Traffic Shaping System Calls Using Threading by Appointment

Christoph Kirsch
University of Salzburg

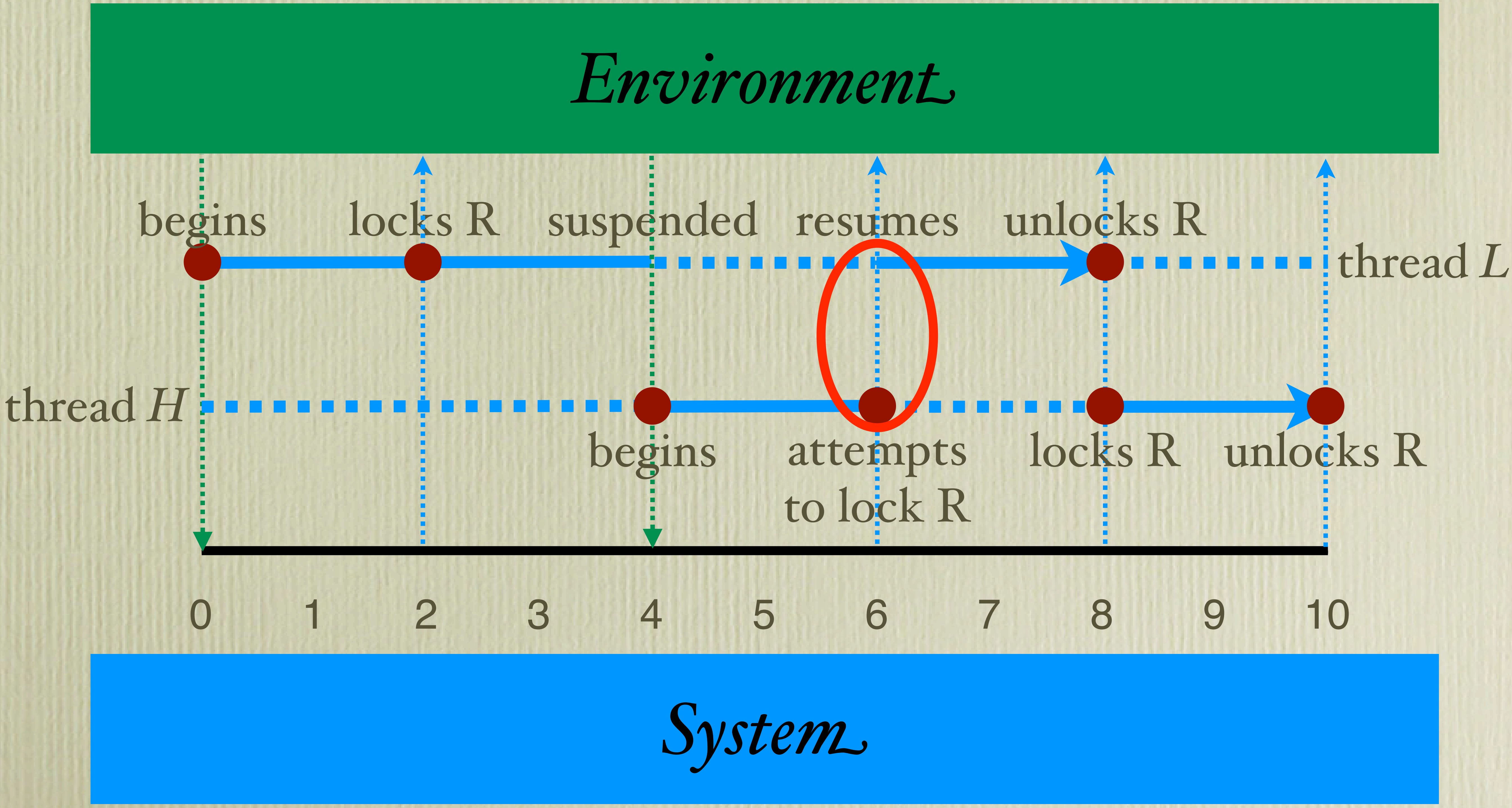
Joint work with Harald Röck

Contributions

1. Threading by Appointment (TAP):
   - ➡ A concurrent programming model that combines the convenience of *automatic stack management* (threads) with the efficiency of *system call queueing* (events).
2. A TAP policy for *traffic shaping* system calls.

Threading by Appointment: Mechanism

Example: Locking

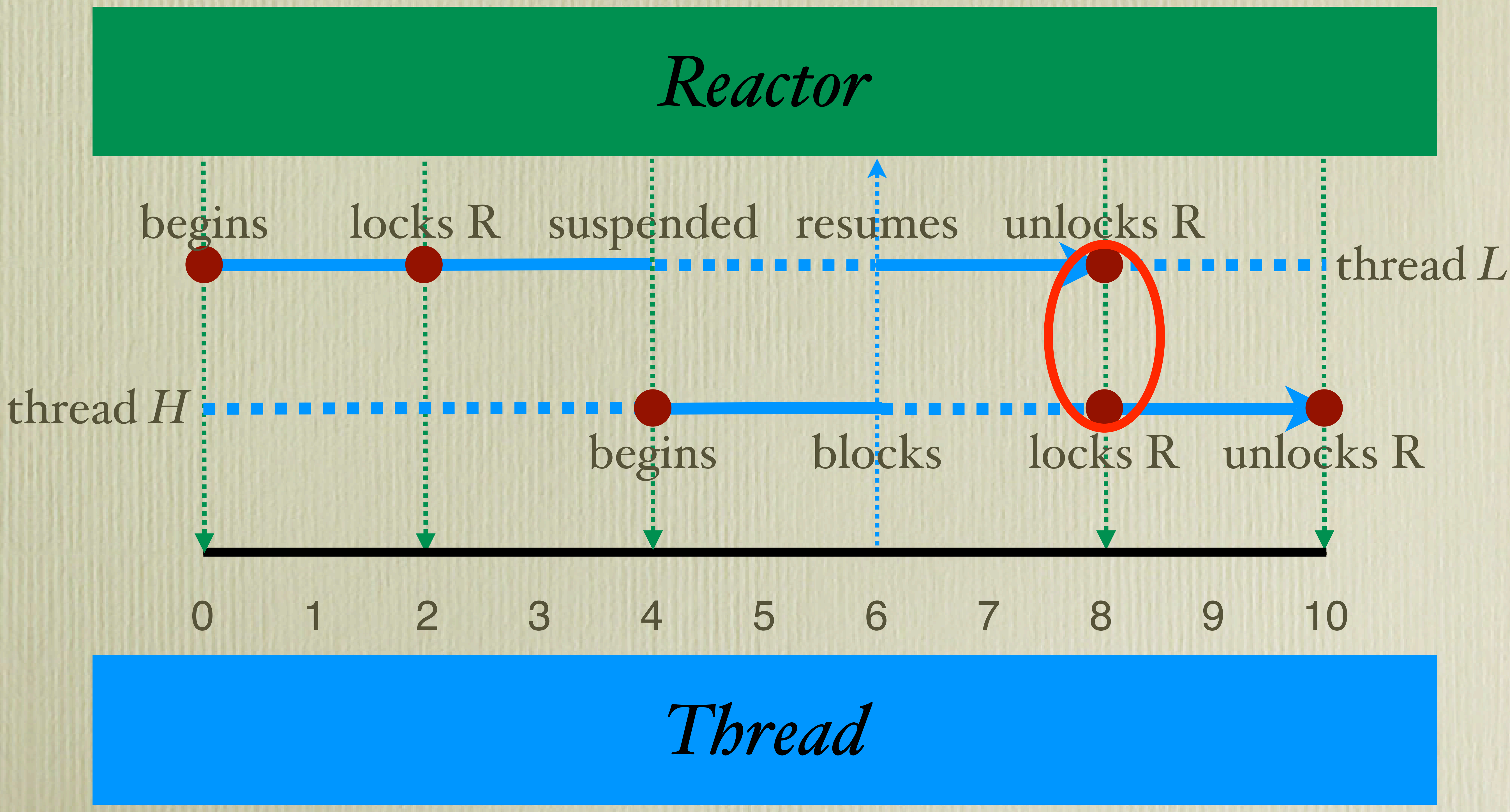


Solutions

1. Solution: *thread queueing*
    - with priority inheritance or similar techniques if priorities are present.
2. Solution: *system call queueing*

➡ enables traffic shaping of system calls (system call = packet).

