# Elfen Scheduling

## Fine-Grain Principled Borrowing from Latency-Critical Workloads using Simultaneous Multithreading (SMT)


Xi Yang

Australian National University

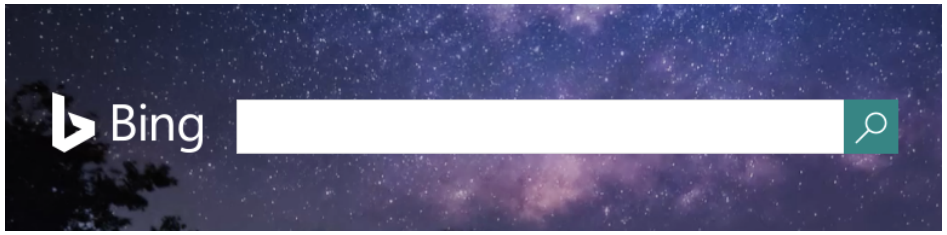
Stephen M Blackburn

Australian National University

Kathryn S McKinley

Microsoft Research

# Tail Latency Matters




Two second slowdown reduced revenue/user by 4.3%.

[Eric Schurman, Bing]

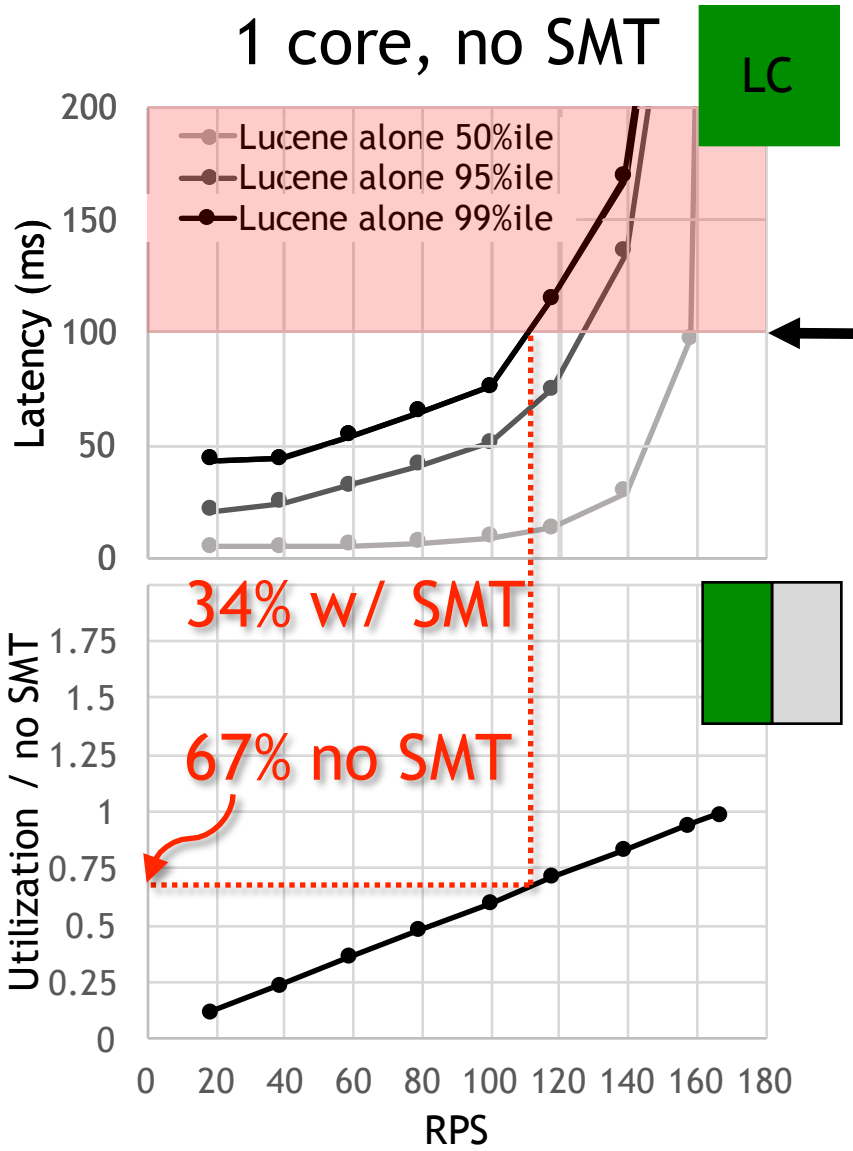

400 millisecond delay decreased searches/user by 0.59%.

[Jack Brutlag, Google]

"Such WSCs tend to have relatively low average utilization, spending most of its time in the 10 - 50% CPU utilization range."

Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

# High Responsiveness—Low Utilization




LC

Lucene

*Intel Xeon-D 1540 Broadwell*

"Such WSCs tend to have relatively low average utilization, spending most of its time in the 10 - 50% CPU utilization range."

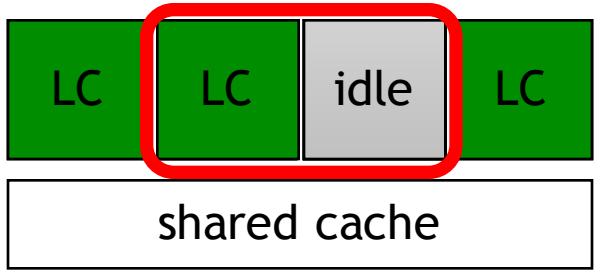
Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

