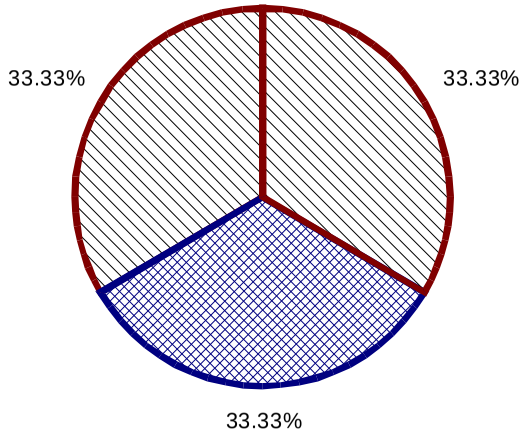
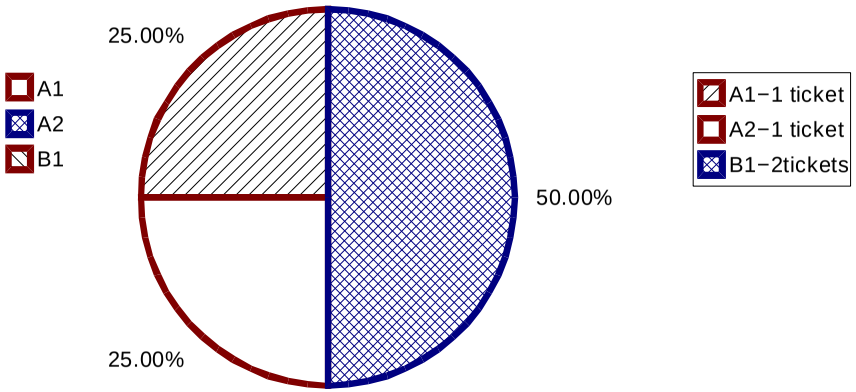
Time Sharing vs. Proportional Share Scheduling

CPU usage
w/ default (time sharing) schedulers



CPU usage
w/ proportional share schedulers



- fine-grained ajustment of proportions
  * global tickets for users
  * local tickets for every thread/process

Proportional Share Scheduling

Lottery vs. Stride Scheduling:

- Lottery Scheduling
  - * randomly pick a ticket and hereby the thread
  - * "instantaneously fair"
  - * chance of being picked is probabilistic and proportionate to number of tickets held
  - \+ easier to implement
- Stride Scheduling
  - \+ next pick predictible
  - \- more computation intensive

Current Solaris Scheduling Schemes

- round robin with priority queues

\+ fast, efficient, "simple data structures"