Example: TAP Locking



System Call Queueing

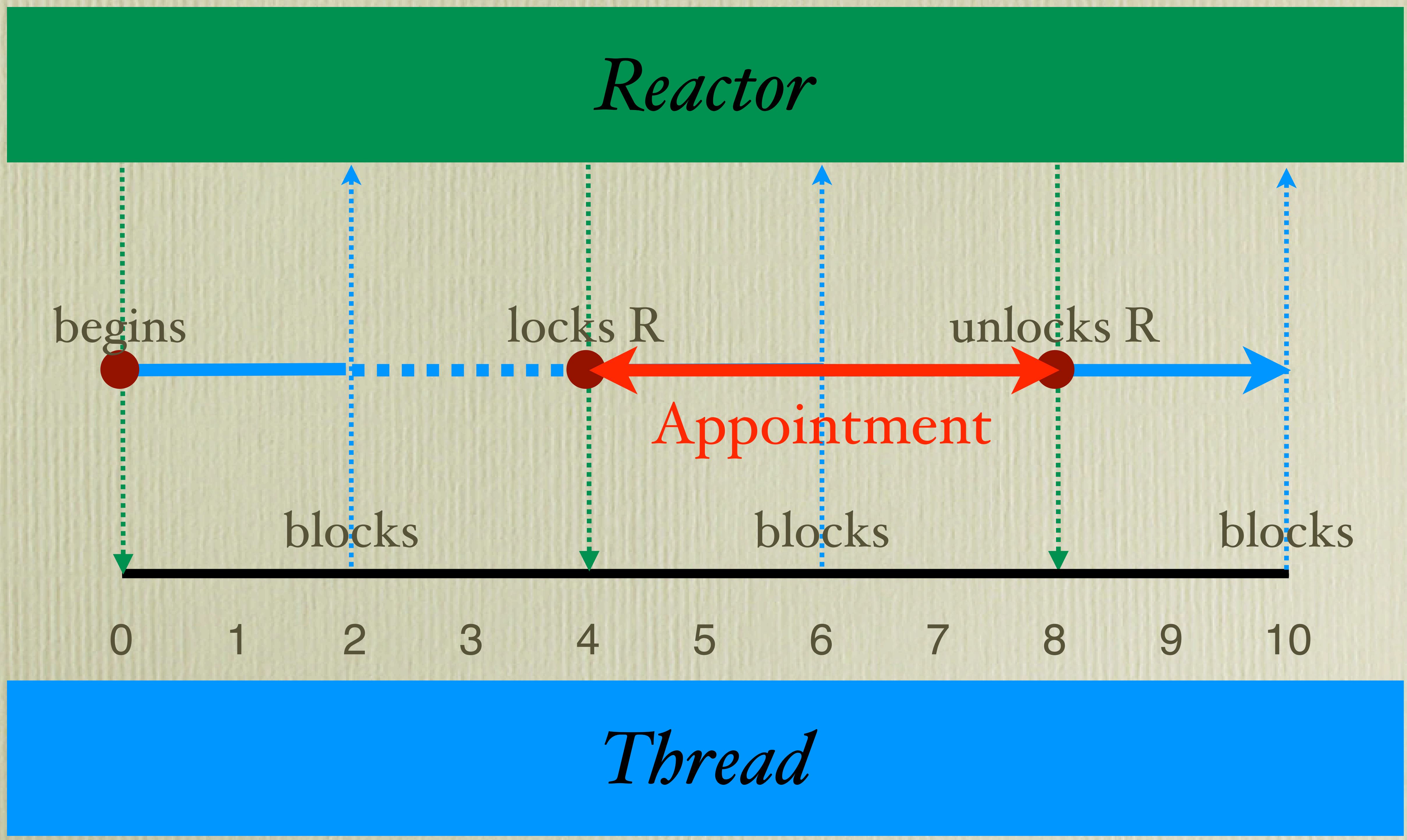
Appointment

- A TAP thread must have an *appointment* before invoking a system call.

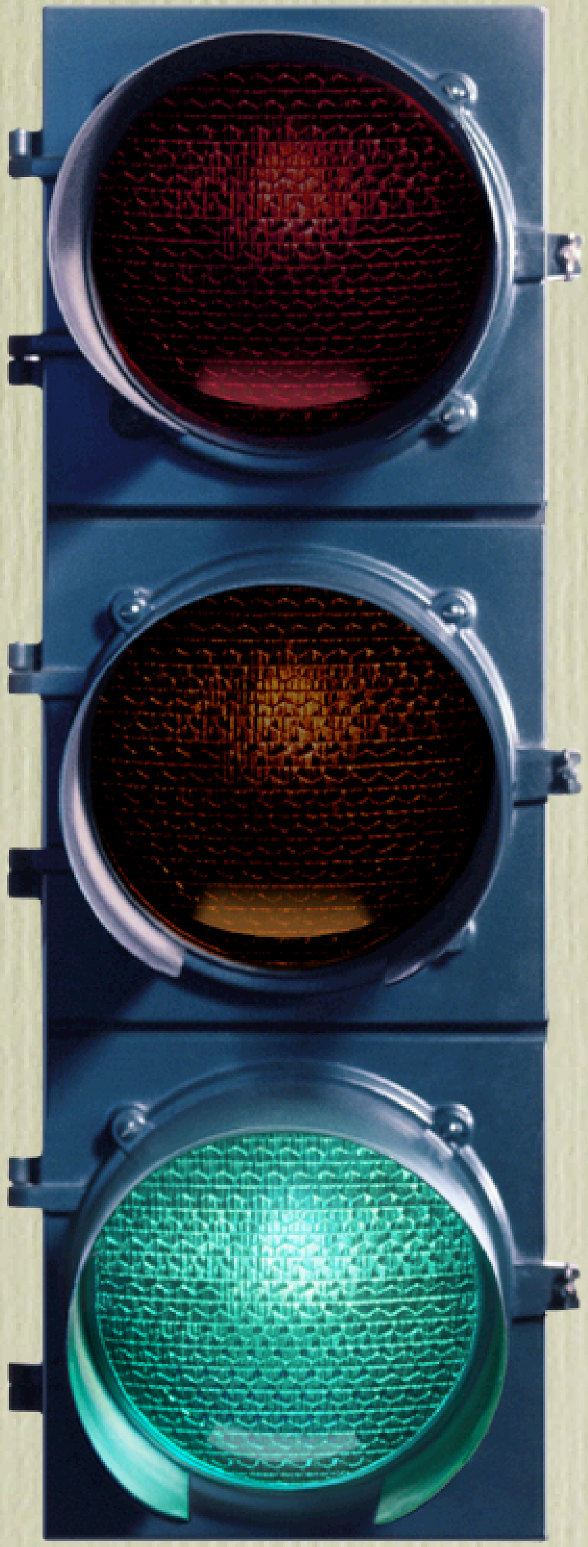
- When a TAP thread *attempts* to invoke a system call, the thread is *blocked* until the time of the appointment and only then gets to invoke the system call.