# Soak up Slack with Batch?

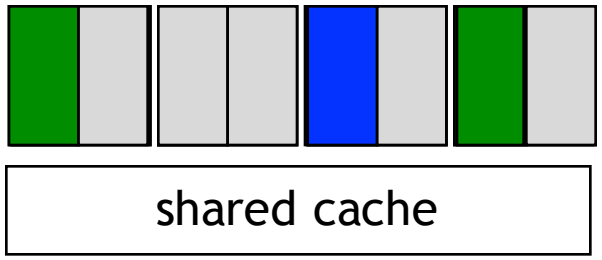



responsiveness requires idle cores

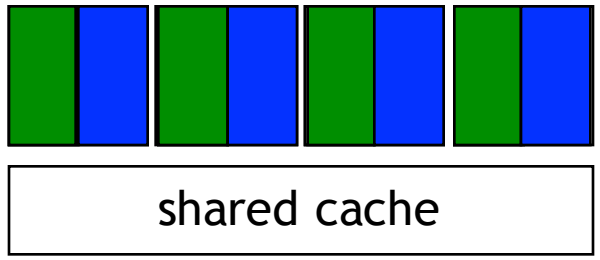
because OS descheduling is slow




Co-running on different cores
SMT turned off



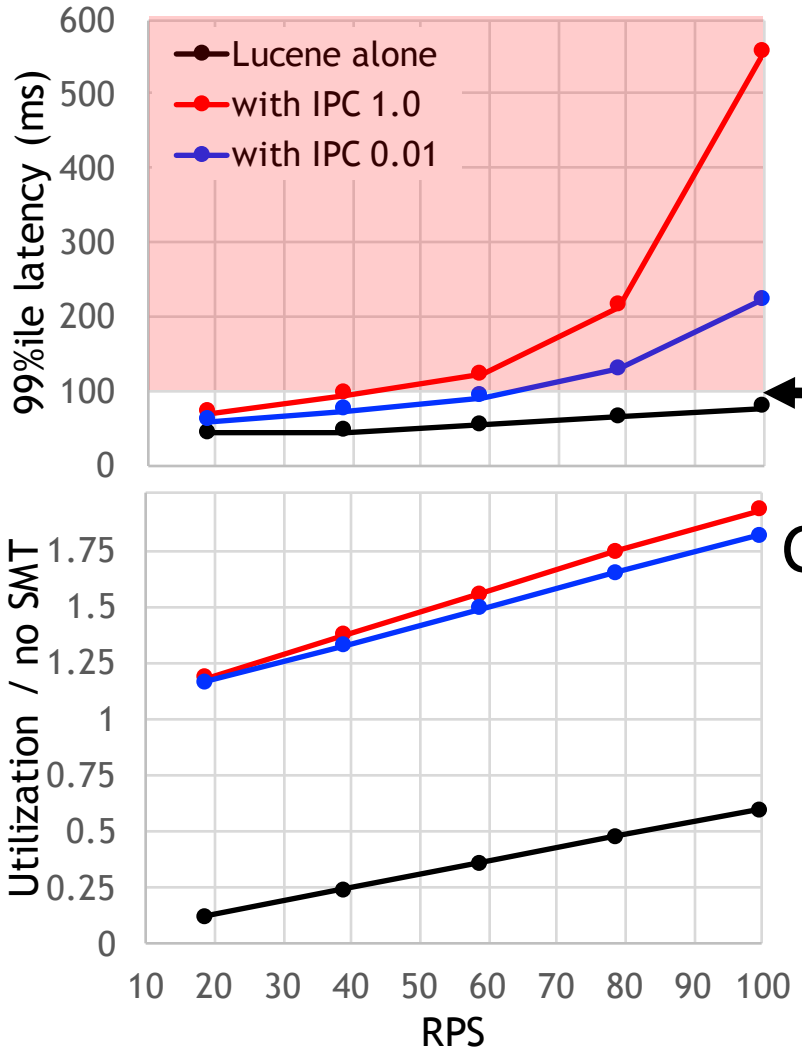

Co-running on different cores
SMT turned off




Co-running on same core
in SMT lanes

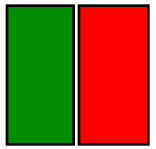
# SMT Co-Runner

1 core, 2 SMT lanes



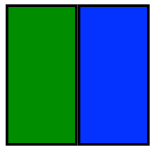

SLO

Great utilization!



```
while(1);
```

IPC 1.0

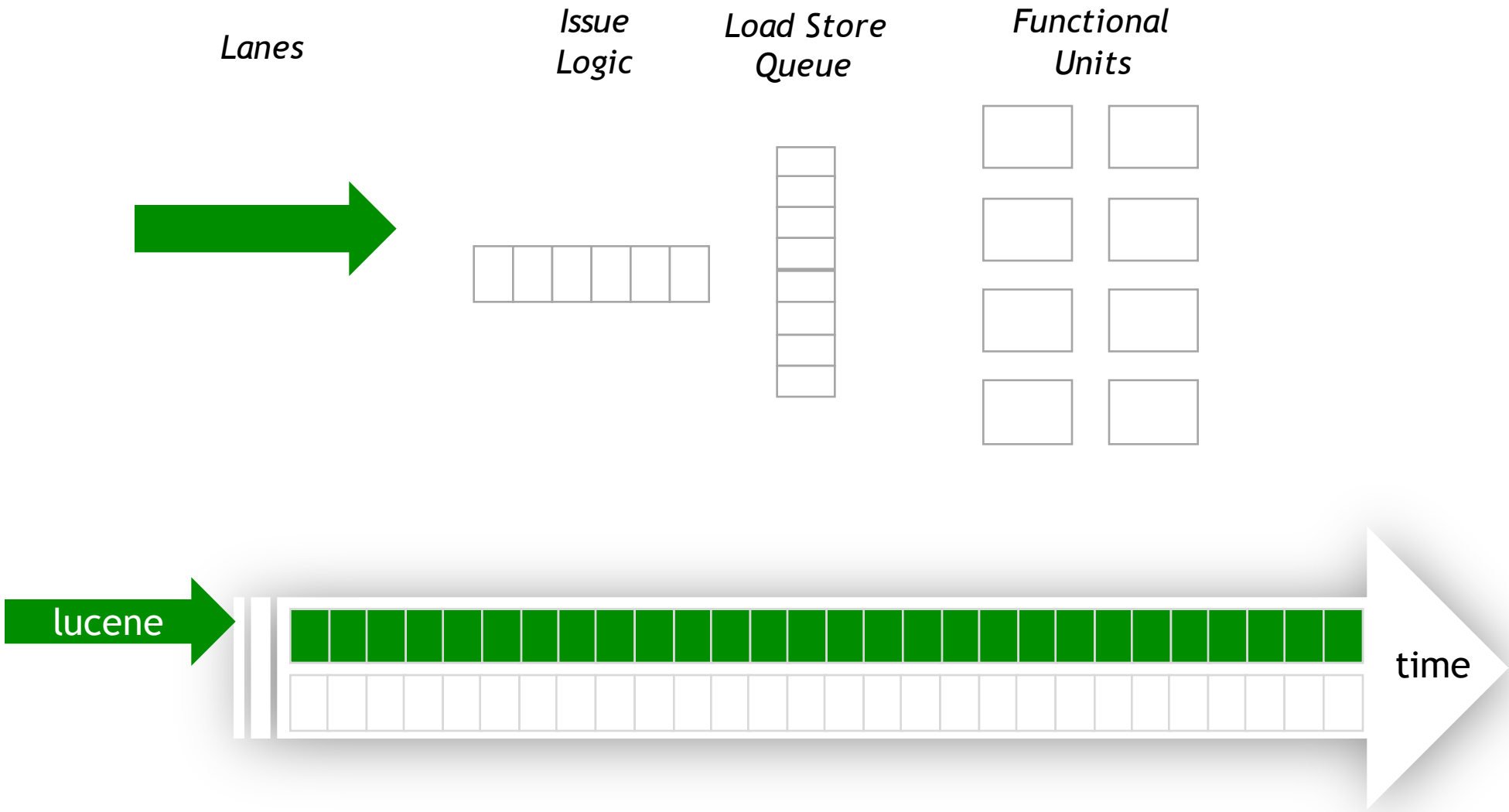


```
while(1) {
  movnti();
  mfence();
}
```

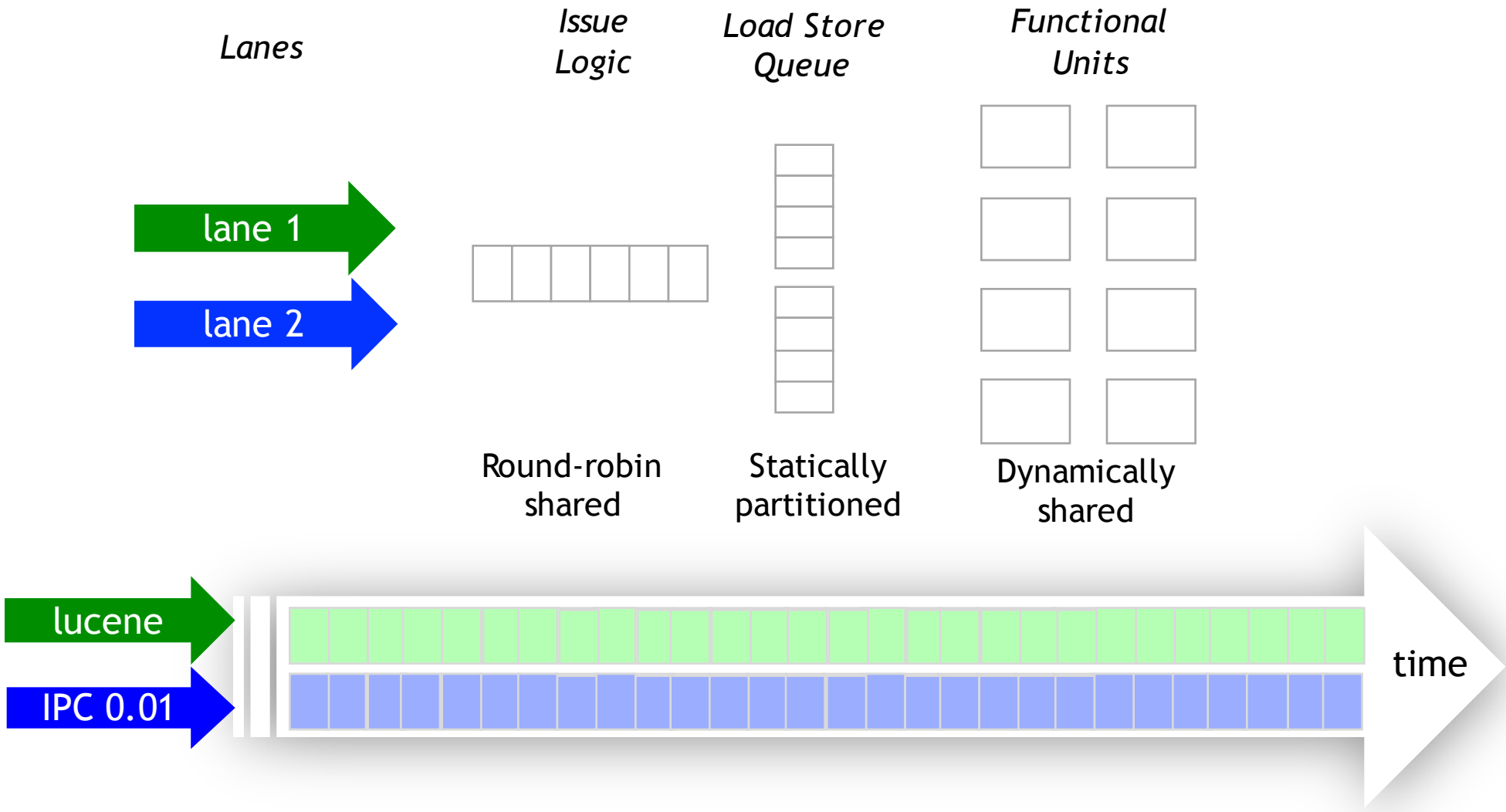
IPC 0.01

Even IPC 0.01 violates SLO at low load!

