LAB 2 discussion

Simulation based Process Scheduling

- Explore multiple Schedulers and what OS tracks
- You will learn how to write a Discrete Event Simulator
- Specification of Process and Random Behavior

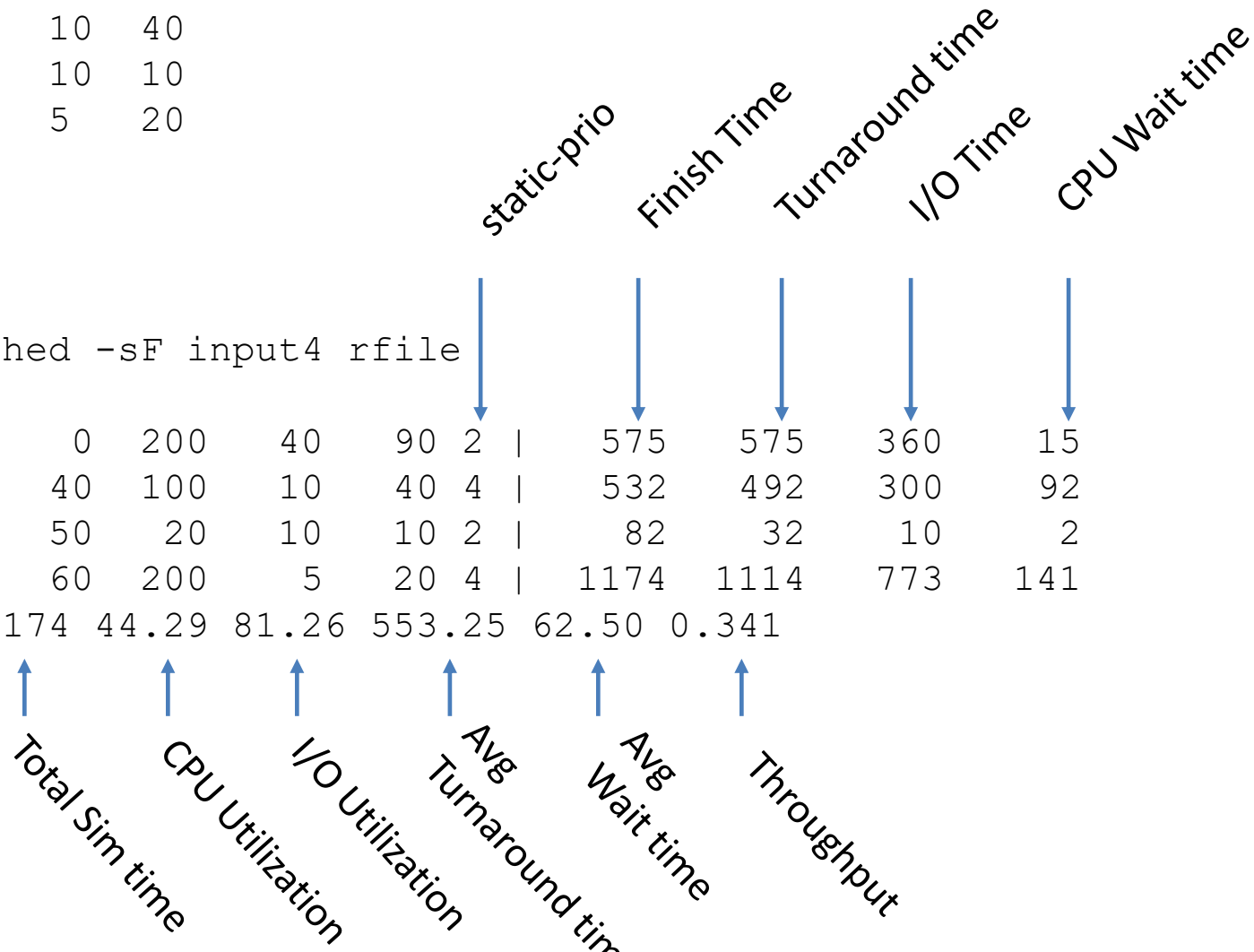
Process-1	0	200	40	90
	40	100	10	40
	50	20	10	10
Process-4	60	200	5	20
	process arrival/start	Total CPU-time	CPU Burst	IO Burst