\+ multiple scheduling classes

RT Real Time - highest priorities

SYS System - e.g., page daemon

IA Interactive - e.g., child processes of window manager

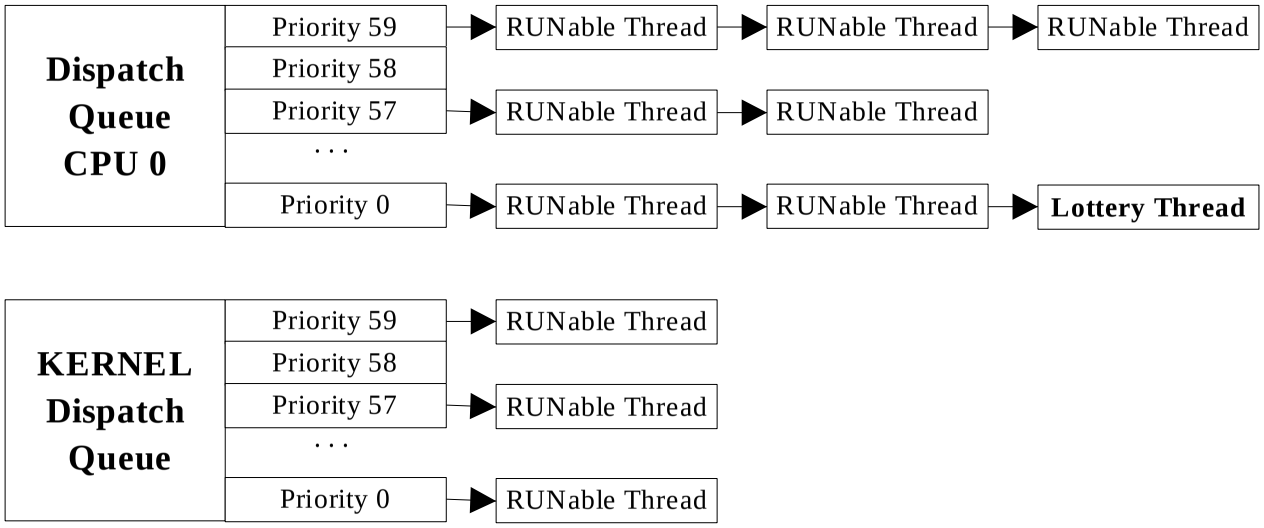
TS Time Sharing - lowest priorities

\- (in same priority/class) every thread is treated the same

\- low-priority threads may starve

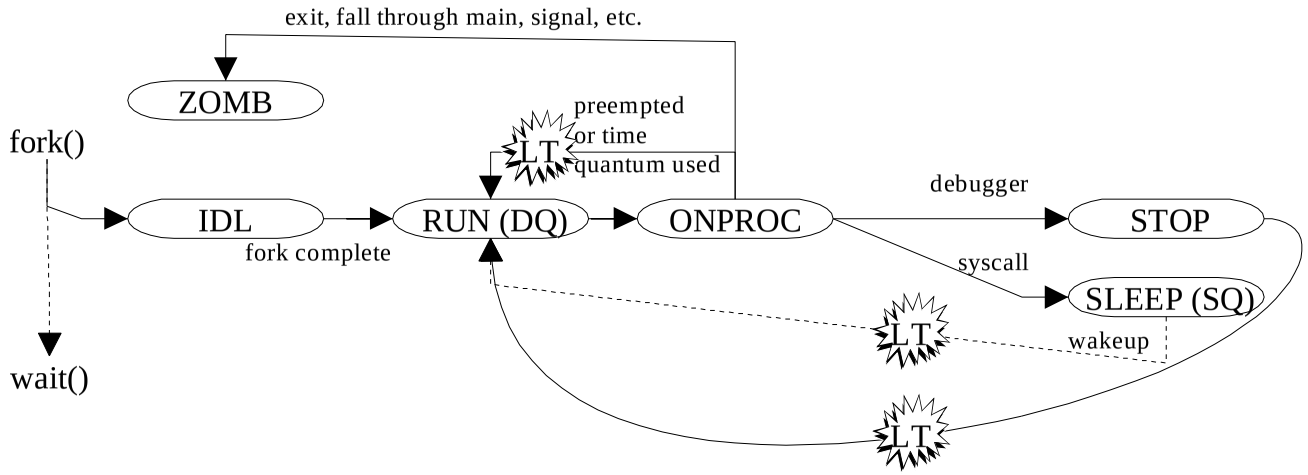
Kernel Internals: Dispatch Queue

- holds linked lists of runnable threads for all priorities
- one for each CPU, one for the kernel (undispatched threads)
- dispatcher does load balancing over CPUs



Kernel Internals: Threads

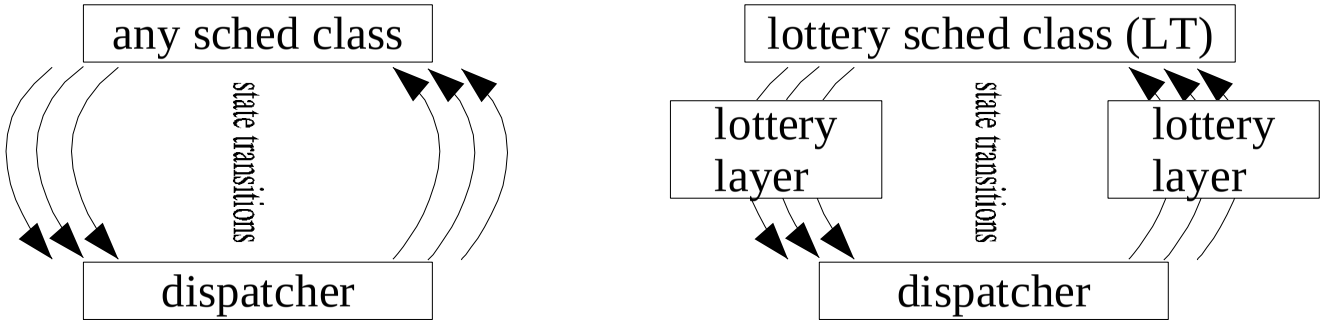
lifecycle:



- scheduler decides priority/time slice for thread
- dispatcher picks next thread from dispatch queue
- threads hold class specific data
- class specific functions for fork, preempt, sleep...