# Simultaneous Multithreading OFF

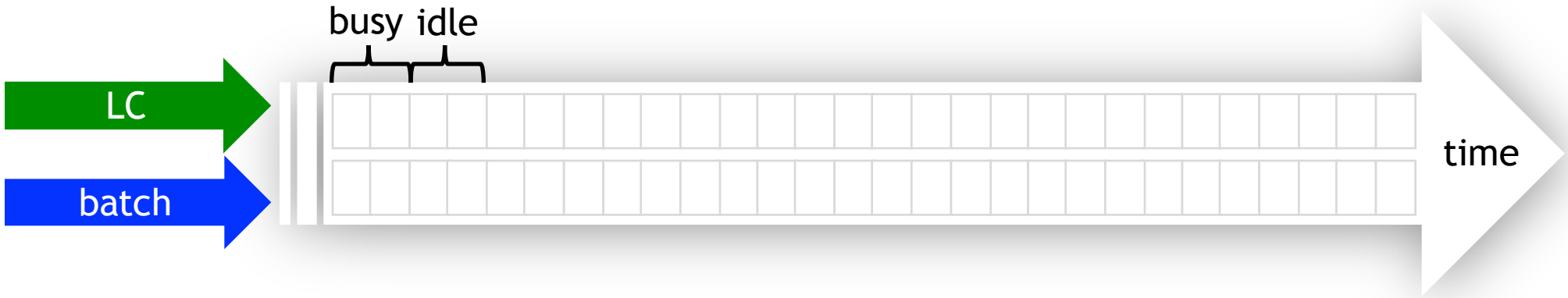


# Simultaneous Multithreading ON




Active SMT lanes share critical resources

# Principled Borrowing

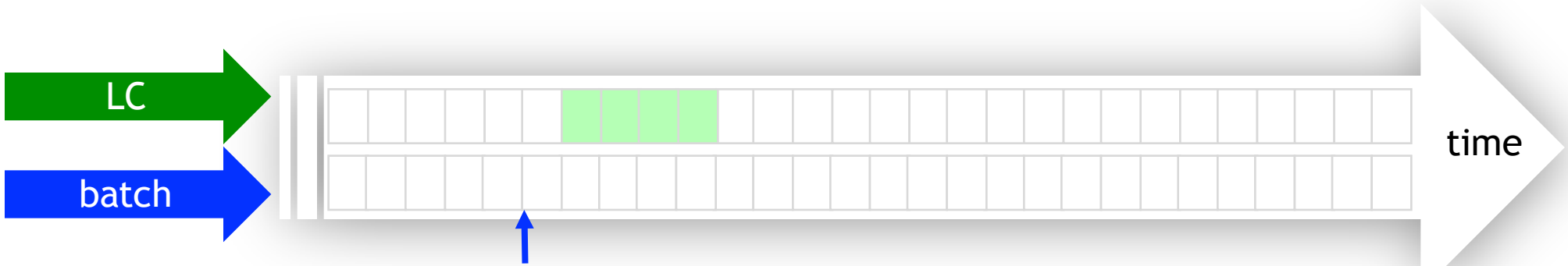



Batch borrows hardware when LC is idle

Batch releases hardware when LC is busy

Can we implement principled borrowing on current hardware?

# Hardware is Ready — Software is Not




Thread sleeps, releasing hardware to OS (~2K cycles)

OS schedules batch lane with any ready job

**OS supports thread sleeping, but not hardware sleeping**

**release SMT hardware to other lane**

# nanonap()

## Semantics

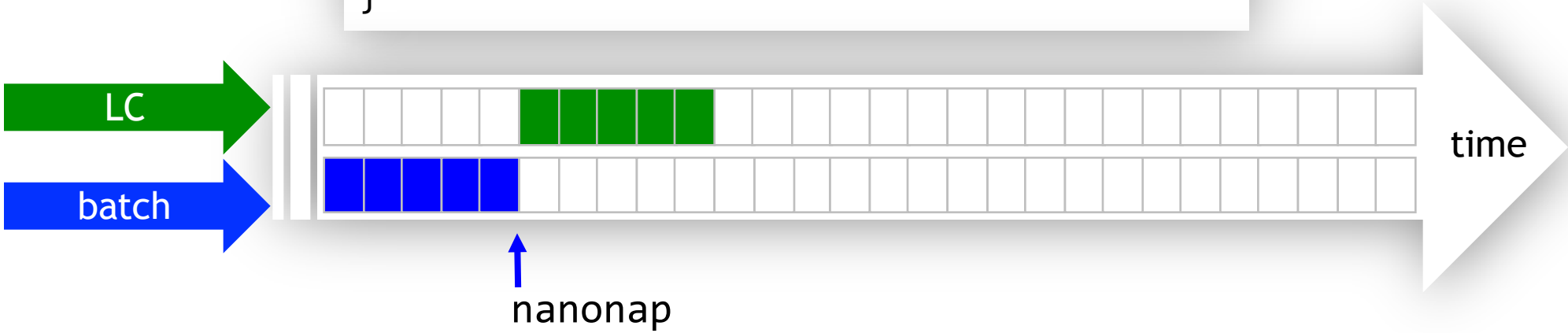
Thread invoking nanonap releases SMT hardware without releasing SMT context

OS cannot schedule hardware context

OS can interrupt and wakeup thread

# nanonap()

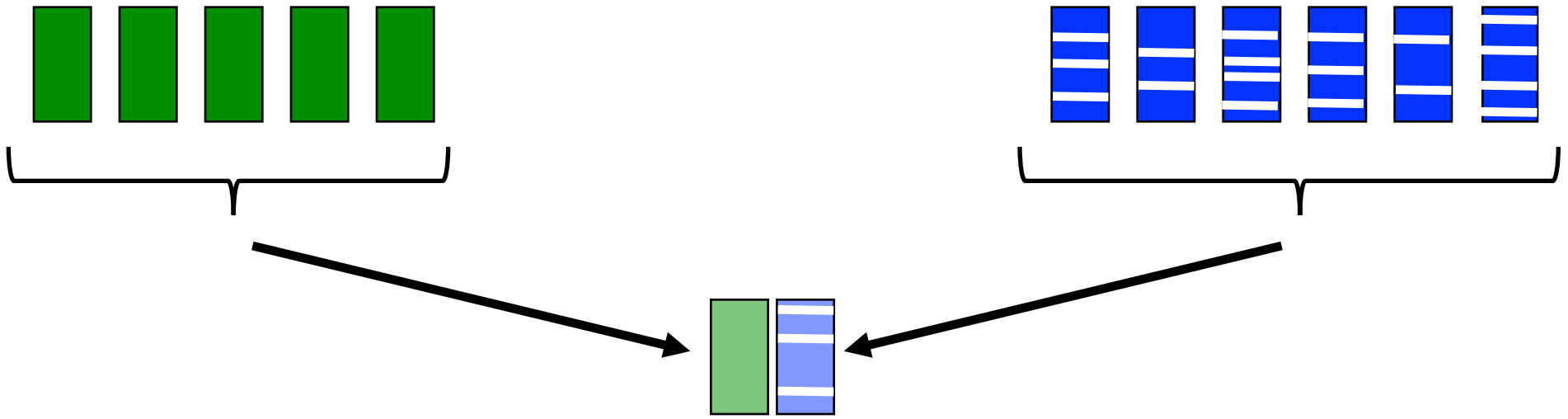
```
per_cpu_variable: nap_flag;
void nanonap() {
    enter_kernel();
    disable_preemption();
    my_nap_flag = this_cpu_flag(nap_flag);
    monitor(my_nap_flag);
    mwait();
    enable_preemption();
    leave_kernel();
}
```



# Elfen Scheduler

No change to latency-critical threads

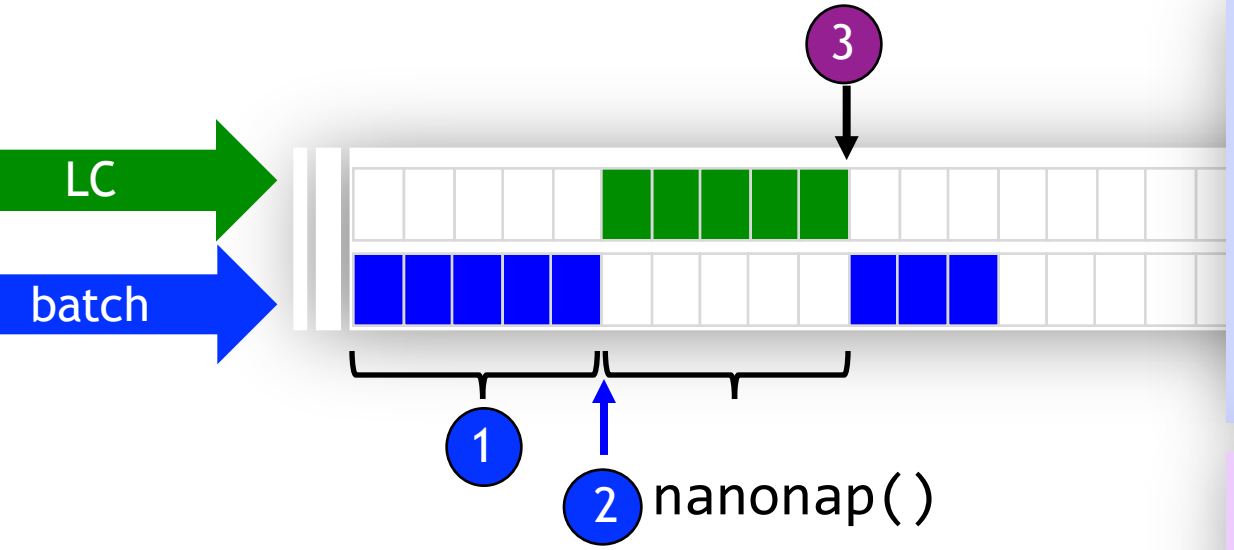
Instrument batch workloads to detect LC threads & nap