- Cpu-burst = random (range [1.. Proc.cpu_burst])
- Ioburst = random (range [1.. Proc.io_burst])

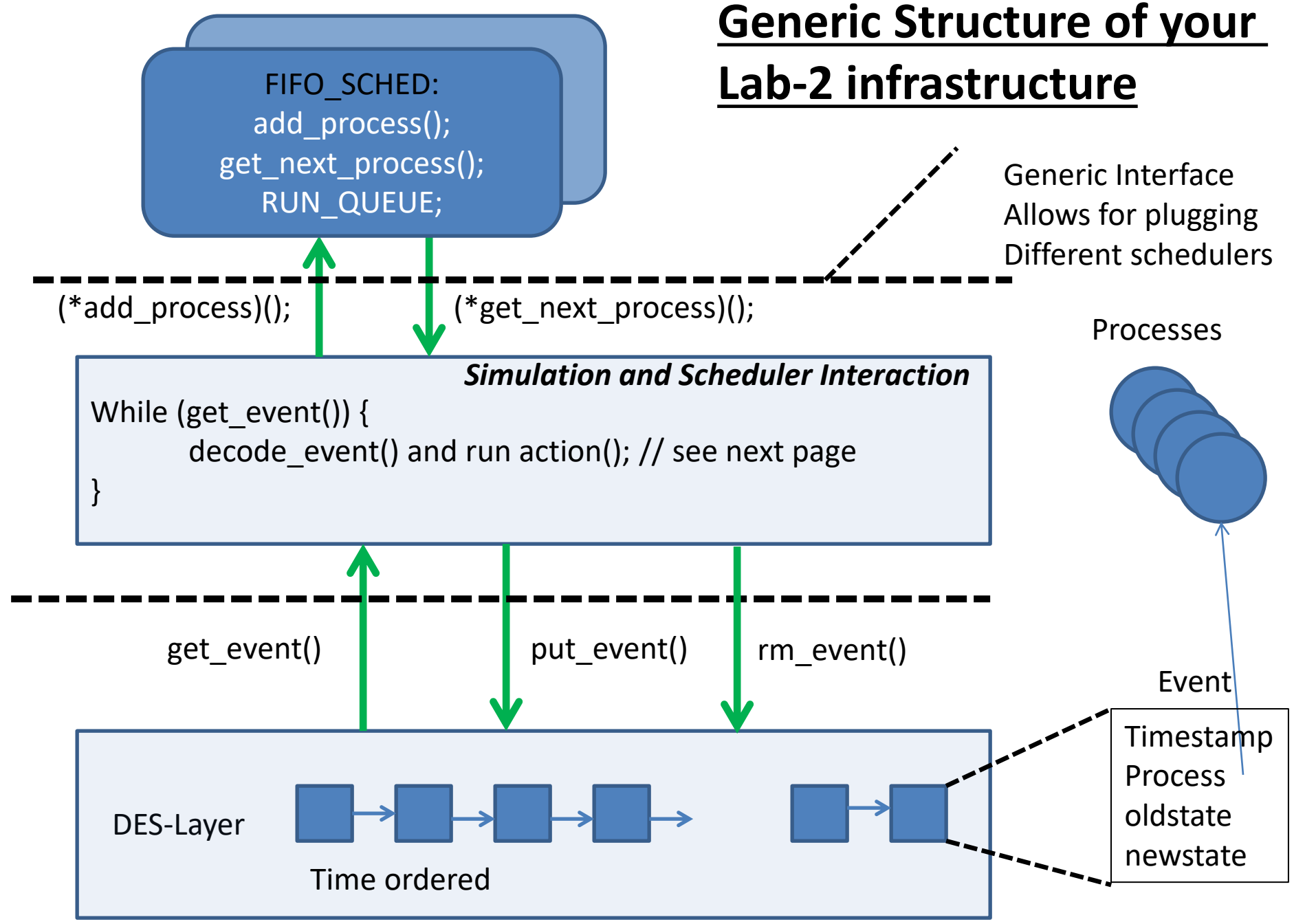
What you will generate ?

```
$ cat input4
0  200   40   90
40 100   10   40
50  20   10   10
60 200   5    20
```

```
$ ./sched -sF input4 rfile
FCFS
0000:    0  200    40    90 2 |    575    575    360     15
0001:   40  100    10    40 4 |    532    492    300     92
0002:   50   20    10    10 2 |     82     32     10      2
0003:   60  200     5    20 4 |   1174   1114    773    141
SUM: 1174 44.29 81.26 553.25 62.50 0.341
```



