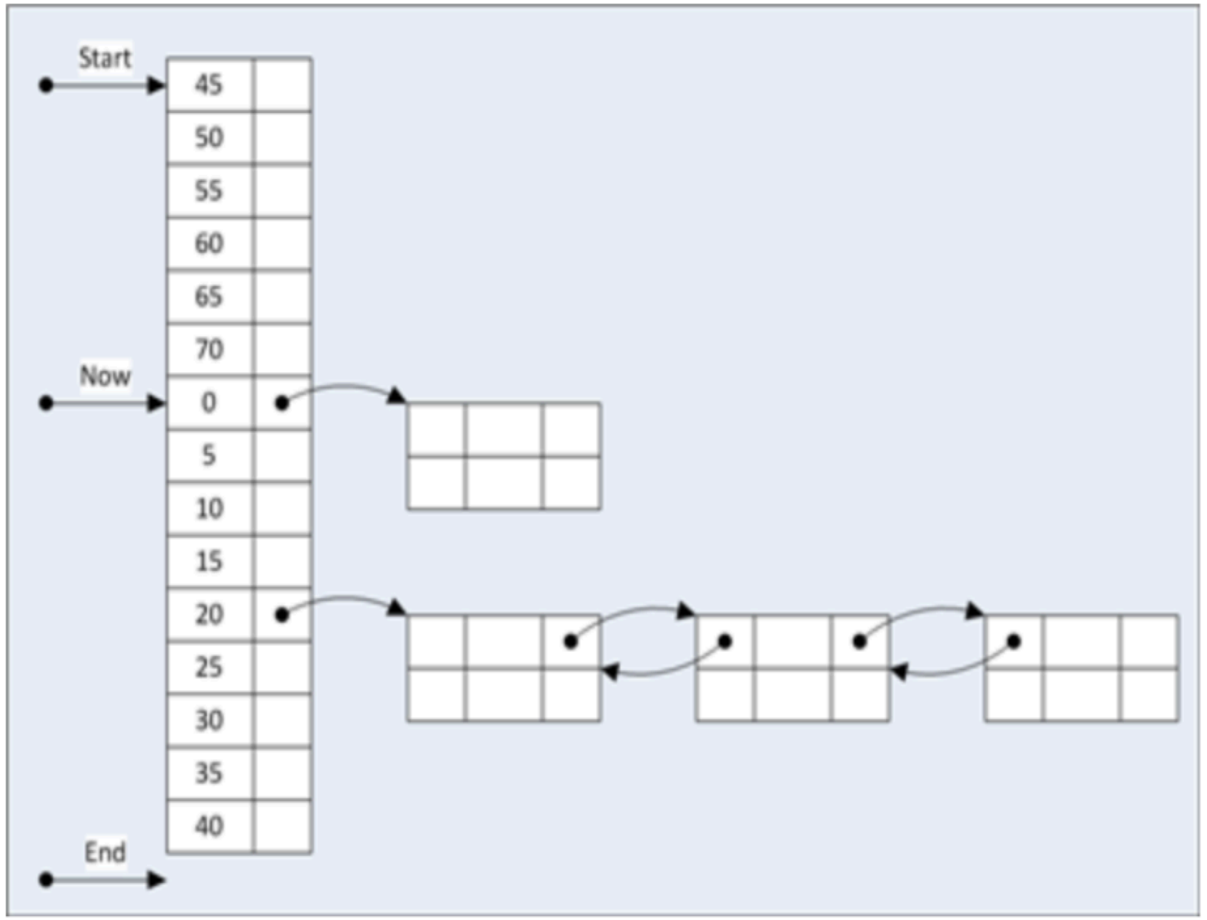
Timers

- Critical for our platform – 5ms to 24 hours
- Time Unit = 5ms
- Interfaces – Set and Cancel
- Timer thread – Thread priority to 99 and CPU affinity
- Timer Wheel Algorithms