Bind latency-critical threads to LC lane

Bind batch threads to batch lane

# Elfen Scheduler



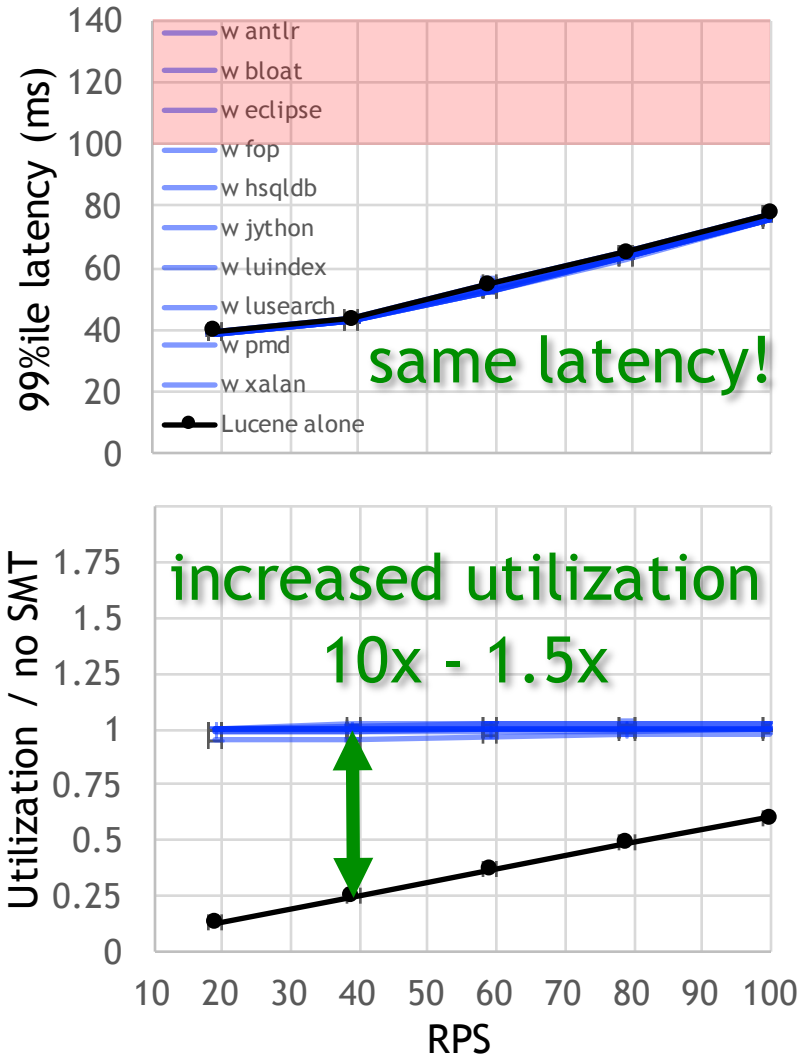

```
/* fast path check injected into method body */
    check:
1   if (!request_lane_idle)
        slow_path();

2   slow_path() {
        nanonap(); }
```

```
/*     maps lane IDs to the running task      */
    exposed SHIM signal: cpu_task_map
    task_switch(task T) {
       cpu_task_map[thiscpu] = T;
    }
    idle_task() {
      // wake up any waiting batch thread
3     update_nap_flag_of_partner_lane();
      ......
}
```

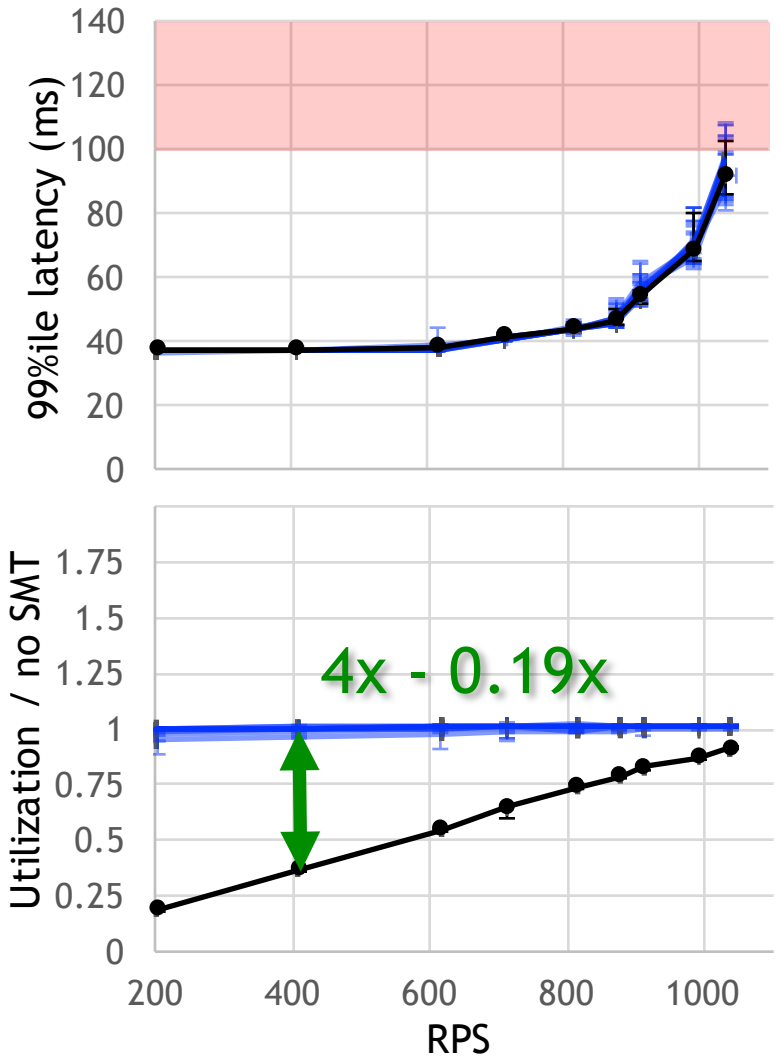
1. Batch thread borrows resources, continuously checks LC lane status
2. LC starts, batch calls nanonap() to release SMT hardware resources
3. OS touches nap_flag to wake up batch thread