Generic Structure of your Lab-2 infrastructure



```
void Simulation() {
        EVENT* evt;
        bool CALL_SCHEDULER = false;
        while( (evt = get_event()) ) {
                Process *proc   = evt->evtProcess;    // this is the process the event works on
                CURRENT_TIME    = evt->evtTimeStamp;  // time jumps discretely
                int timeInPrevState = CURRENT_TIME - proc->state_ts; // good for accounting
                int transition = evt->transition;
                delete evt; evt = nullptr;   //remove current event object from Memory and don't touch anymore

                switch(transition) {  // which state to transition to?
                case TRANS_TO_READY:
                        // must come from BLOCKED or from PREEMPTION
                        // must add to run queue
                        CALL_SCHEDULER = true; // conditional on whether something is run
                        break;

                case TRANS_TO_RUN:
                        // create event for either preemption or blocking
                        break;

                case TRANS_TO_BLOCK:
                        //create an event for when process becomes READY again
                        CALL_SCHEDULER = true;
                        break;

                case TRANS_TO_PREEMPT:
                        // add to runqueue (no event is generated)
                        CALL_SCHEDULER = true;
                        break;
                }

                if(CALL_SCHEDULER) {
                        if (get_next_event_time() == CURRENT_TIME) {
                                continue; //process all events at the same time before scheduling
                        }
                        CALL_SCHEDULER = false;
                        if (CURRENT_RUNNING_PROCESS == nullptr) {
                                CURRENT_RUNNING_PROCESS = THE_SCHEDULER->get_next_process();
                                if (CURRENT_RUNNING_PROCESS == nullptr)
                                        continue;
                                // create event to make process runnable for same time.
                        }
                }
        }
}
```

./sched -v -e input_show rfile

```
ShowEventQ:  0:0  20:1

0 0 0: CREATED -> READY
  AddEvent(0:0:RUNNG):  20:1:READY ==>   0:0:RUNNG  20:1:READY

0 0 0: READY -> RUNNG cb=8 rem=100 prio=1
  AddEvent(8:0:BLOCK):  20:1:READY ==>   8:0:BLOCK  20:1:READY

8 0 8: RUNNG -> BLOCK  ib=2 rem=92
  AddEvent(10:0:READY):  20:1:READY ==>   10:0:READY  20:1:READY

10 0 2: BLOCK -> READY
  AddEvent(10:0:RUNNG):  20:1:READY ==>   10:0:RUNNG  20:1:READY

10 0 0: READY -> RUNNG cb=10 rem=92 prio=1
  AddEvent(20:0:BLOCK):  20:1:READY ==>   20:1:READY  20:0:BLOCK

20 1 0: CREATED -> READY

20 0 10: RUNNG -> BLOCK  ib=7 rem=82
  AddEvent(27:0:READY): ==>   27:0:READY
  AddEvent(20:1:RUNNG):  27:0:READY ==>   20:1:RUNNG  27:0:READY

20 1 0: READY -> RUNNG cb=7 rem=100 prio=3
  AddEvent(27:1:BLOCK):  27:0:READY ==>   27:0:READY  27:1:BLOCK
```