Lottery Threads (LT): Our Approach

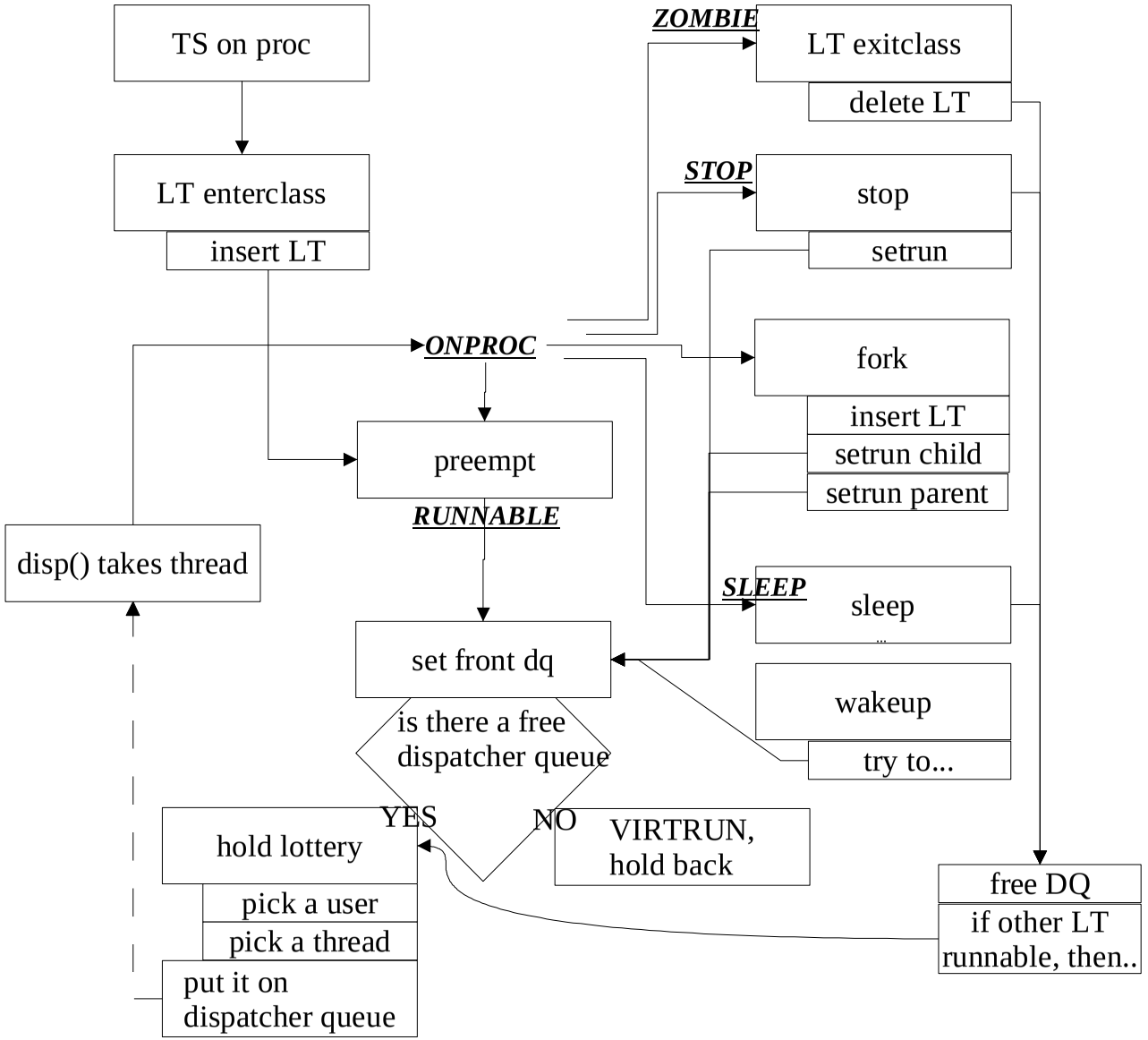
- LT scheduler is a module
- work within current framework
- minimize/avoid changes to other kernel parts (e.g., dispatcher)