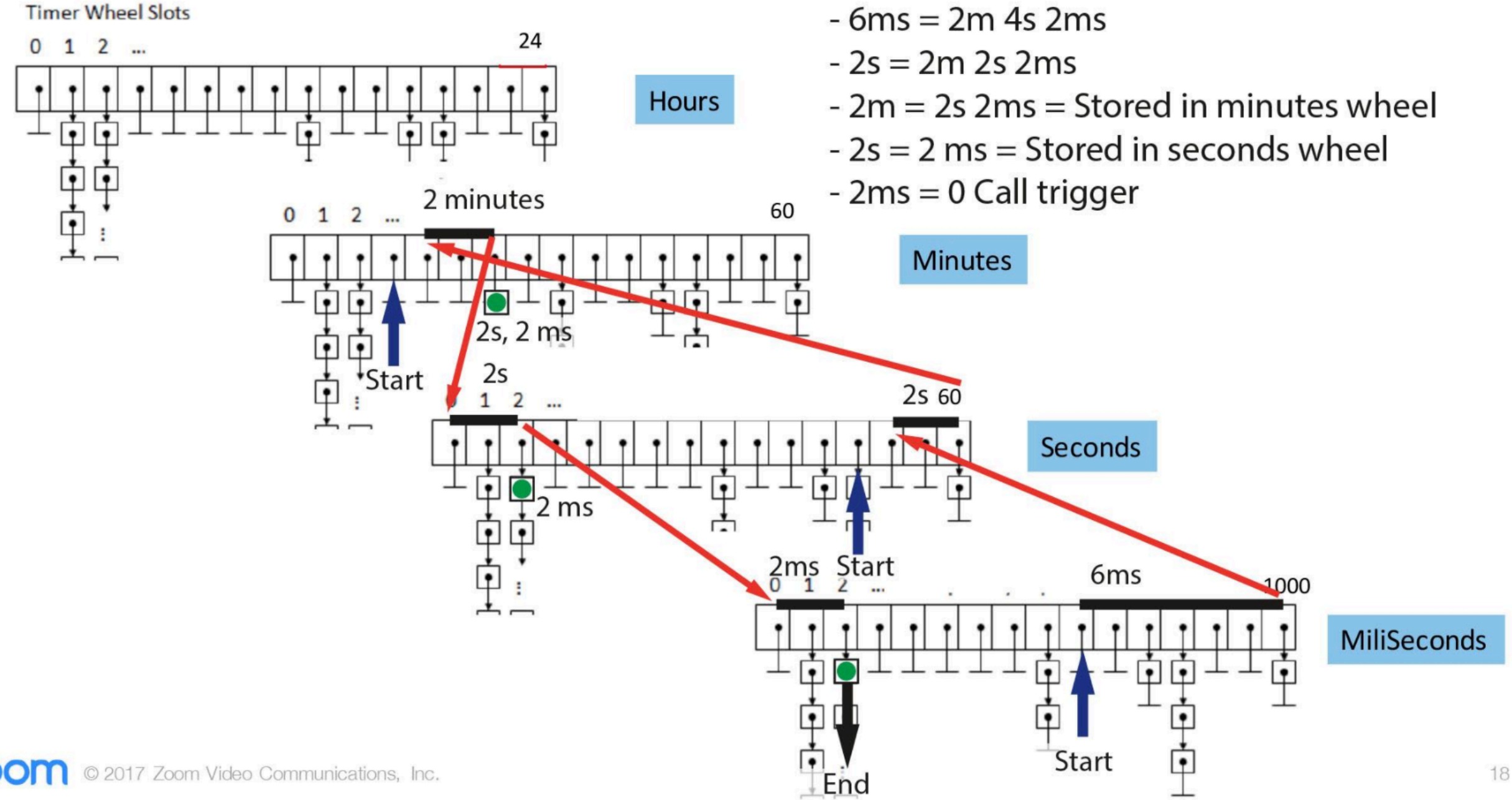
Timers – Basic Timer Wheel

- Each slot can be an array index - insertion and removal O(1)





Timers – Staggered Wheel



Example:

Trigger timer in 2 minutes, 4 seconds, and 8 ms

- 6ms = 2m 4s 2ms
- 2s = 2m 2s 2ms
- 2m = 2s 2ms = Stored in minutes wheel
- 2s = 2 ms = Stored in seconds wheel
- 2ms = 0 Call trigger

Performance Optimizations

- Threading
- Branch prediction
- Timers
- Containers
- Object Lifetimes

Containers

```
std::vector<int> v;
v.push_back(10);
```

```
std::list<int> l;
l.push_back(10);
```

Containers

```
std::list<int> l;
l.push_back(10);
```

Containers

Containers

```
49    movl $0, %eax
50    jmp .L8
51    movq %rax, %rbx
52    leaq -33(%rbp), %rax
53    movq %rax, %rdi
54    call std::allocator<int>::~allocator()
55    movq %rbx, %rax
56    movq %rax, %rdi
57    call _Unwind_Resume
```

Handle exception

Containers

```
646   movq (%rax), %rcx
647   movq -16(%rbp), %rdx
648   addq %rcx, %rdx
649   movq %rdx, (%rax)
650   nop
651   leave
652   ret
```