- Appointments can be made by the thread and the TAP runtime system (only the latter is implemented).

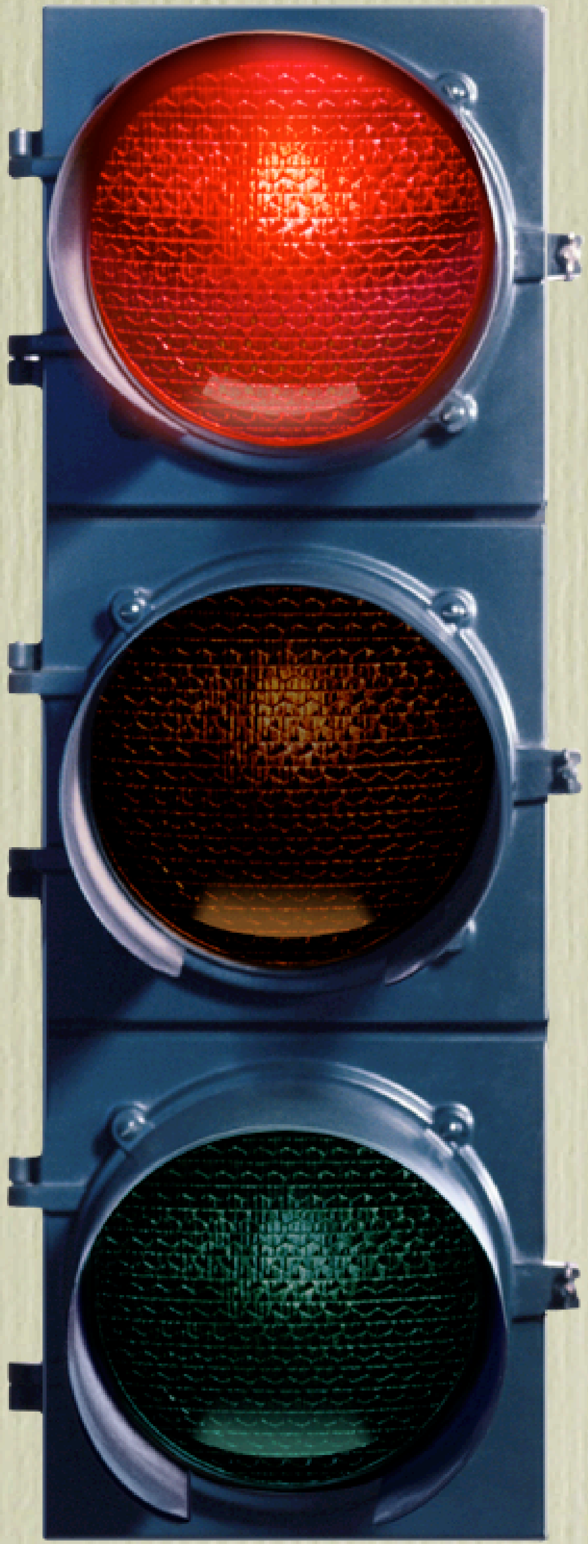
Example: Appointment



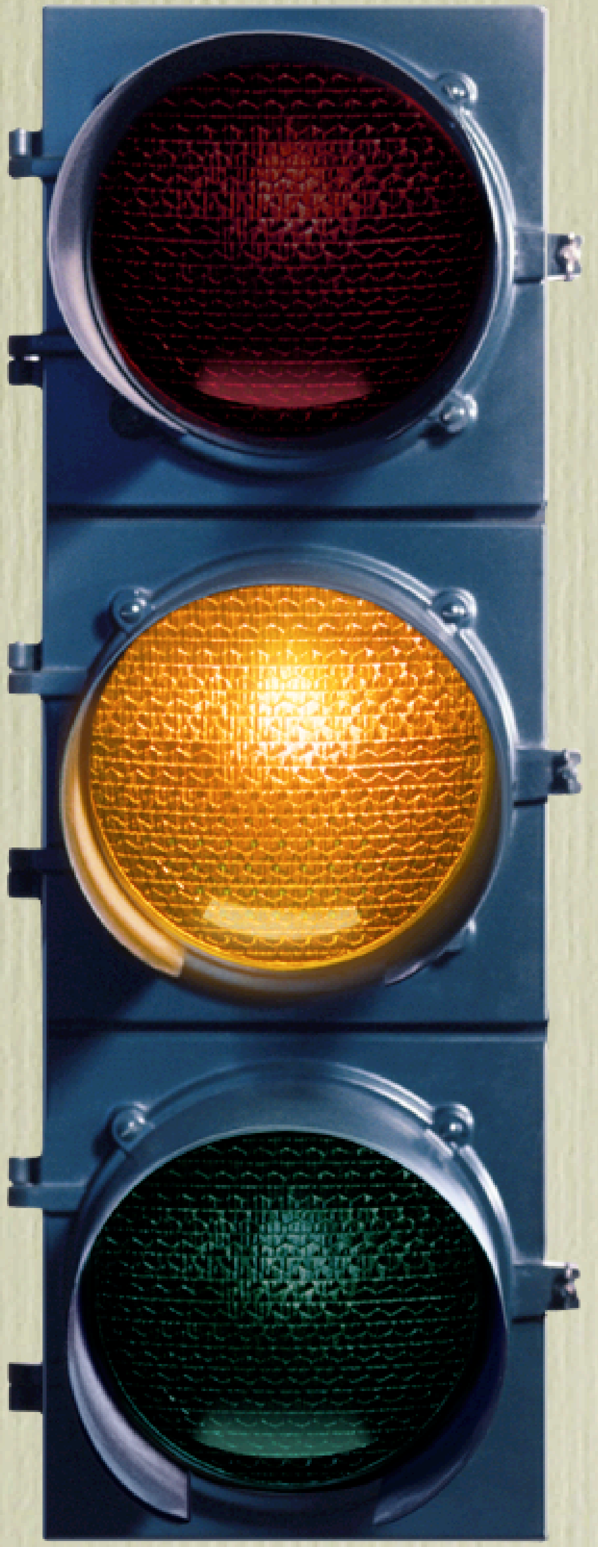
Running Thread



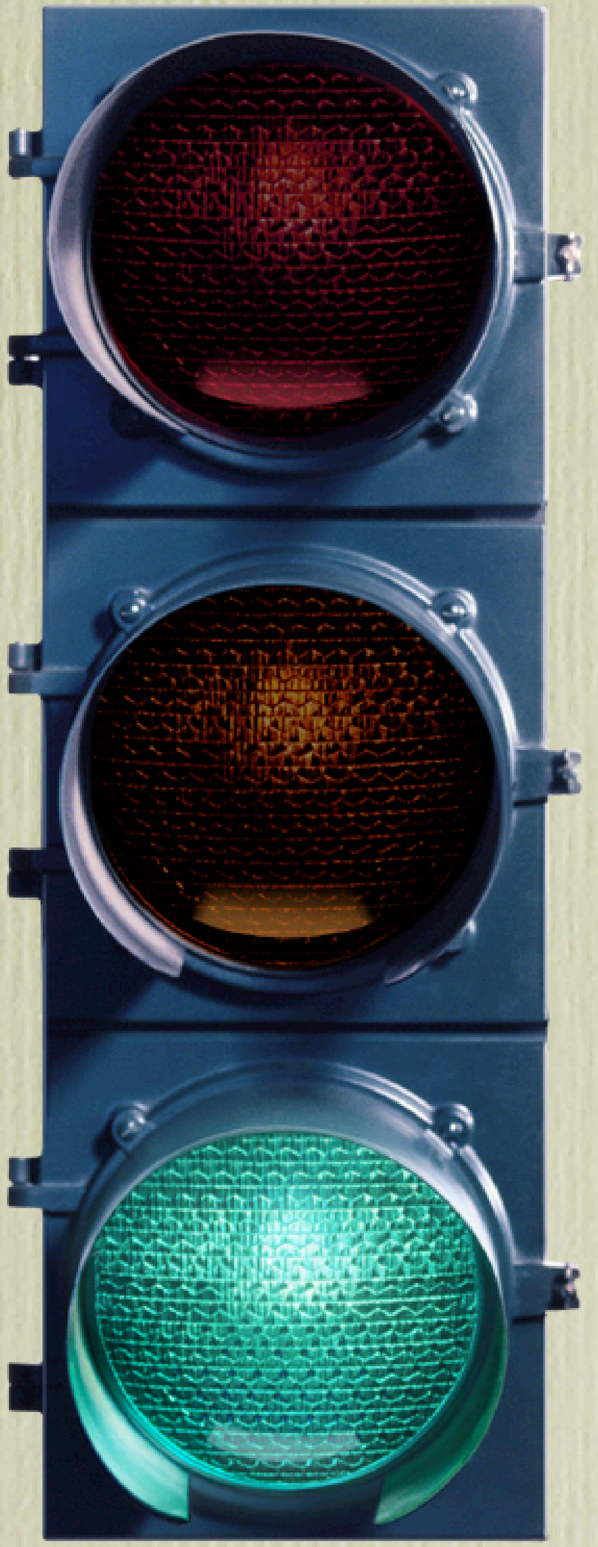
Blocked Thread



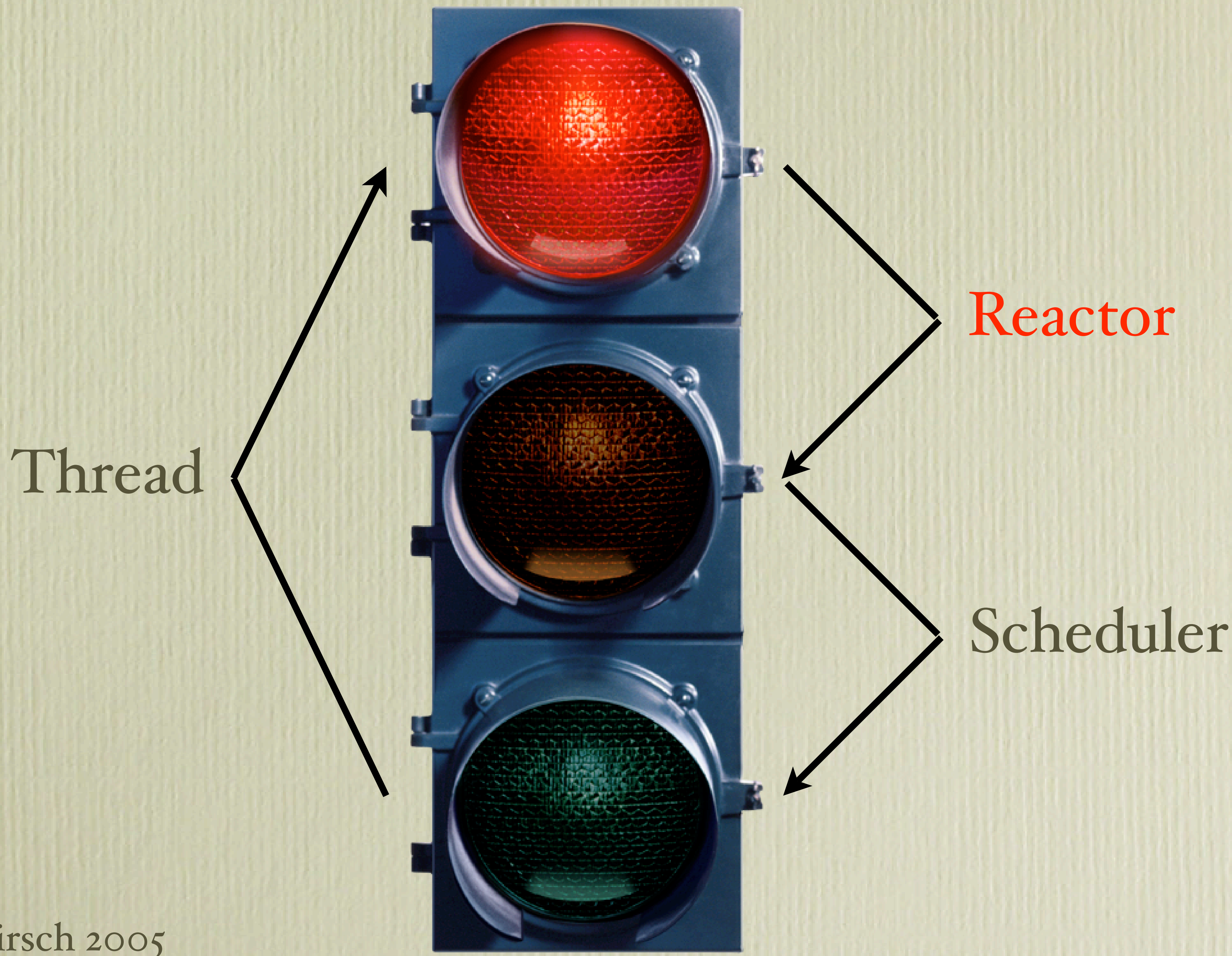
Released Thread



Running Thread



State Transitions



Reactor

1. blocks thread upon *attempt* to invoke system call.

2. releases thread to scheduler at *beginning* of appointment (in current implementation: invokes system call on behalf of thread).

3. blocks thread upon *return* from system call.

4. releases thread to scheduler at *end* of appointment.

Correctness

- We say a thread has *broken* its appointment if the thread is not blocked at the beginning and end of the appointment.

➡ In our implementation, threads cannot break appointments.

System Call Queueing

- The reactor maintains multiple queues of system calls called *calendars* and determines the exact order and time of system calls.

➡ Threading by Appointment enables *system call queueing*

Observation

➡ Threading by Appointment is orthogonal to automatic stack management, i.e., it might as well be used in *event-based* systems.