Input file

```
0   100 10 10
20  100 20 10
```

RESULTS OF SIMULATION

FCFS

```
0000:   0  100   10   10 2 |   234   234    89    45
0001:  20  100   20   10 4 |   226   206    77    29
SUM: 234 85.47 57.26 220.00 37.00 0.855
```

./sched -v -e input_show rfile

Input file

0 100 10 10
20 100 20 10

```
ShowEventQ:  0:0  20:1

0 0 0: CREATED -> READY
  AddEvent(0:0:RUNNG):  20:1:READY ==>   0:0:RUNNG  20:1:READY

0 0 0: READY -> RUNNG cb=8 rem=100 prio=1
  AddEvent(8:0:BLOCK):  20:1:READY ==>   8:0:BLOCK  20:1:READY

8 0 8: RUNNG -> BLOCK  ib=2 rem=92
  AddEvent(10:0:READY):  20:1:READY ==>   10:0:READY  20:1:READY

10 0 2: BLOCK -> READY
  AddEvent(10:0:RUNNG):  20:1:READY ==>   10:0:RUNNG  20:1:READY

10 0 0: READY -> RUNNG cb=10 rem=92 prio=1
  AddEvent(20:0:BLOCK):  20:1:READY ==>   20:1:READY  20:0:BLOCK

20 1 0: CREATED -> READY

20 0 10: RUNNG -> BLOCK  ib=7 rem=82
  AddEvent(27:0:READY): ==>   27:0:READY
  AddEvent(20:1:RUNNG):  27:0:READY ==>   20:1:RUNNG  27:0:READY

20 1 0: READY -> RUNNG cb=7 rem=100 prio=3
  AddEvent(27:1:BLOCK):  27:0:READY ==>   27:0:READY  27:1:BLOCK
```

line triggered by “-v”
Timestamp pid howlong:
FROM -> TO

One Event

line triggered by “-e”
event added: (time,pid,transition)
EventQ-Before ➔ EventQ-After
(should be time ordered)

./sched –v -t input_show rfile

```
0 0 0: CREATED -> READY
SCHED (1):  0:0
0 0 0: READY -> RUNNG cb=8 rem=100 prio=1
8 0 8: RUNNG -> BLOCK  ib=2 rem=92
SCHED (0):
10 0 2: BLOCK -> READY
SCHED (1):  0:10
10 0 0: READY -> RUNNG cb=10 rem=92 prio=1
20 1 0: CREATED -> READY
20 0 10: RUNNG -> BLOCK  ib=7 rem=82
SCHED (1):  1:20
20 1 0: READY -> RUNNG cb=7 rem=100 prio=3
27 0 7: BLOCK -> READY
27 1 7: RUNNG -> BLOCK  ib=9 rem=93
SCHED (1):  0:27
27 0 0: READY -> RUNNG cb=7 rem=82 prio=1
34 0 7: RUNNG -> BLOCK  ib=1 rem=75
SCHED (0):
35 0 1: BLOCK -> READY
SCHED (1):  0:35
35 0 0: READY -> RUNNG cb=9 rem=75 prio=1
36 1 9: BLOCK -> READY
44 0 9: RUNNG -> BLOCK  ib=9 rem=66
SCHED (1):  1:36
44 1 8: READY -> RUNNG cb=11 rem=93 prio=3
53 0 9: BLOCK -> READY
55 1 11: RUNNG -> BLOCK  ib=4 rem=82
```