# Results: Borrow Idle

## 1 core, 2 SMT lanes

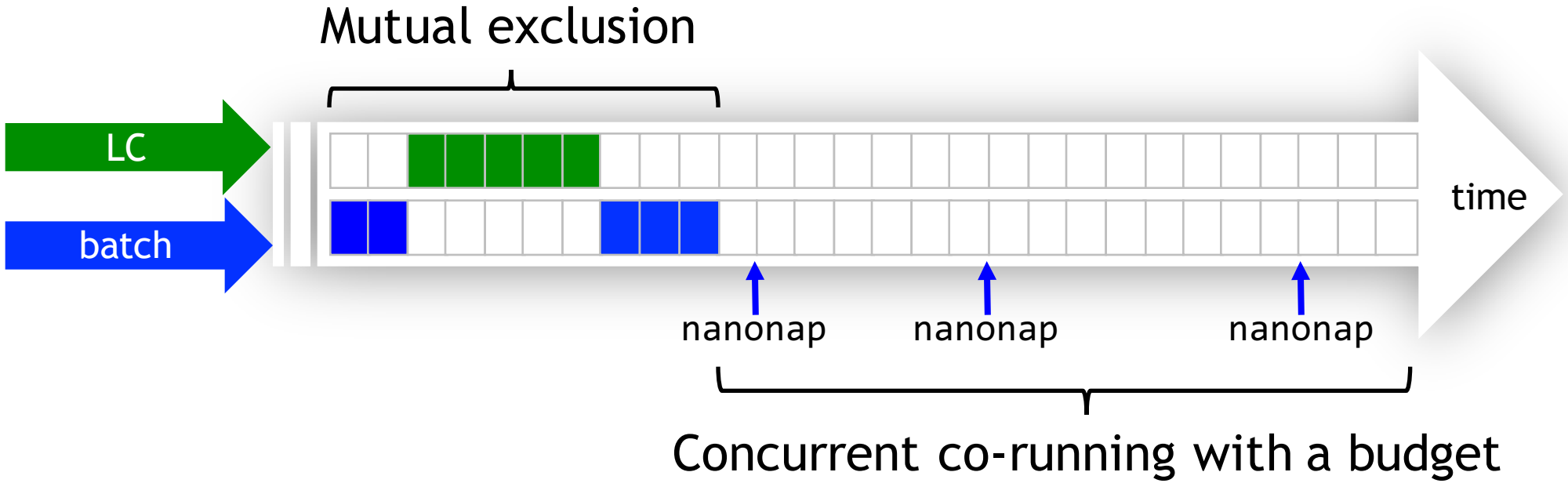



## 7 cores, 2x7 SMT lanes



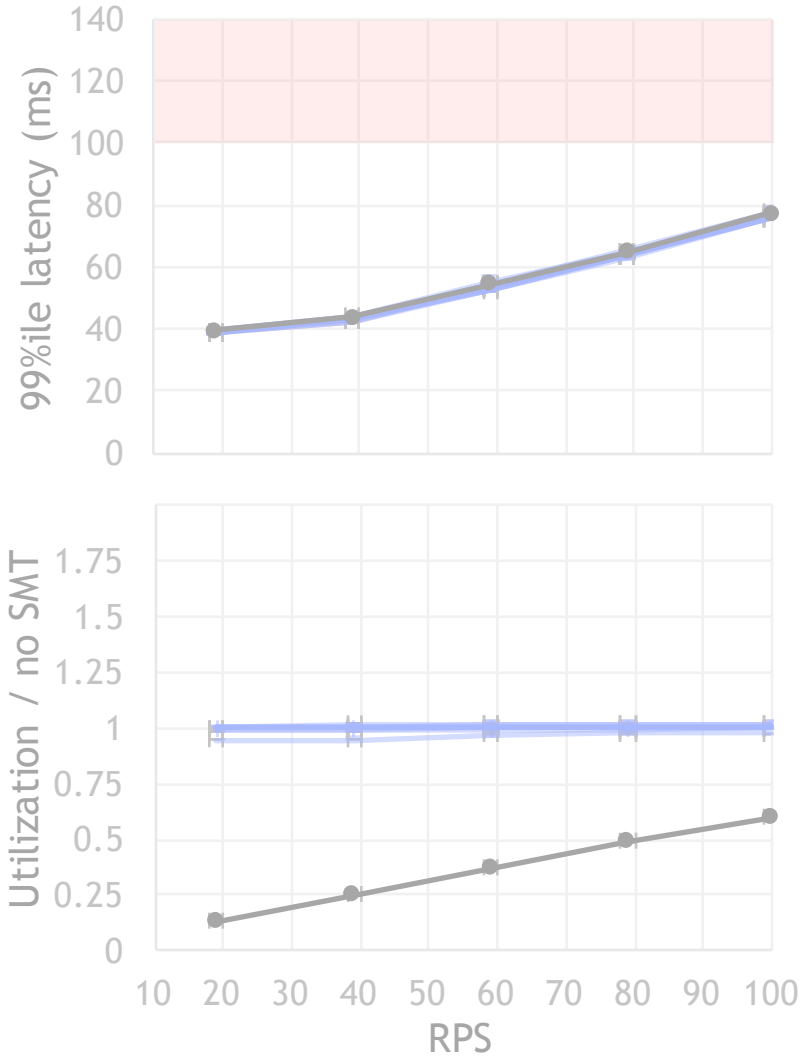
# Beyond Mutual Exclusion

- Borrow idle policy

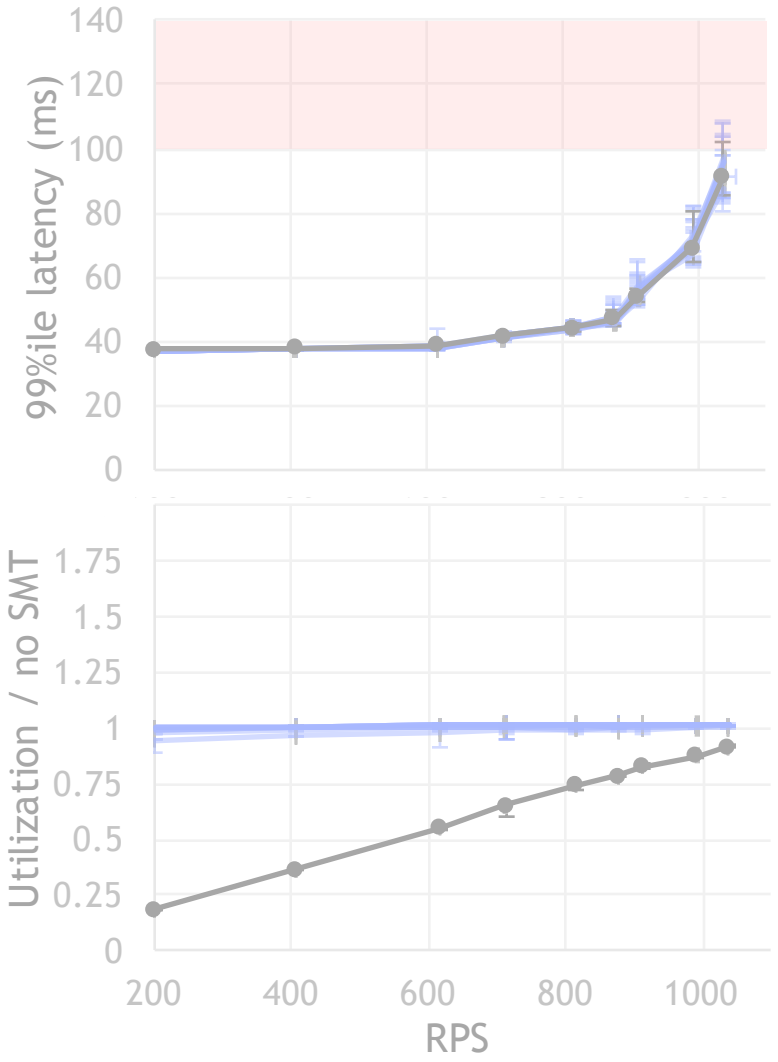



- Fixed budget policy
- Refresh budget policy
- Dynamic budget policy

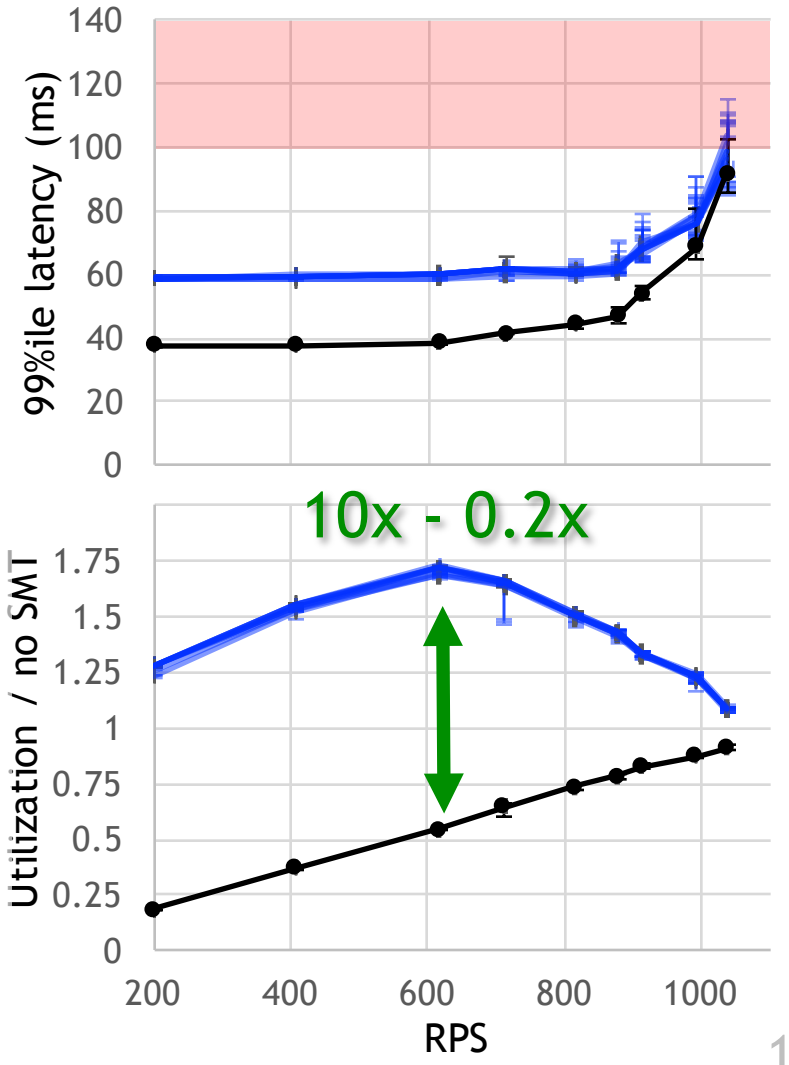
Borrow idle on 1 core




Borrow idle on 7 cores




Dynamic budget 7 cores



# Conclusion

Principled borrowing for SMT

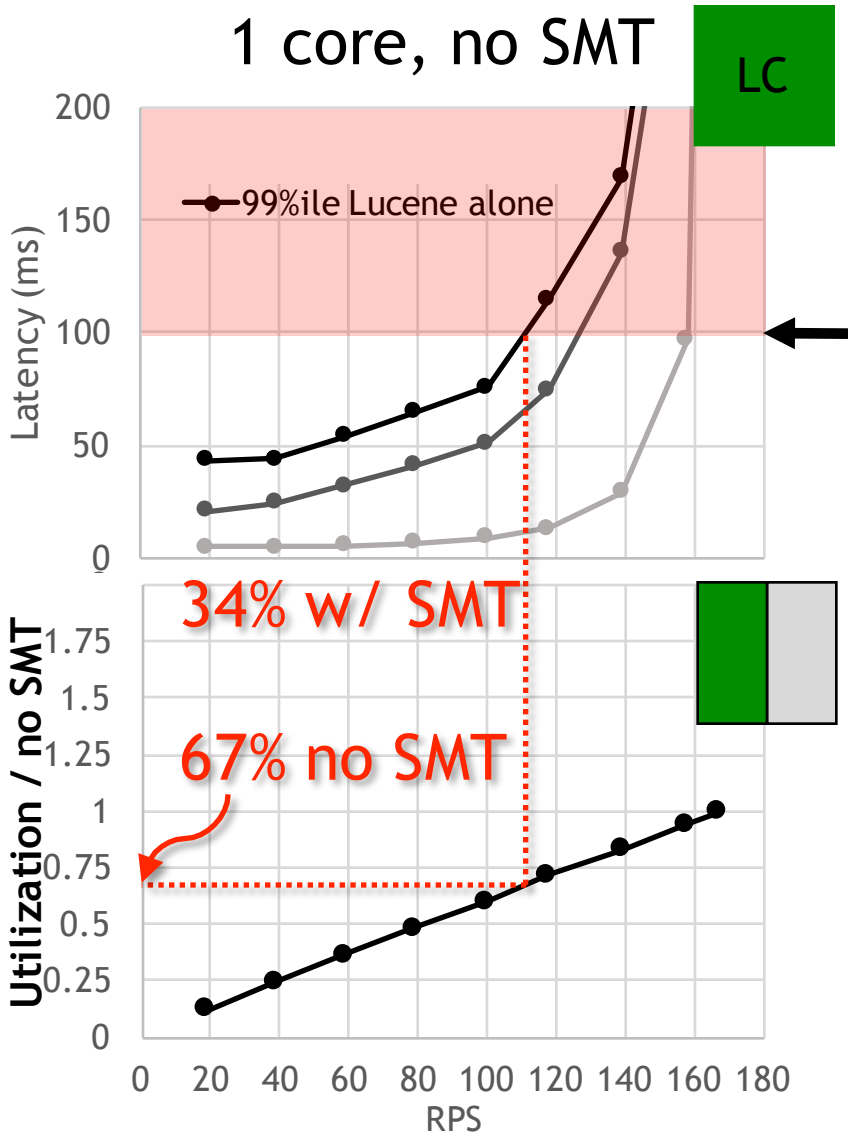
- `nanonap()` releases STM hardware *without* giving it to OS
- Elfen scheduler implementation & policies
- 90% to 25% increase in utilization
- Same tail latency!

`https://github.com/elfenscheduler`

Thank you!