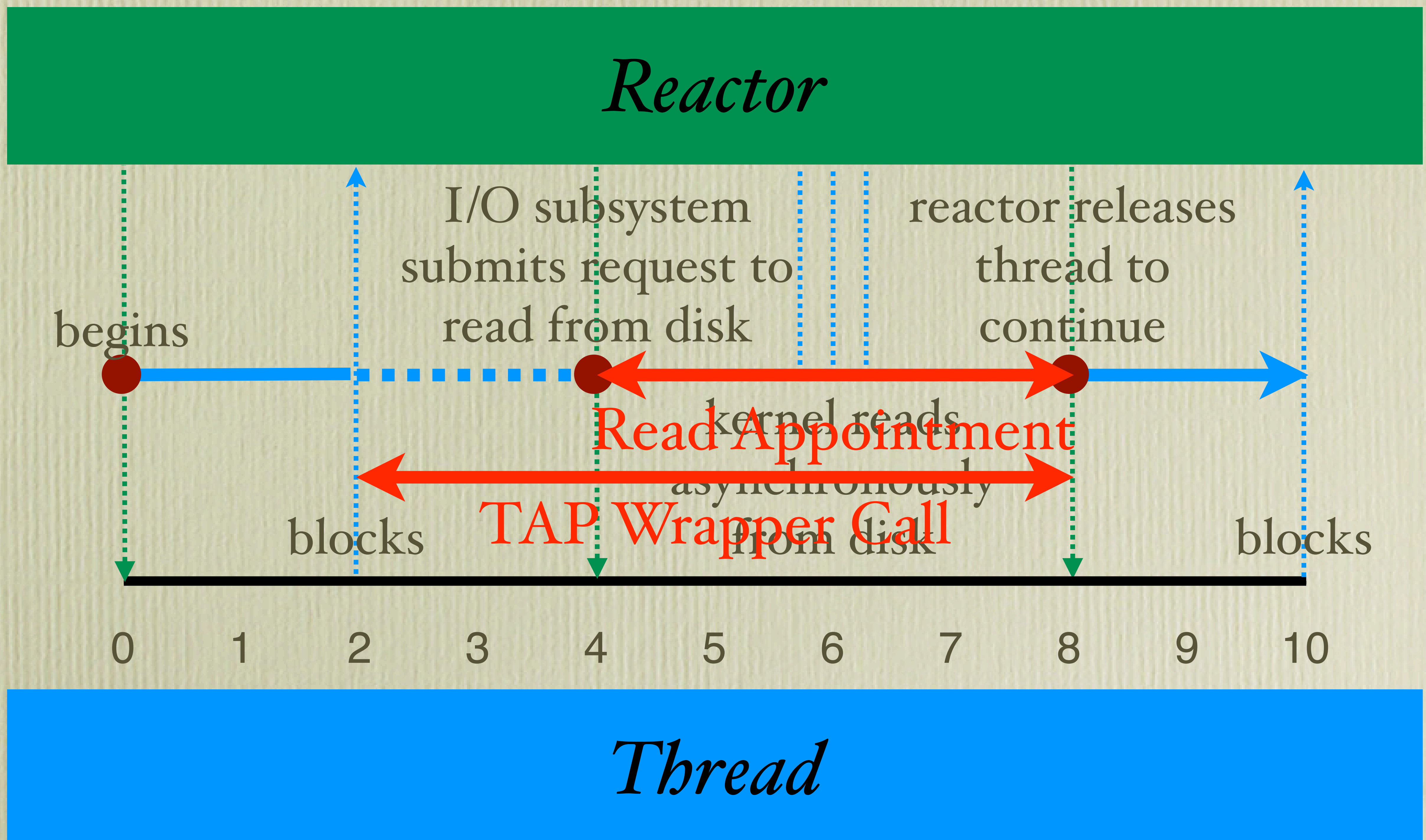
I/O

The TAP I/O Subsystem

- The TAP I/O subsystem uses *nonblocking* network calls and *asynchronous* disk calls.

➡ How does the subsystem map nonblocking and asynchronous I/O calls to TAP?

Example: Disk Read

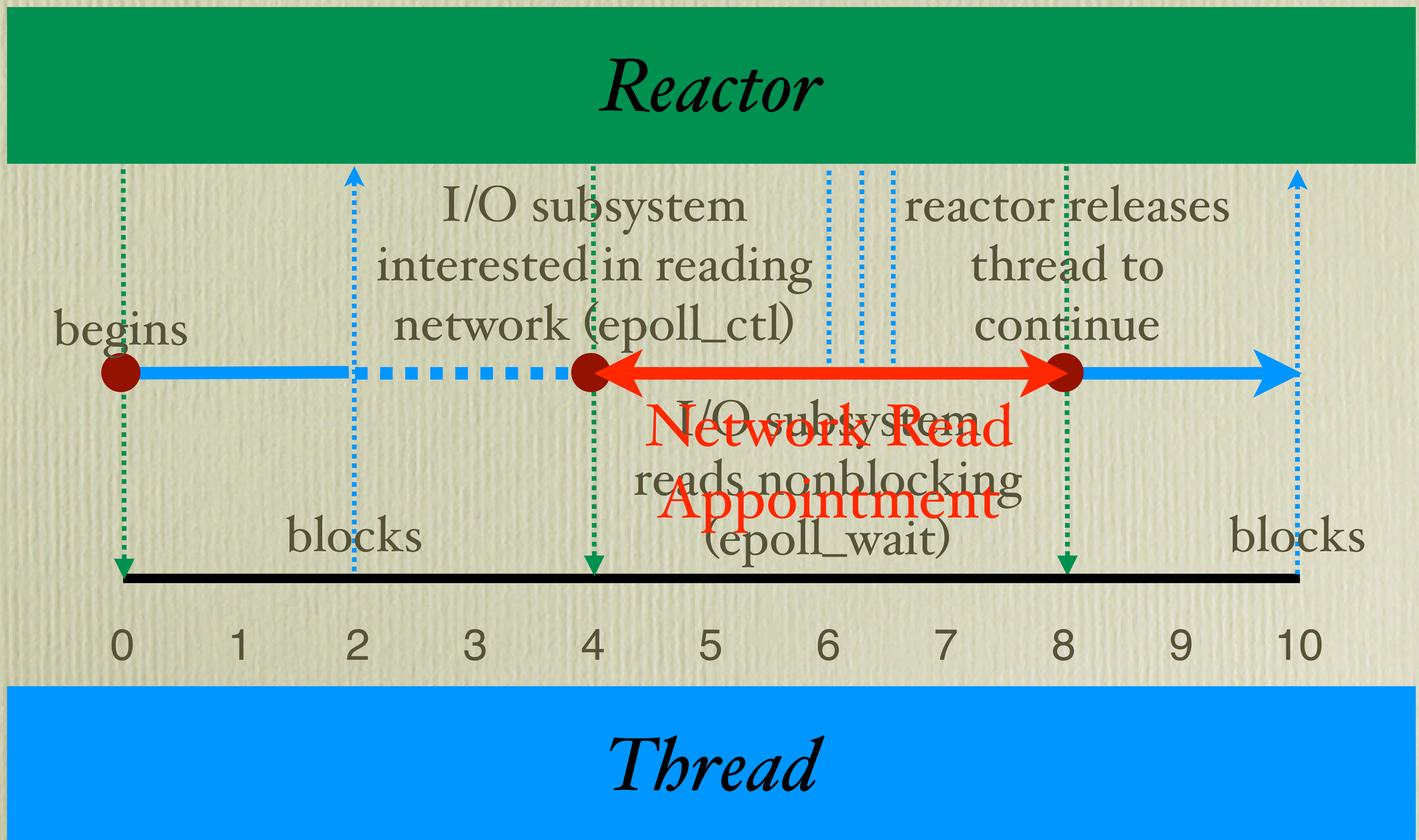


Correctness

- We say a resource has *broken* its appointment with a thread if the resource was not available during the appointment.