Input file

```
0   100 10 10
20  100 20 10
```

line triggered by “-t” to show scheduler runqueue

SCHED(len): { “pid:timestamp” }*

where len is length of RQ followed by RQ entries

Current cpu-burst

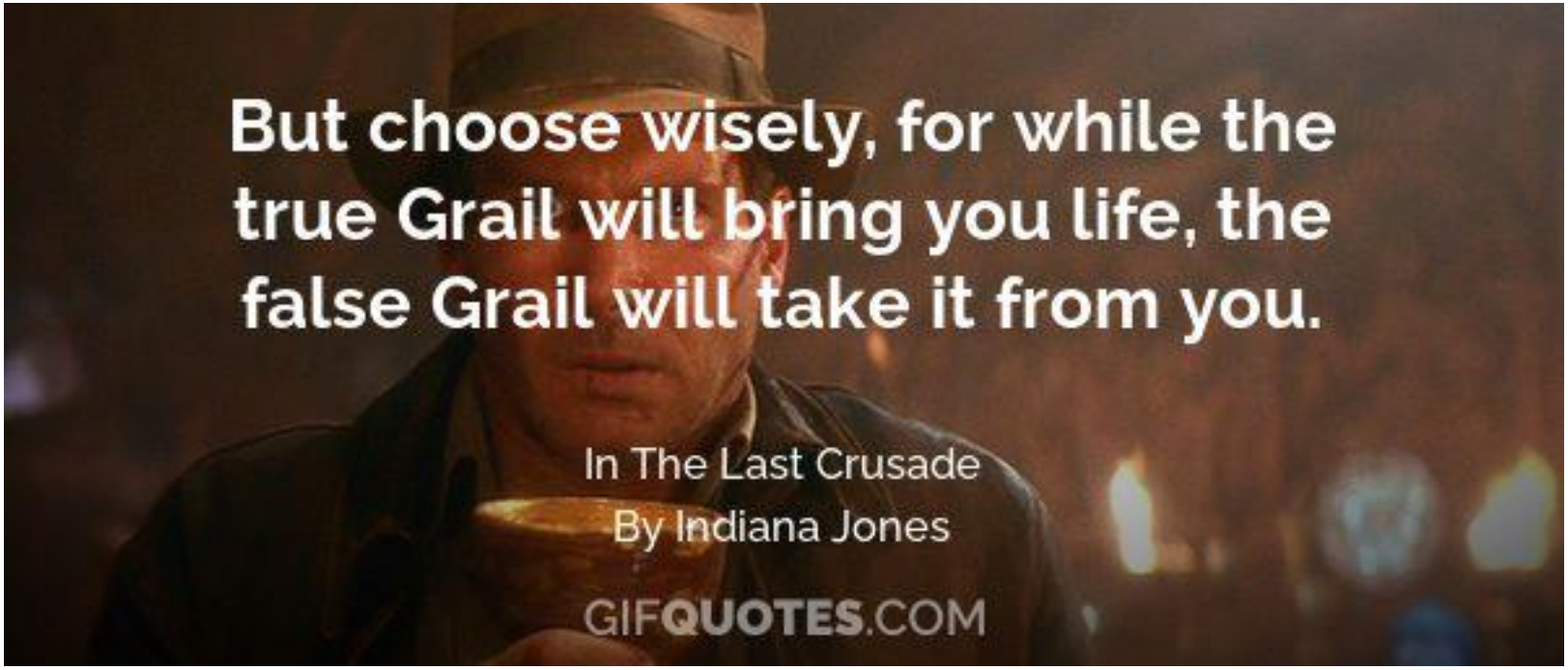
Current io-burst

Dynamic priority

Remaining cputime

Data Structures

- Schedulers are often about pooling requests and selecting:
  - Queueing , searching , indexing, mapping, priorities
  - Efficiency matters, not just effectiveness
  - Underlying data structures matter