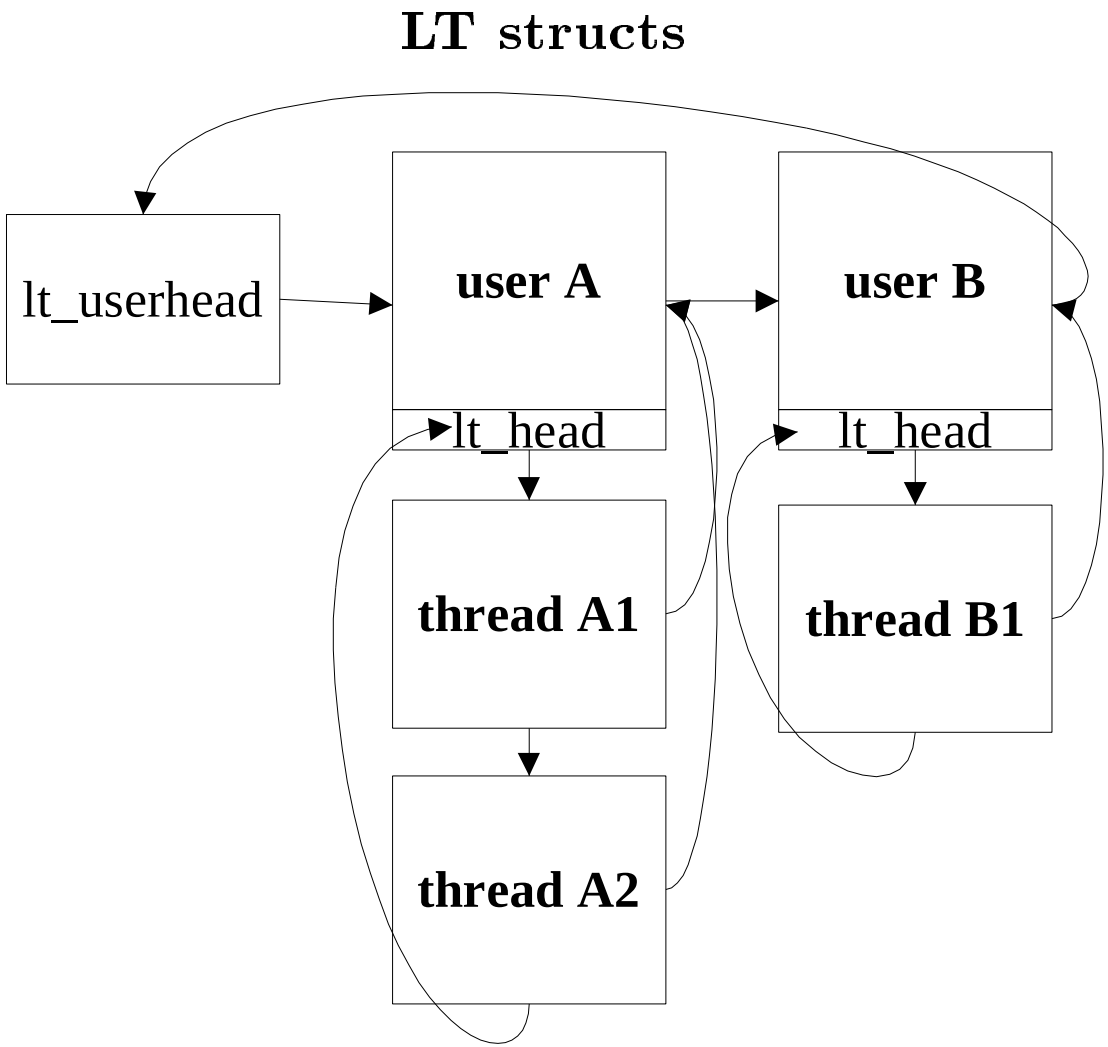


Lottery Threads (LT): Implementation

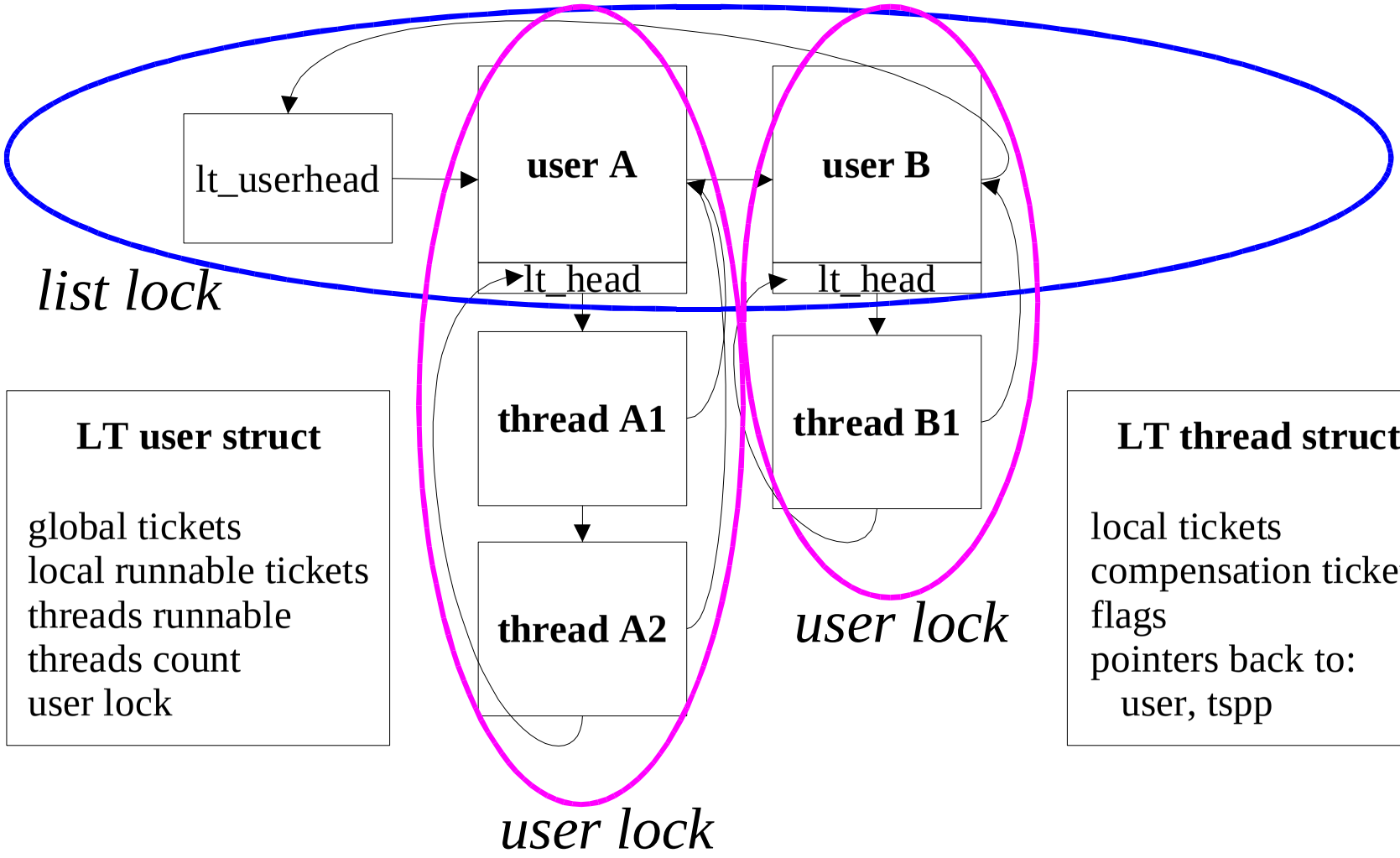
- like IA, LT is only a special case of TS
- scheduler only puts one LT thread/CPU on dispach queues
- dispatcher only finds one LT thread/CPU to run
- LT sched class holds lottery to put next thread on DQ
- fixed priority 0 (lowest), time slice 40ms

LT Lifecycle




LT structs
lt_userhead
user A
user B
lt_head
lt_head
thread A1
thread B1
thread A2