➡ In our implementation, resources cannot break appointments.

Example: Network Read



PL vs. OS

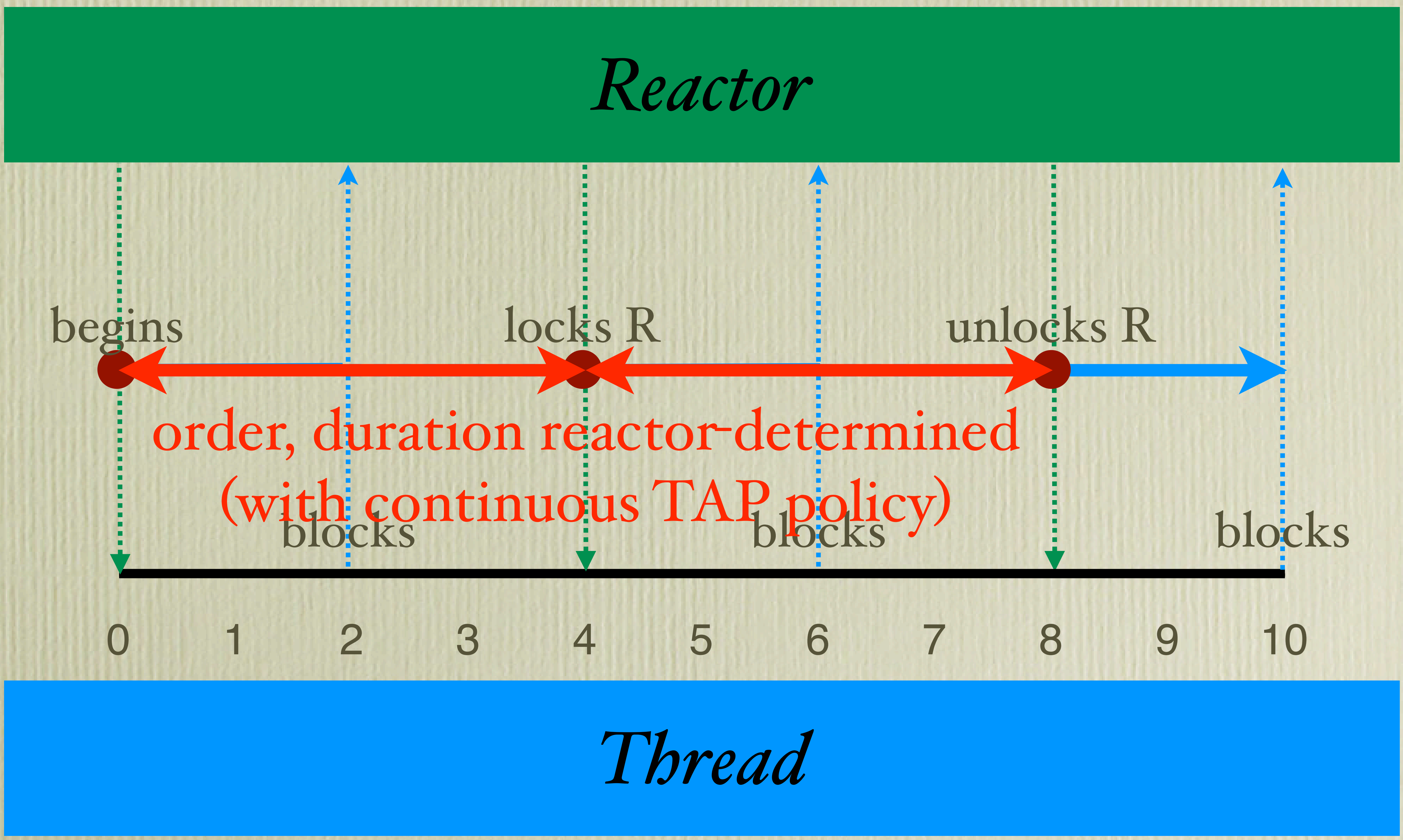
- TAP mechanism *separates* concurrency model (PL) from implementation model (OS).
- TAP policies may focus on PL, OS, or both.
- PL example: we say a TAP policy is *order-preserving* if it guarantees that the relative order of system calls of different threads is preserved under any system performance scenario (load, speed, scheduler...).
- OS example: traffic shaping system calls.

Threading by Appointment: Policy

TAP Policy

- A TAP policy consists of:
  1. an appointment *strategy*.
  2. an appointment *clock*.
- The appointment strategy determines the *order* of appointments (*insertion* into calendar).
- The appointment clock determines the *time* of appointments (*deletion* from calendar).

When Make Appointments?



Continuous TAP Policy

- We say a TAP policy is *continuous* if it guarantees that every TAP thread always has at least one appointment (TAP threads with multiple appointments are future work).

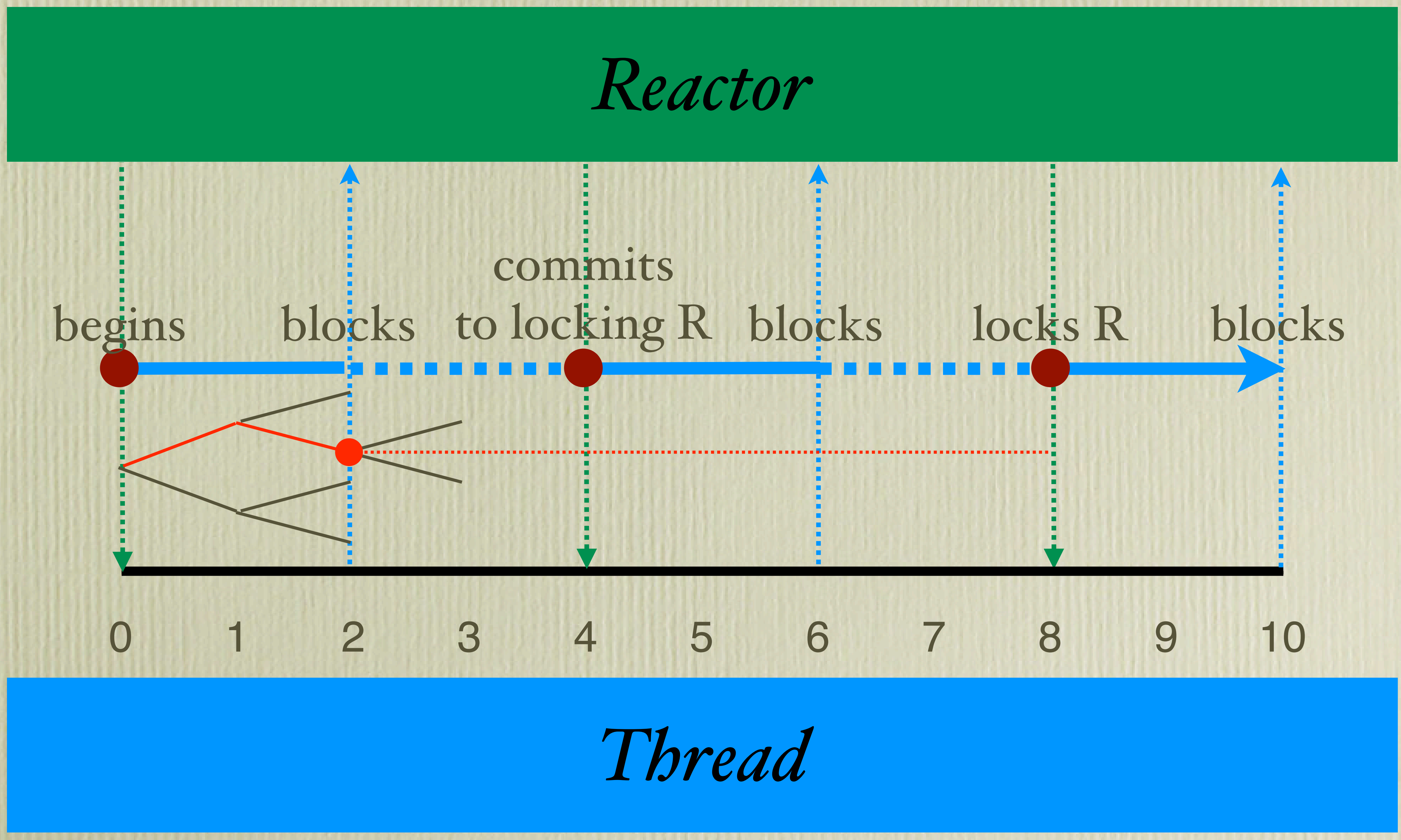
➡ At the end of an appointment, a new appointment has to be made.