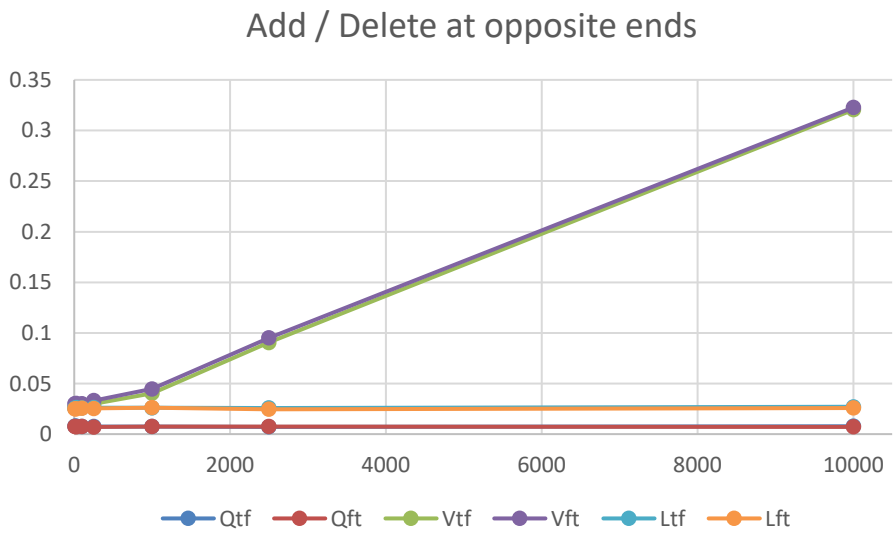


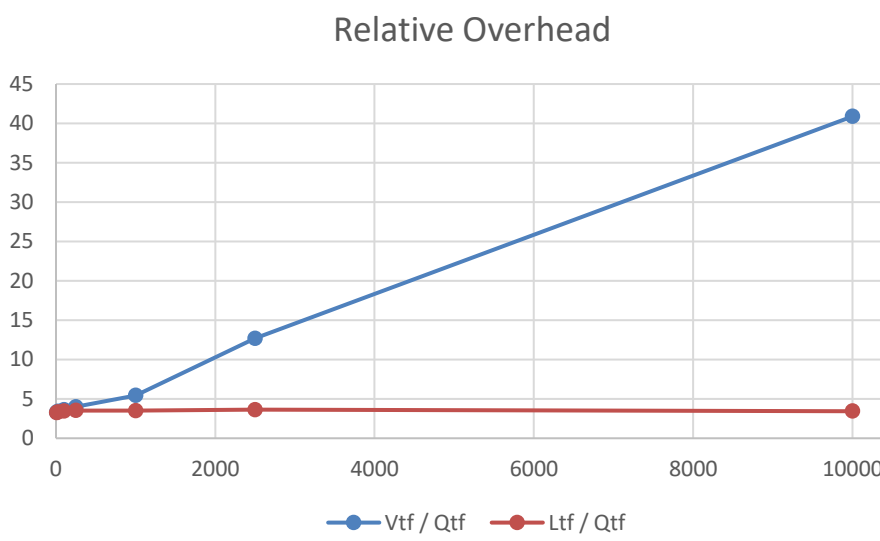
Example: Queue Semantics

- Assume existing queue size N
- Consider the following operations
  - Add front / delete tail (FT)
  - Add Tail / delete front (TF)
  - Iterate through queue with
    - C++ iterator (Iter) or C++ [] operator
- Time these operations
- Consider the following container classes:
  - Deque (Q)
  - Vector (V)
  - List (L)

Add / Delete Opposite ends

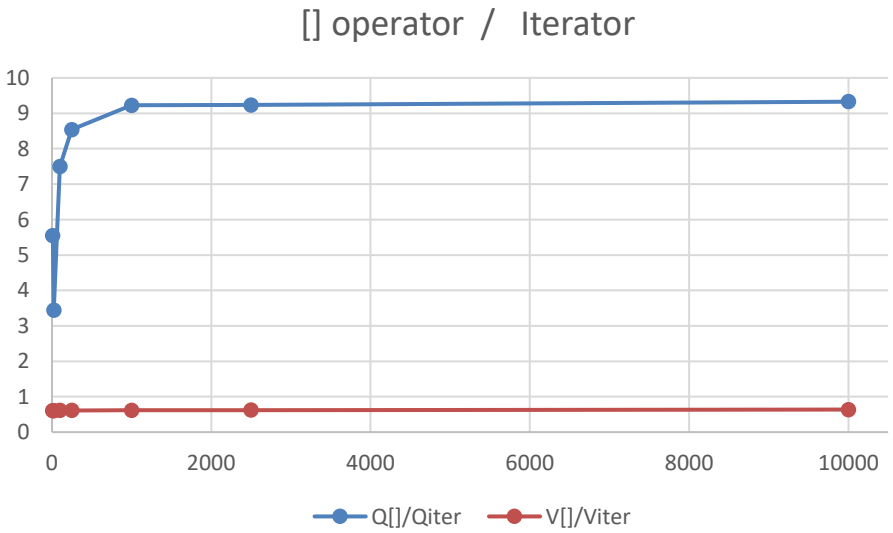
- Scale the size of the queue (x-axis = N)
- TF and FT semantics
- Why are vectors so expensive ?