# BACKUP SLIDES

# High Responsiveness—Low Utilization



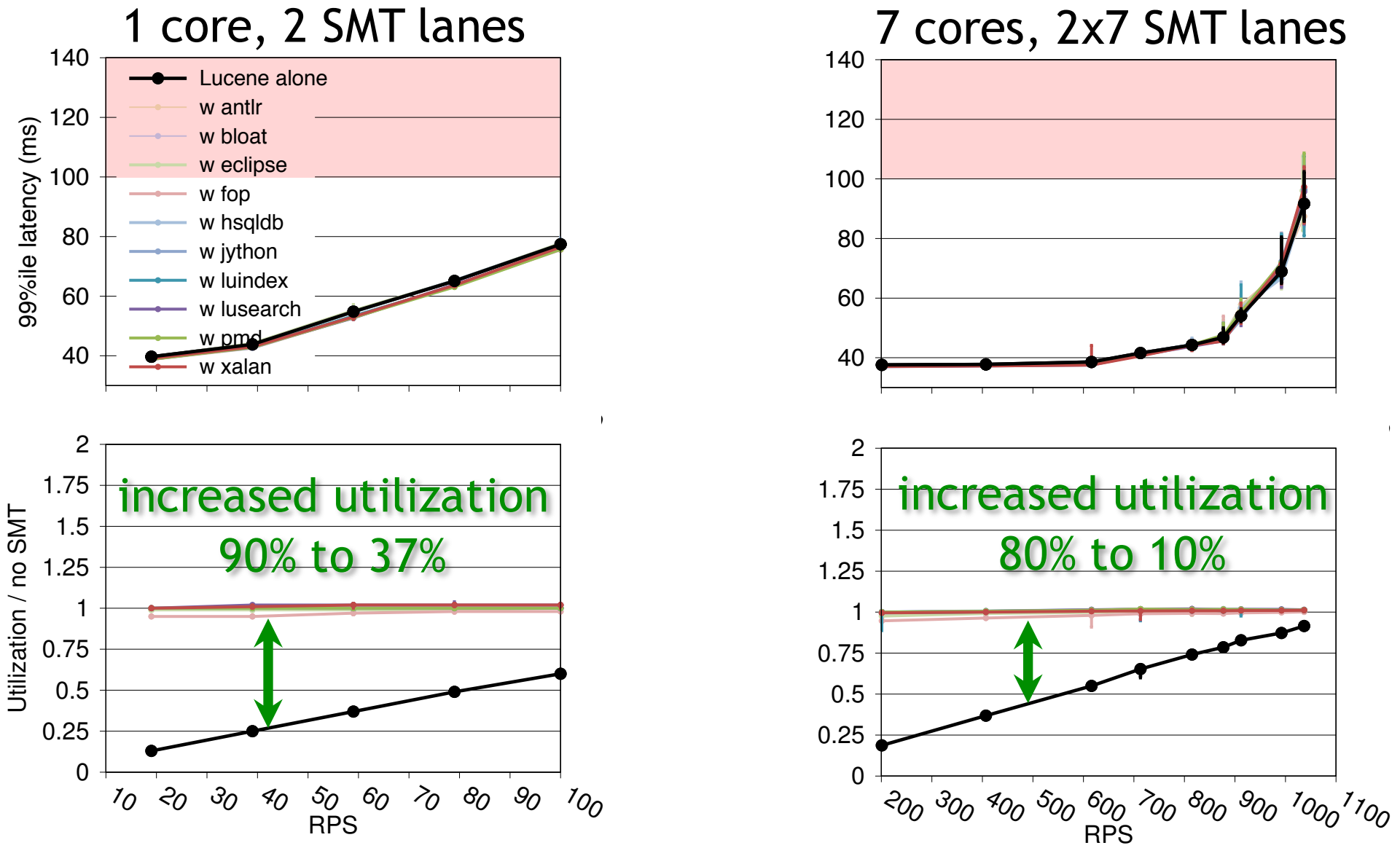

LC Lucene

*Intel Xeon-D 1540 Broadwell*

"Such WSCs tend to have relatively low average utilization, spending most of its time in the 10 - 50% CPU utilization range."

Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

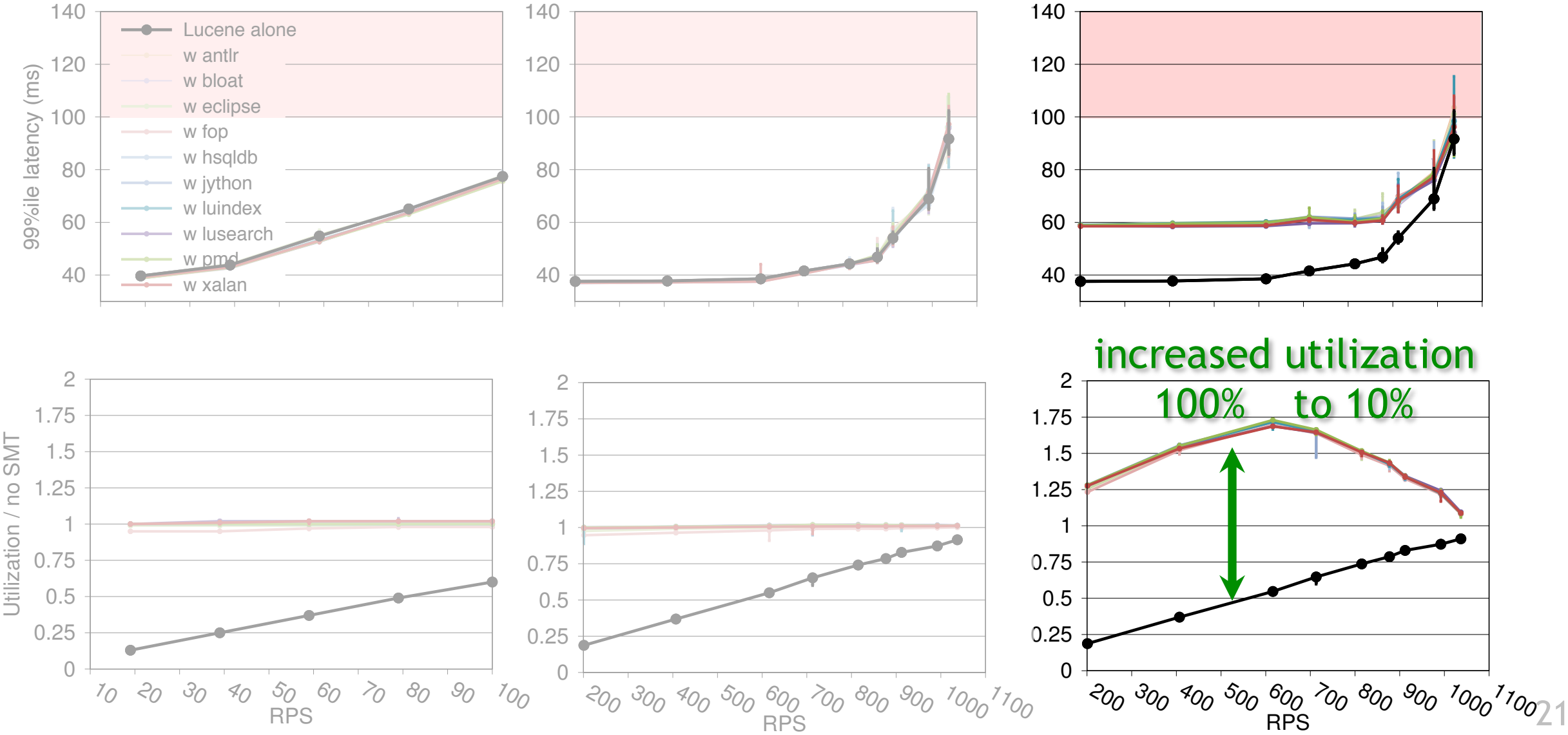
# Results: Borrow Idle



## Borrow idle on 1 cores

## Borrow idle on 7 cores

## Dynamic budget 7 cores



# Result

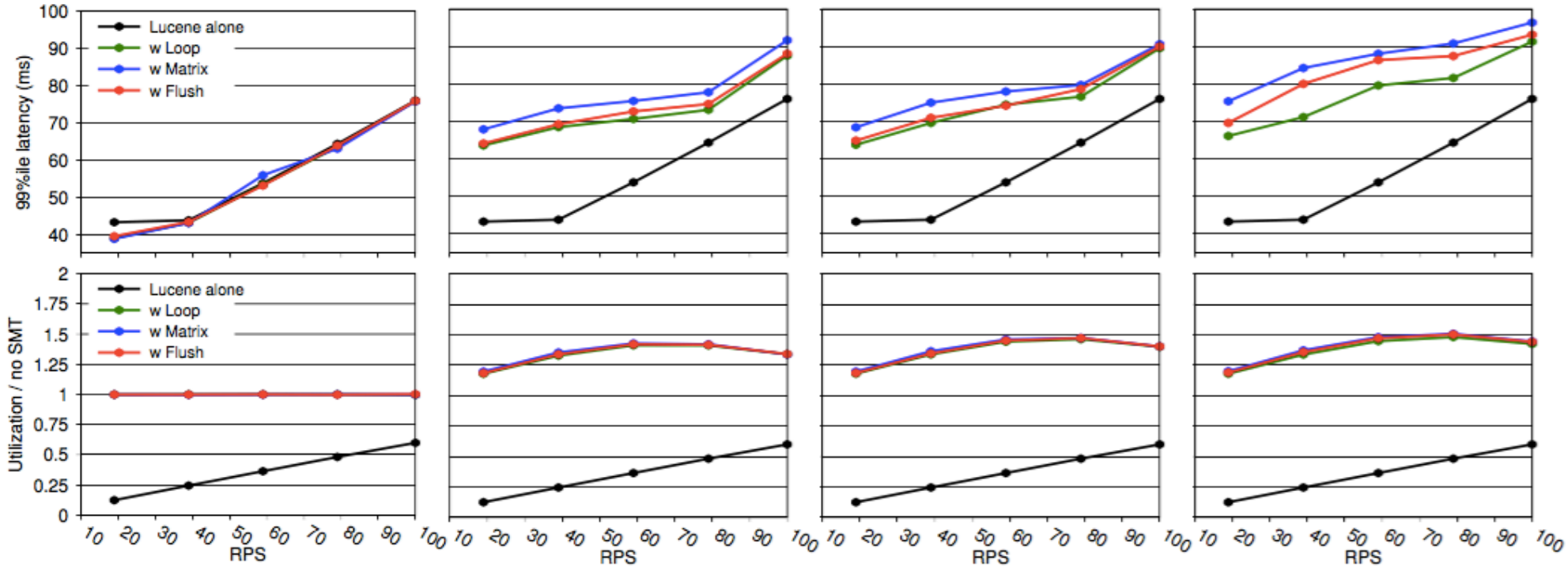



**Figure 11.** 99th percentile latency (top) and utilization (bottom) with C microbenchmarks for policies with *budget* targets on one 2-way SMT core.