LT structs with locks



LT user struct

global tickets
local runnable tickets
threads runnable
threads count
user lock

LT thread struct

local tickets
compensation tickets
flags
pointers back to:
user, tspp

TNF probes: an extract

1 CPU, 4 Threads, 2 Users (1:3)

```
ts_preempt                tp_pid: 916 lttp_uid: 101640 ticks: 3
show_all_lt_general   __pid: 916 ondq: 0 global_tickets: 200
show_all_lt_user      __uid: 39809 global_tickets: 100 num_thr: 1 runbl: 1
show_all_lt_thread    _X__pid: 917 tp: 0x30002200560 ENQ: 0 VIRTRUN: 1
show_all_lt_user      __uid: 101640 global_tickets: 100 num_thr: 3 runbl: 1
show_all_lt_thread    _X__pid: 916 tp: 0x300022002c0 ENQ: 0 VIRTRUN: 1
show_all_lt_thread    _X__pid: 865 tp: 0x300022897e0 ENQ: 64 VIRTRUN: 0
show_all_lt_thread    _X__pid: 556 tp: 0x30002201520 ENQ: 0 VIRTRUN: 0
hold_lottery_user   tp_pid: 916 uid: 39809 winner_user: 30 glob_tkts: 200
hold_lottery_last   tp_pid: 916 lttp_pid: 917
lt_setfrontdq       tp_pid: 916 lttp_pid: 917 ondq: 1
```

Test Programs

loop-while.c - “Computation Bound”:

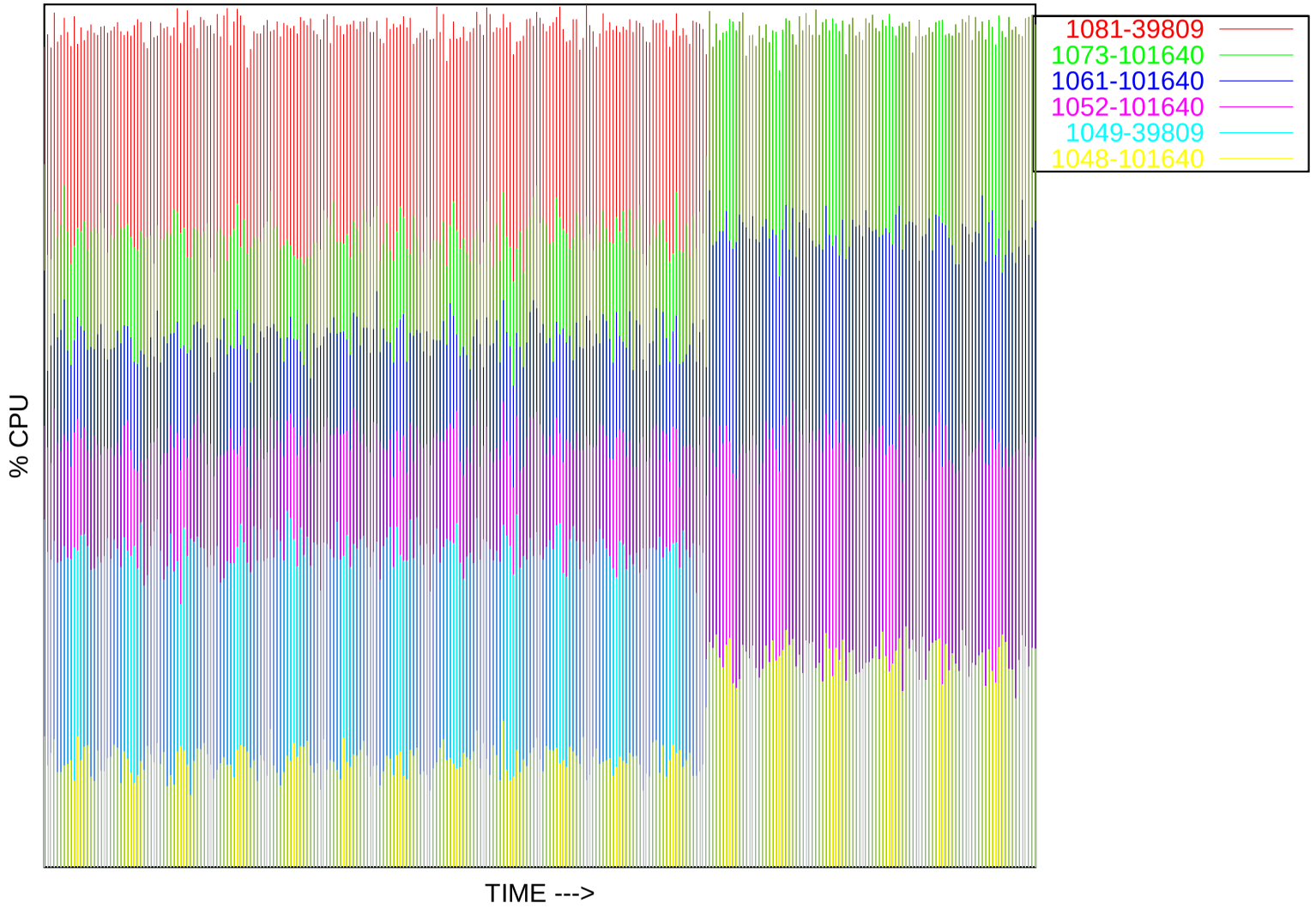
```
void main() {
  while(1==1) {};
}
```

loop-syscall.c - “I/O Bound”:

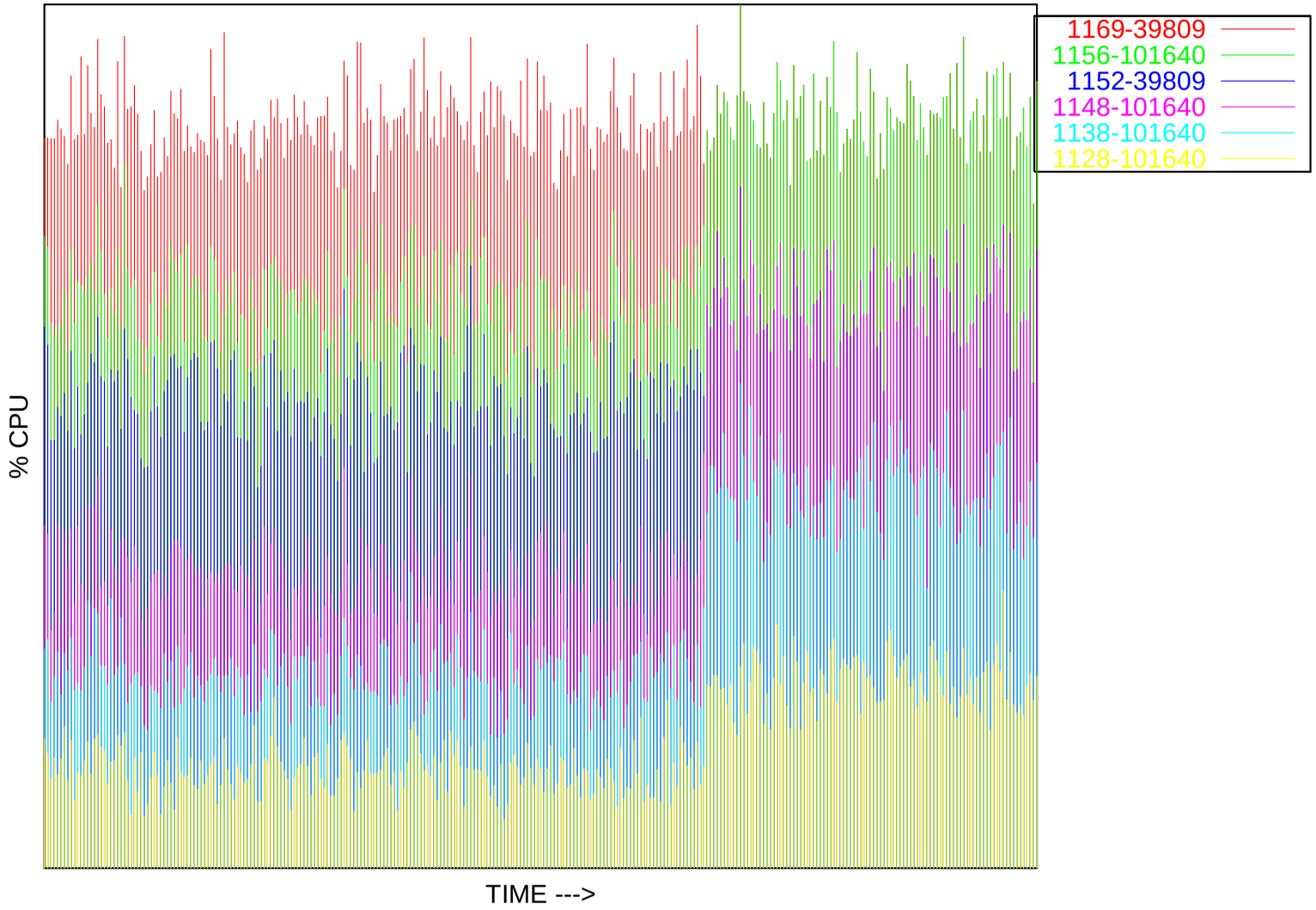
```
void main() {
  while(1==1) {
      char buf;
       int fd = open("/etc/motd", O_RDONLY);
      read(fd, &buf, 1);
      close(fd);
  };
}
```