Iterations

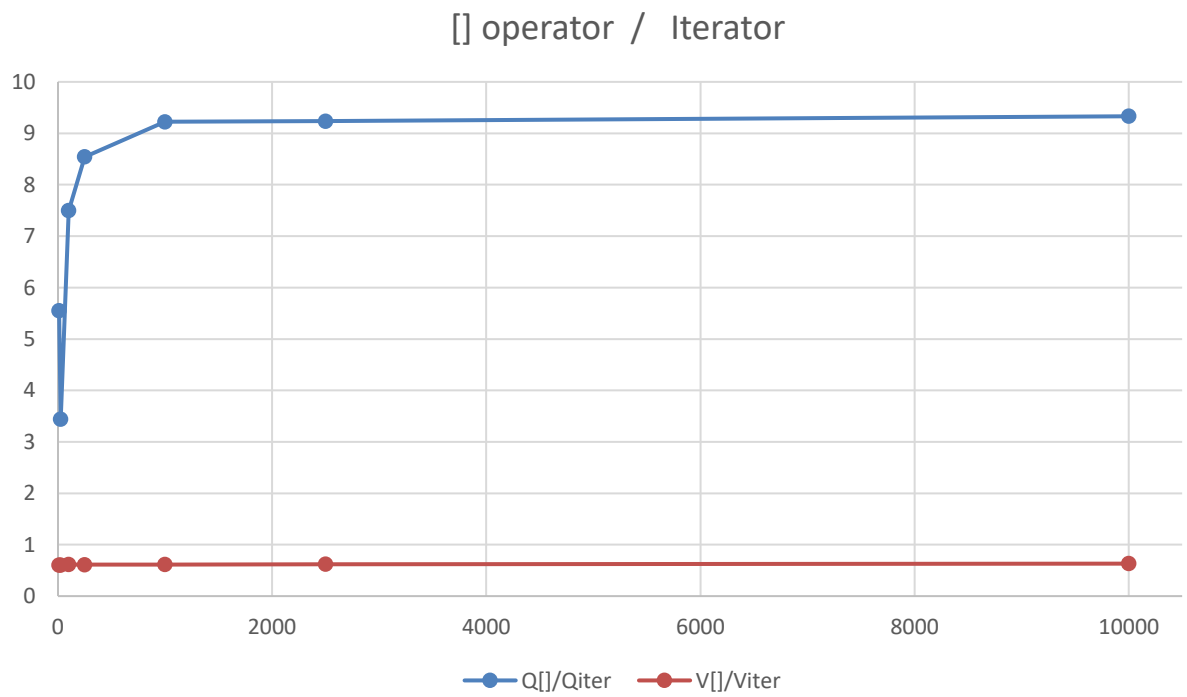
- Iterate by 2 approaches:
  iter: `for ( x : objs ) { x } ;`
  []: `for (i = 0; i<obj.size(); i++) { x }`



note: list[] does not exist

Iteration vector vs deque

- [] operator relatively expensive for deque over iterator, but not for vector



Iteration vector vs deque vs list

- Avoid [] operator on queues
- Think about underlying data structures how something would be implemented and think about cost of operations.