# Result

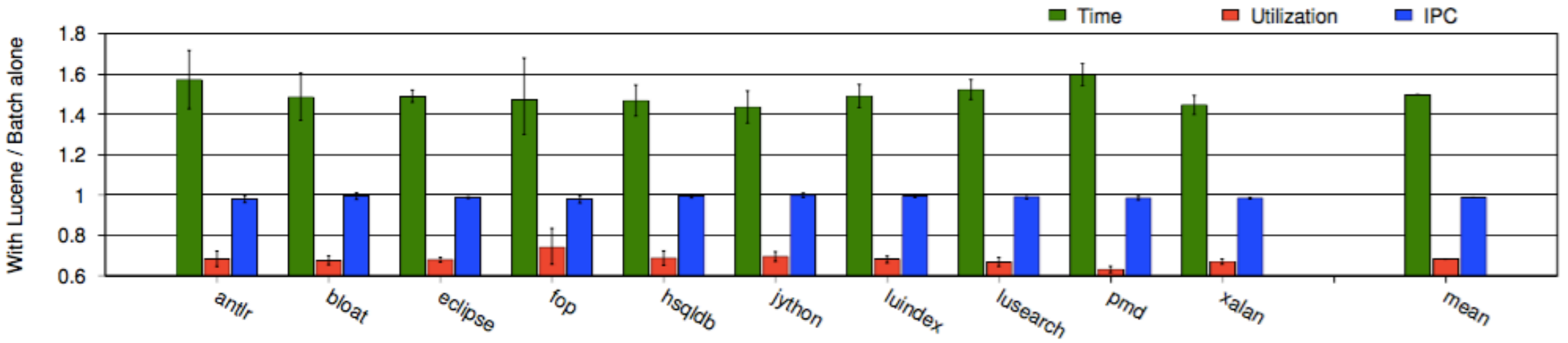
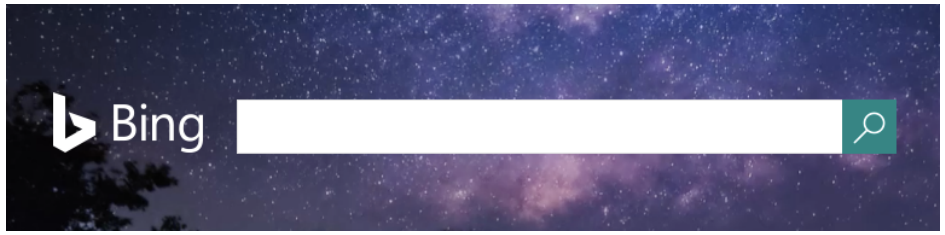



**Figure 12.** Normalized DaCapo benchmark execution time, user space CPU utilization, and IPC.

# BACKUP SLIDES

# Tail Latency Matters



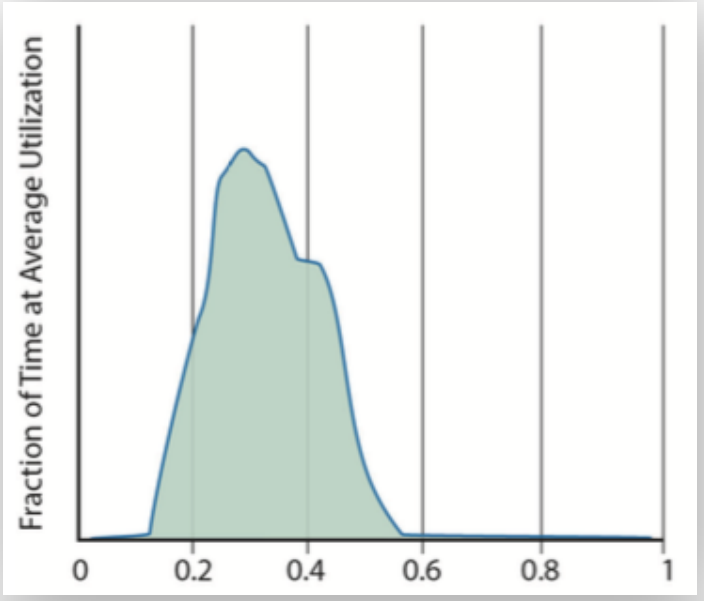

Two second slowdown reduced revenue/user by 4.3%.

[Eric Schurman, Bing]

Google Australia

400 millisecond delay decreased searches/user by 0.59%.

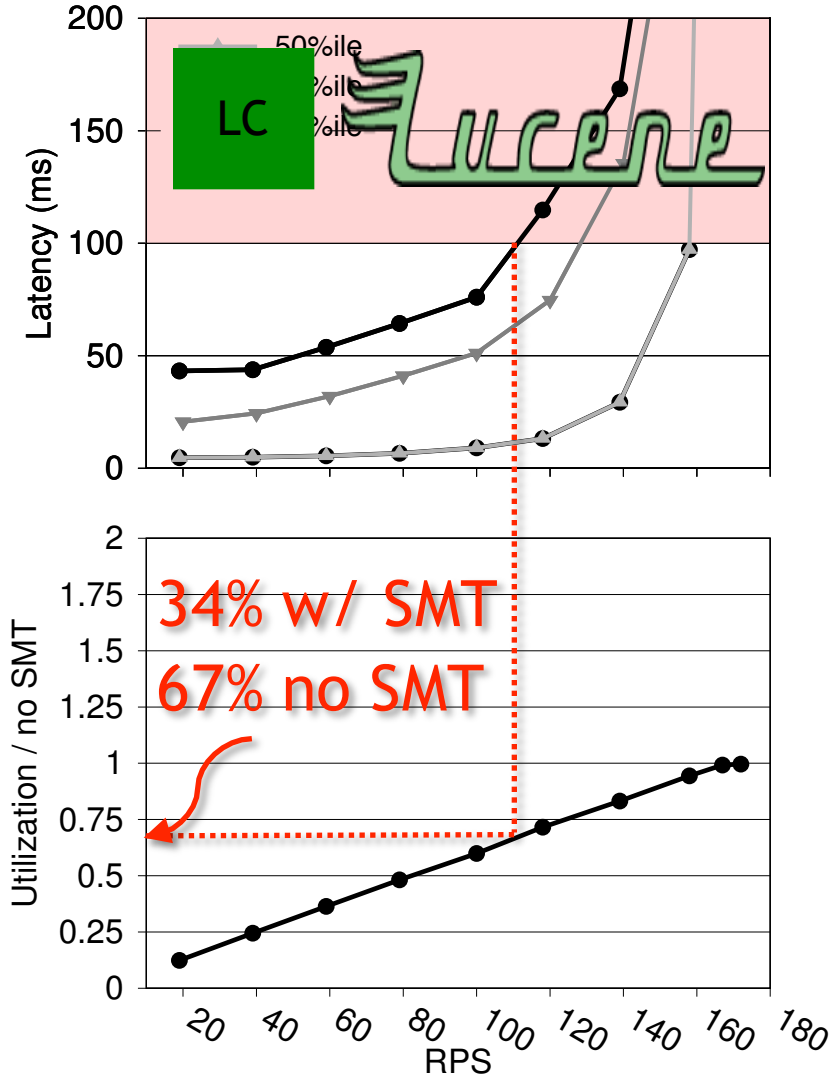
[Jack Brutlag, Google]

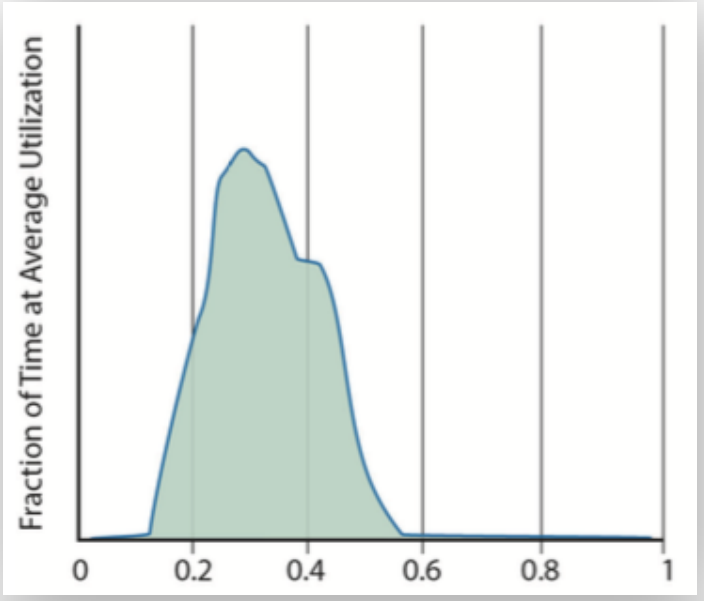



"Such WSCs tend to have relatively low average utilization, spending most of its time in the 10 - 50% CPU utilization range."