1 CPU, 2 users, 3 Threads, 1:2, while

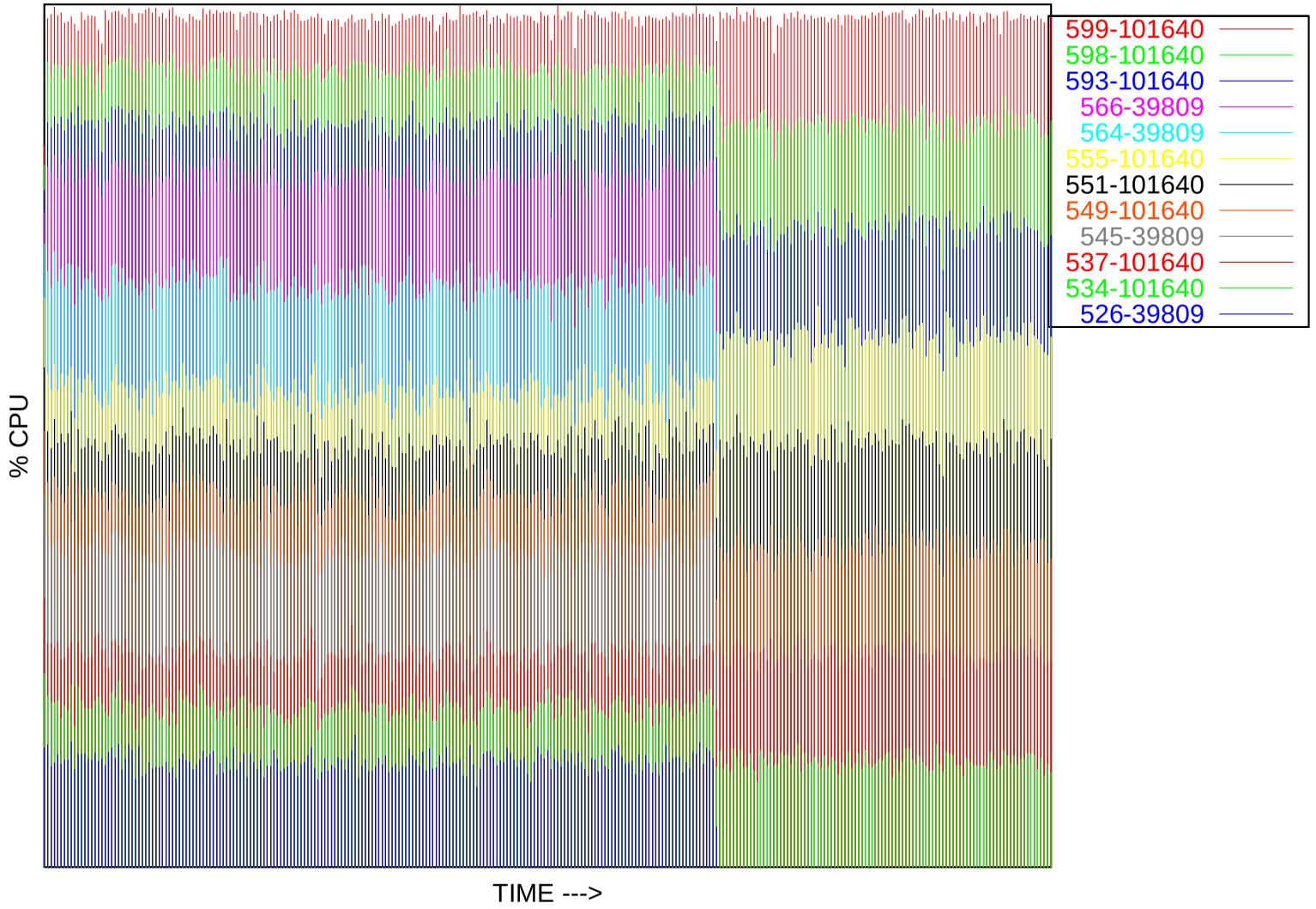
2 CPU, 2 users, 6 Threads, 2:4, while



2 CPU, 2 users, 6 Threads, 2:4, syscall



4 CPU, 2 users, 12 Threads, 4:8, while



8 CPU, 2 users, 24 Threads, 8:16, while