Assign the value

Containers

```
660 __gnu_cxx::new_allocator<std::_List_node<int> >::deallocate(st
661   pushq %rbp
662   movq %rsp, %rbp
663   subq $32, %rsp
664   movq %rdi, -8(%rbp)
665   movq %rsi, -16(%rbp)
666   movq %rdx, -24(%rbp)
667   movq -16(%rbp), %rax
668   movq %rax, %rdi
669   call operator delete(void*)
670   nop
671   leave
672   ret
```

Delete the node

Containers

std::vector<int> v;
v.push_back(10);

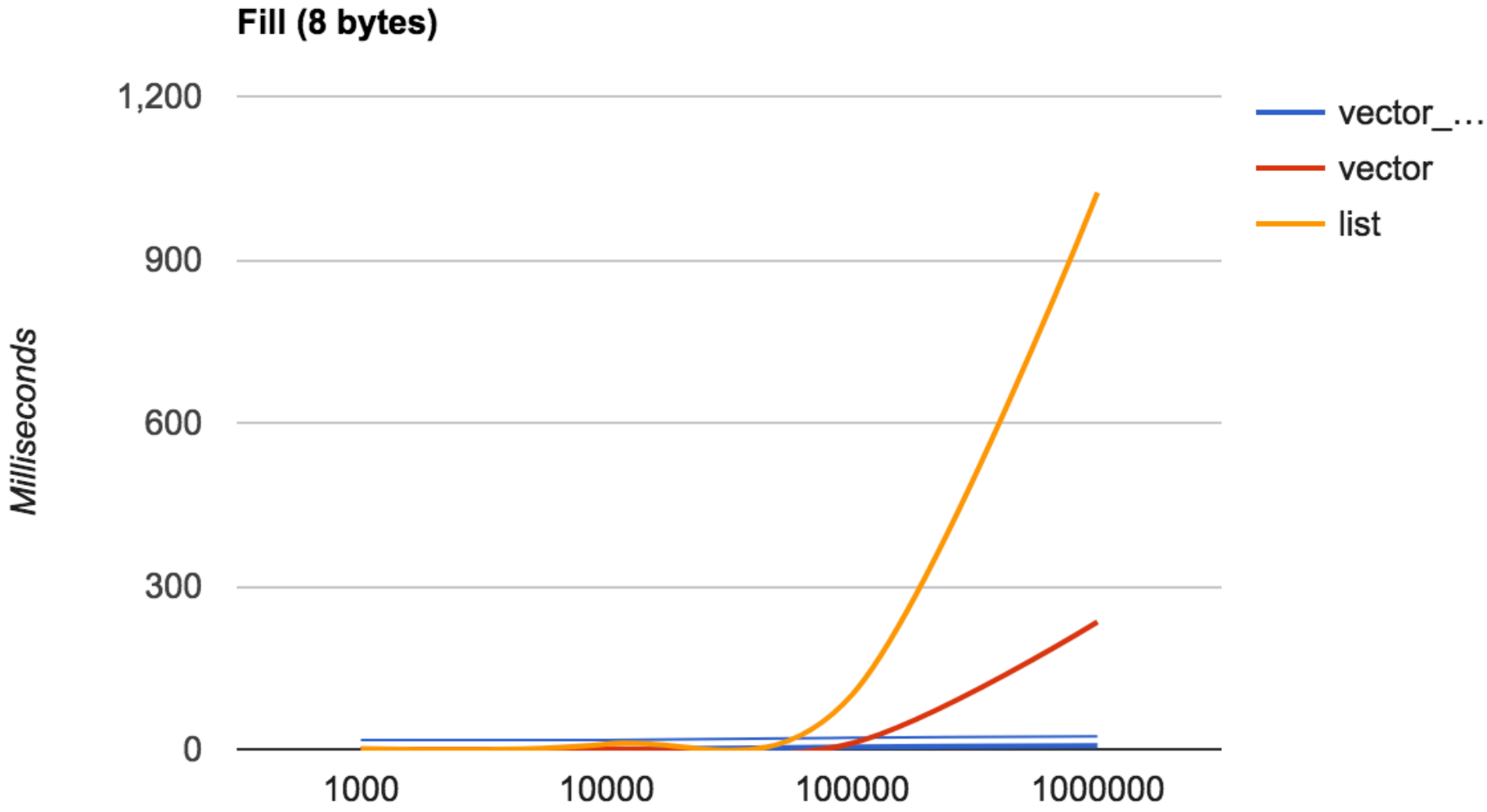
```
1  main:
2          sub     rsp, 8
3          mov     edi, 4
4          call    operator new(unsigned long)
5          mov     DWORD PTR [rax], 10
6          mov     rdi, rax
7          call    operator delete(void*)
8          mov     eax, 1
9          add     rsp, 8
10         ret
```