Multiple Calendars

- The reactor maintains multiple calendars for network and disk (and memory, not implemented yet).

➡ How does a TAP thread make an appointment for a system call that it does not know yet?

Commit Appointment



Predicting System Calls

1. Runtime System: dynamic analysis?
    - our implementation: commit @ system call.
    - enables POSIX-compliant interface.
2. Compiler: static analysis? e.g., Capriccio!
3. Programmer: new PL constructs?

Traffic Shaping System Calls

Traffic Shaping...

- ...controls volume, throughput, and latency of network traffic, using:
- queueing disciplines such as:
    - the *leaky-bucket* algorithm (creates fixed transmission rate on varying flows).
    - the *token bucket* algorithm (allows bursts while limiting average transmission rates).
- classification schemes: *interactive* vs. *bulk* traffic.

Traffic Shaping System Calls

- system call = packet
- appointment strategy + appointment clock = queueing discipline
- thread behavior = classification scheme

Queueing Discipline

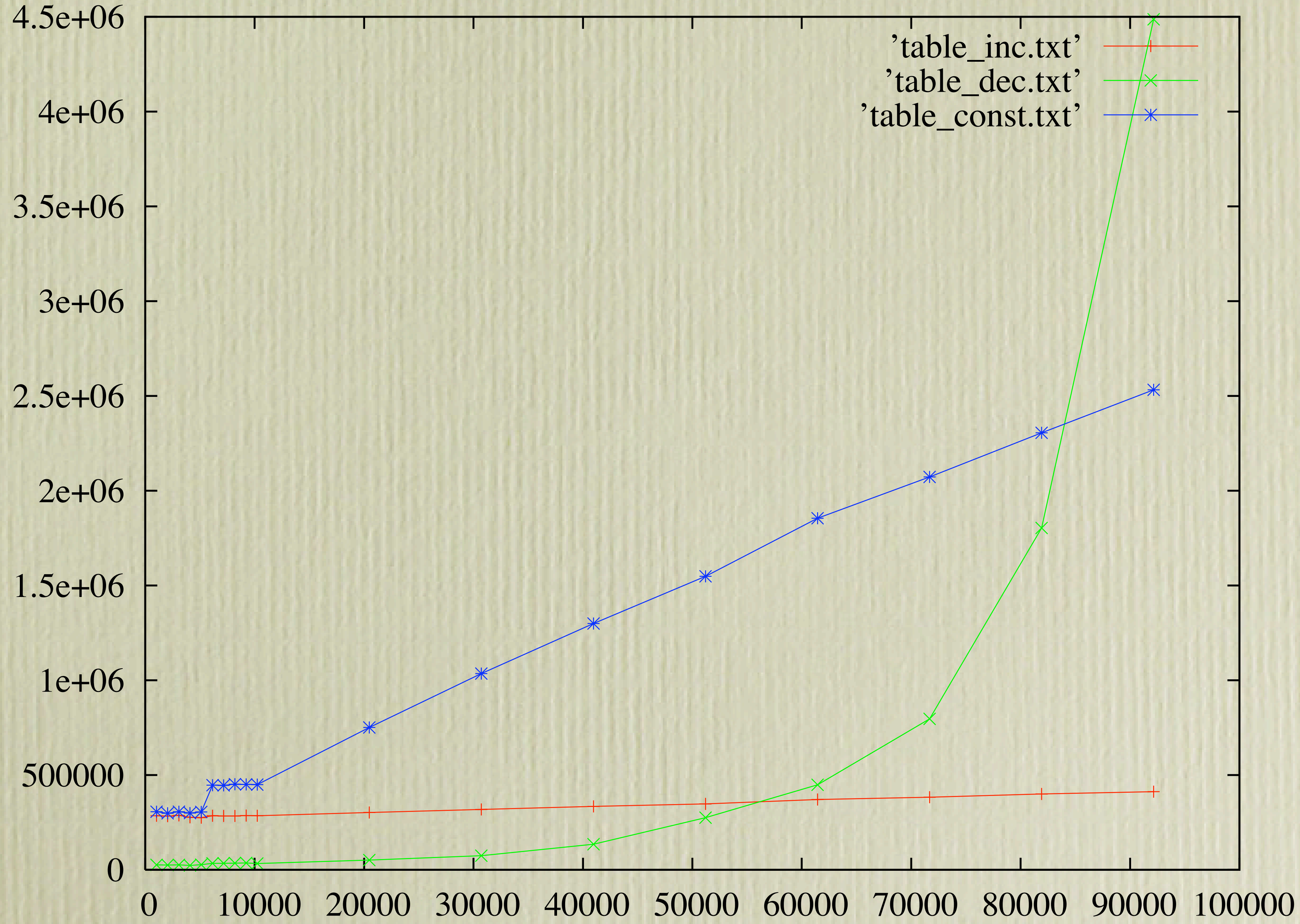
- Appointment strategy:
  - three prioritized, classful queues called CPU, NET, and DISK.
- Appointment clock:
  - ticks whenever all *next-appointed* threads are blocked and their I/O is ready (thus broken appointments are not possible).
  - round-robin CPU, NET, and DISK (ratio!).

Classification Scheme

- Thread behavior:
  - accept on network resets to *highest* priority.
  - read/write on network/disk *lower* priority.