Containers

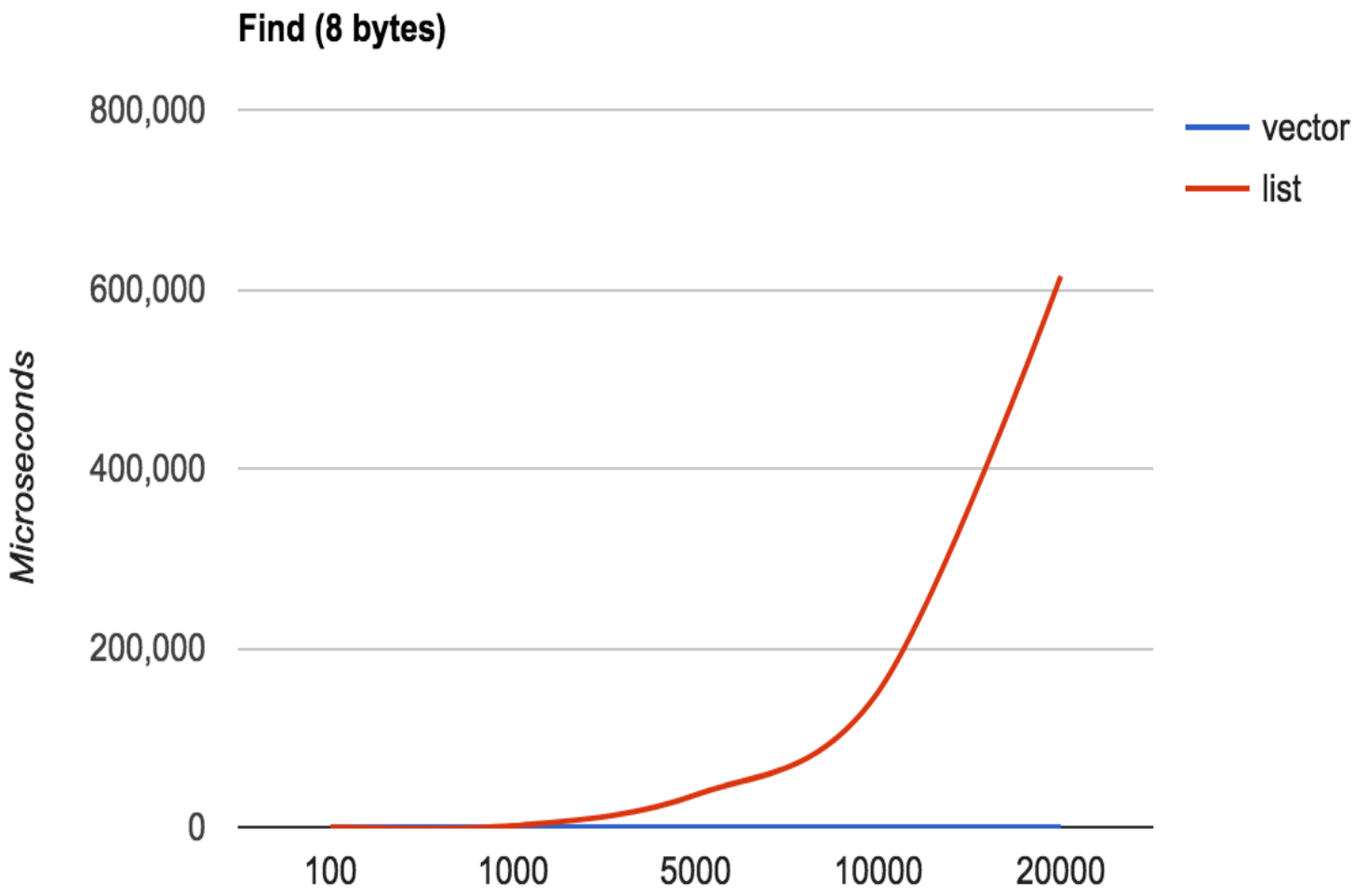
```
std::vector<int> v;
v.push_back(10);
```

```
std::list<int> l;
l.push_back(10);
```

Containers – Vectors vs List



Containers – Vectors vs List



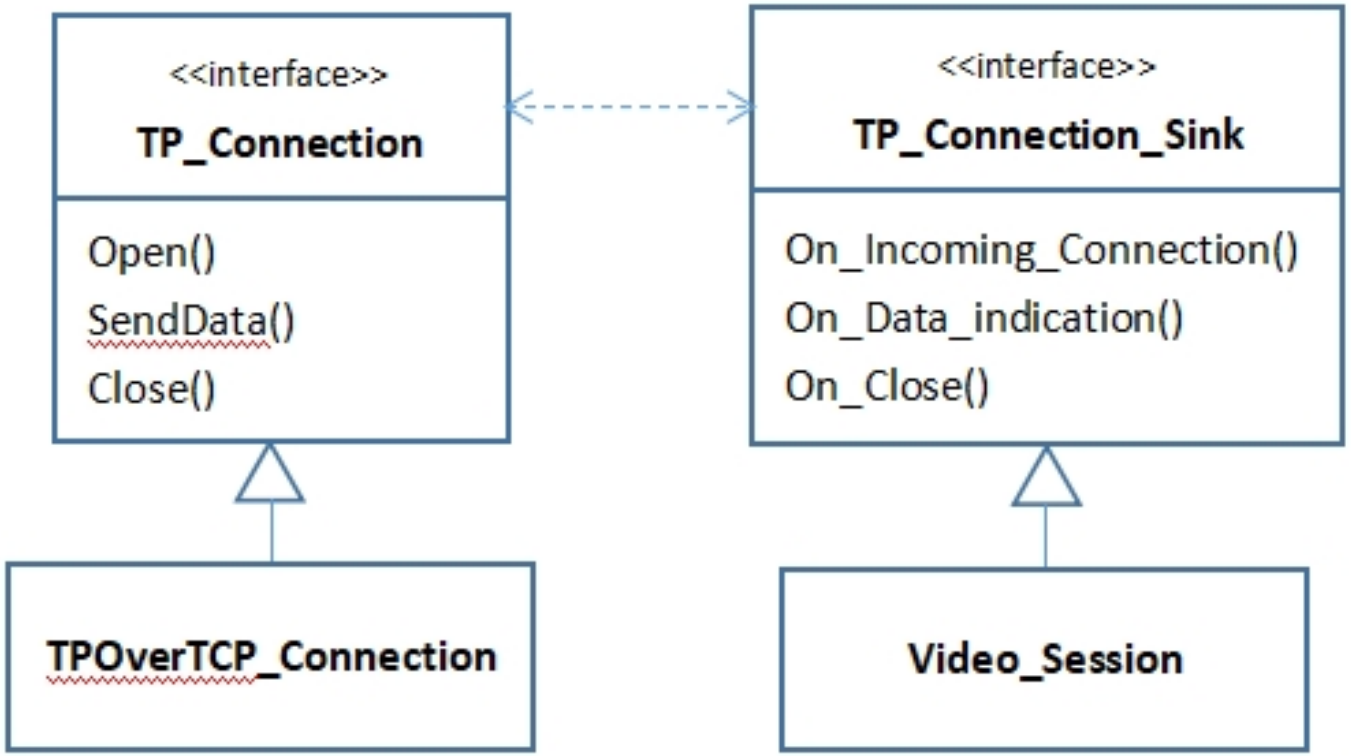
Containers – Beware of hidden [cost of] memory allocations

```
class mem {
    private:
        map<int, string> m;
    public:
        void insert(map<int, string> &m) {
            m[100] = “test”;
        }
}
```

Performance Optimizations

- Threading
- Branch prediction
- Timers
- Containers
- Object Lifetimes

Object Lifetimes



Object Lifetimes

```
int cnt=_packet_vector.size();

for(int i=0;i<cnt;++i)
{
    if(_sink) _sink->On_Data_indication(_packet_vector[i]);
}
```

Object Lifetimes

```
void Video_Session::On_Data_indication(packet_t *pkt)
{
    Video_Data vdata(pkt);
    if(vdata.m_resolution == 1080P)
    {
        if(_conn)
       {
           _conn->Close(E_NOT_SUPPORTED);
           _conn->release_reference();
           _conn=nullptr;
       }
     }
}
```

Object Lifetimes

```
void TCP_Connection::Close(int error_code)
{
    _packet_vector.clear();

    if(_sink) _sink->release_reference();
    _sink=nullptr;

}
```

Profiling, Debugging

- Godbolt.org
- Valgrind, Intel Inspector, Google sanitizer
- Cppcheck
- Intel vTunes
- Google benchmark library

Final Thoughts

C++ is hard – use the right tools and judgment

Thank You

Connect With Us

@zoom_us | blog.zoom.us