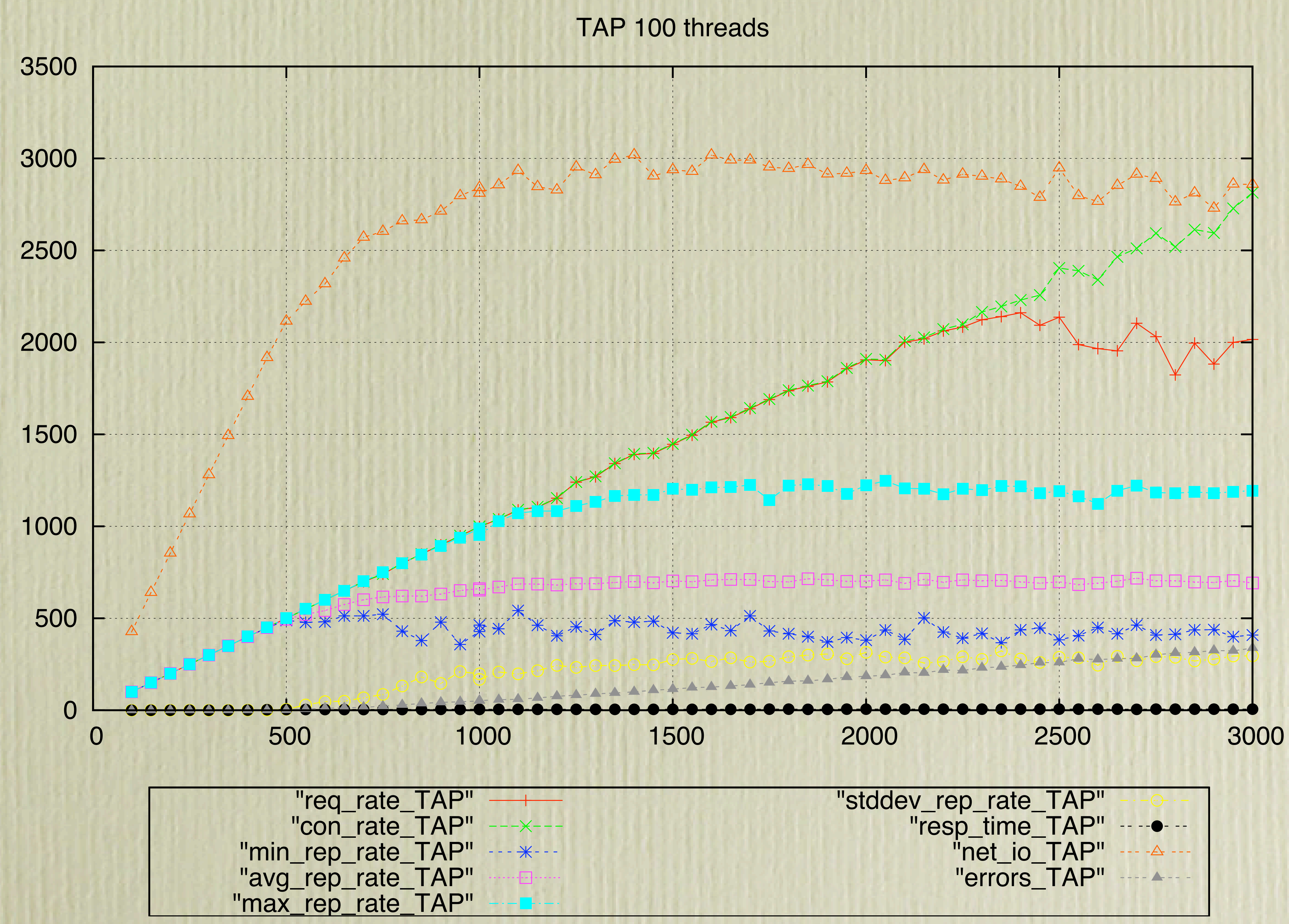
➡ Improves latency of interactive threads.

Latency



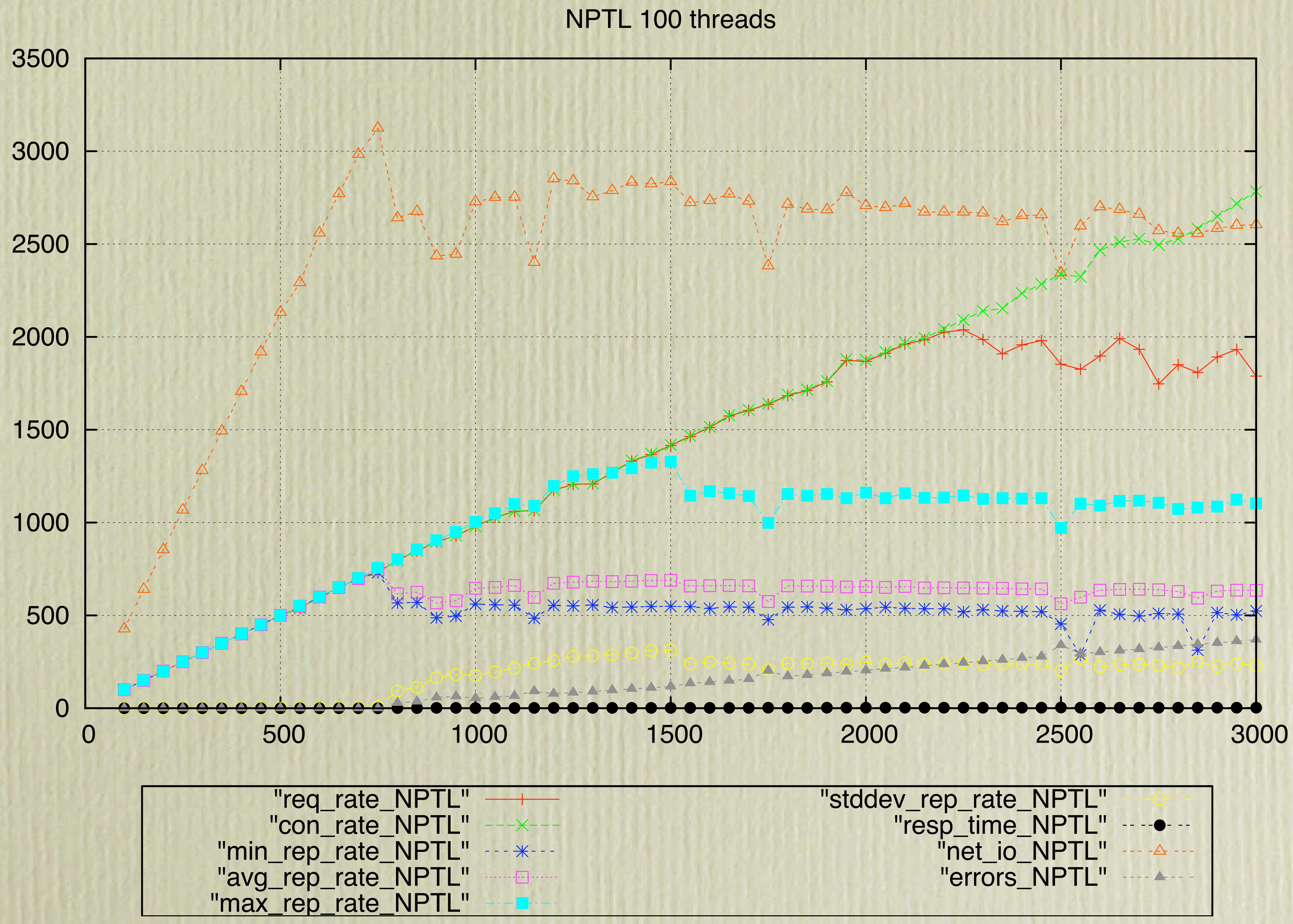
4.5e+06
4e+06
3.5e+06
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
'table_inc.txt'
'table_dec.txt'
'table_const.txt'

Throughput



TAP 100 threads
3500
3000
2500
2000
1500
1000
500
0
0
500
1000
1500
2000
2500
3000
"req_rate_TAP"
"con_rate_TAP"
"min_rep_rate_TAP"
"avg_rep_rate_TAP"
"max_rep_rate_TAP"
"stddev_rep_rate_TAP"
"resp_time_TAP"
"net_io_TAP"
"errors_TAP"

Throughput: NPTL



Thank you