p75, Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

# High Responsiveness—Low Utilization



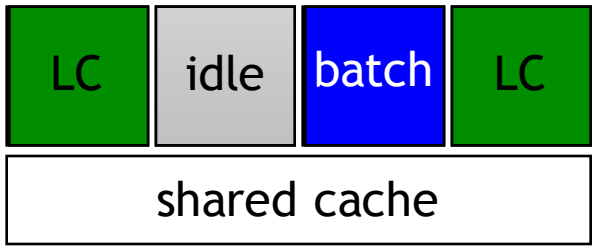

"Such WSCs tend to have relatively low average utilization, spending most of its time in the 10 - 50% CPU utilization range."

p75, Luiz André Barroso, Urs Hölzle
"The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines"

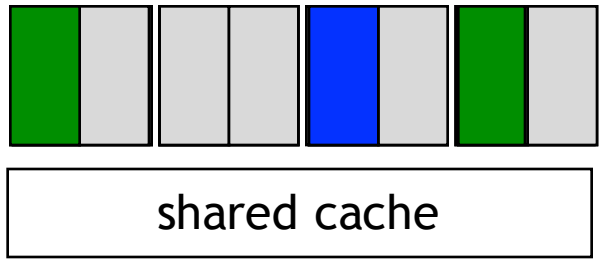
*Lucene search benchmark on one core of an Intel Xeon-D 1540 Broadwell.*

# Soak up Slack with Batch?

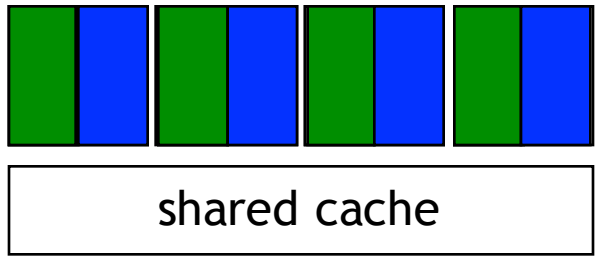





Co-running on different cores

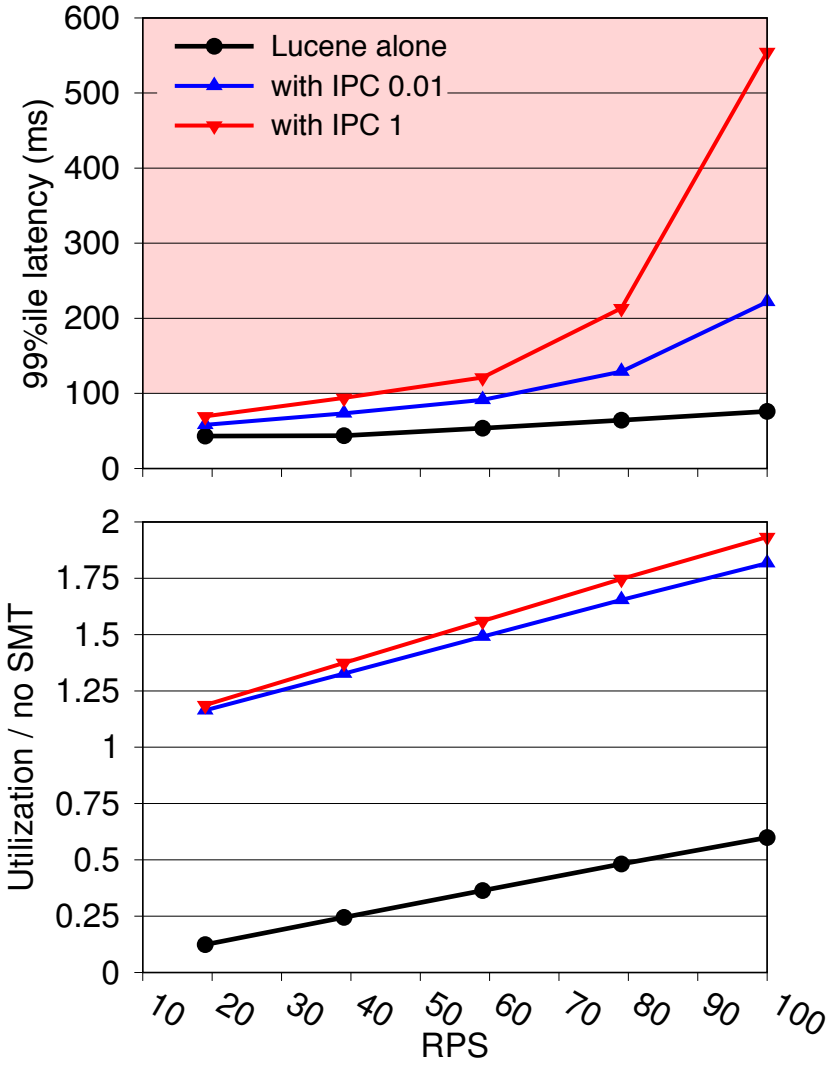


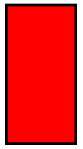

SMT turned off




Co-running on same core in SMT lanes

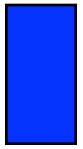
# SMT Co-Runner



```
while(1);
```

IPC 1.0

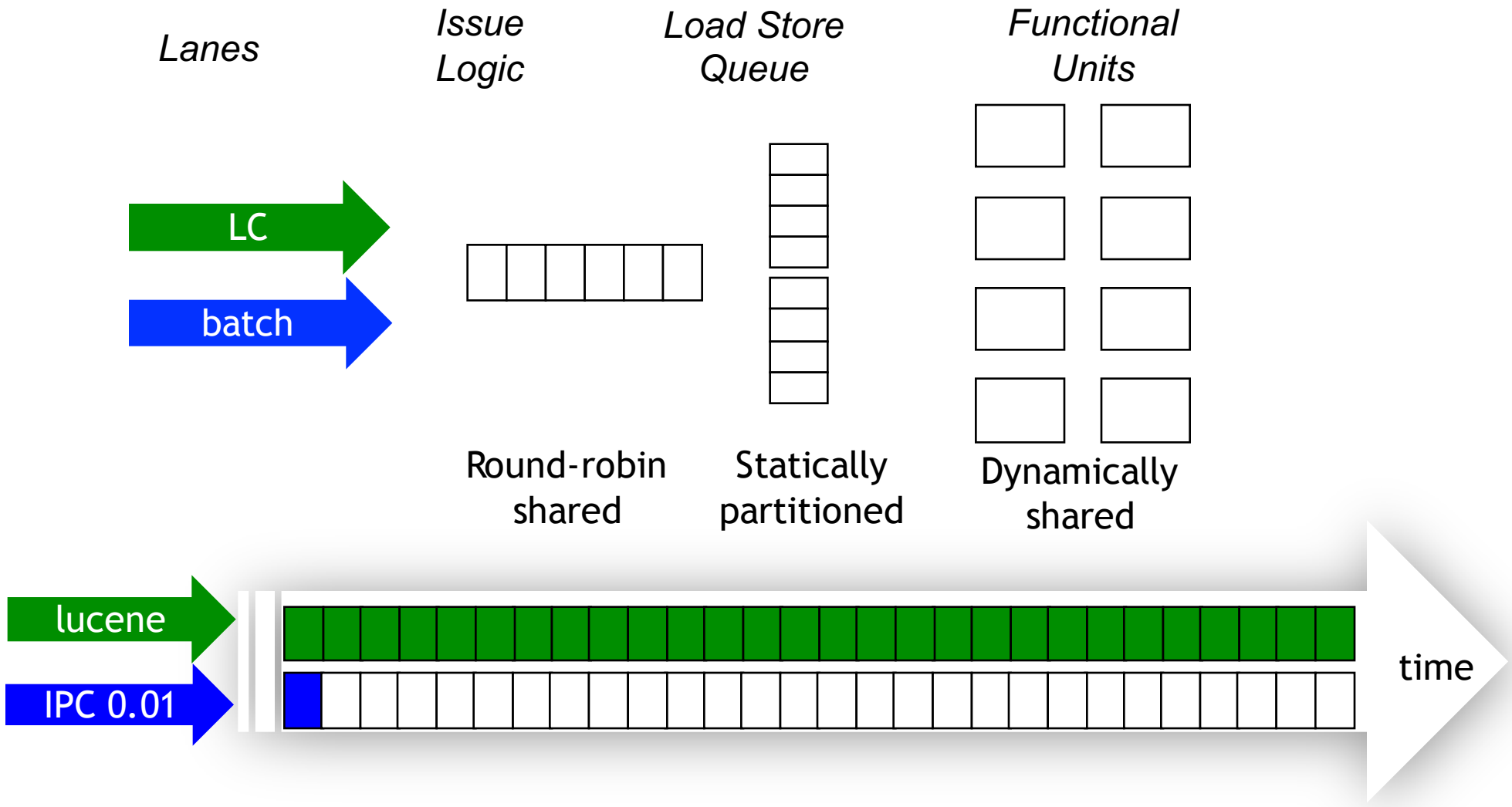


```
while(1) {
  movnti();
  mfence();
}
```

IPC 0.01

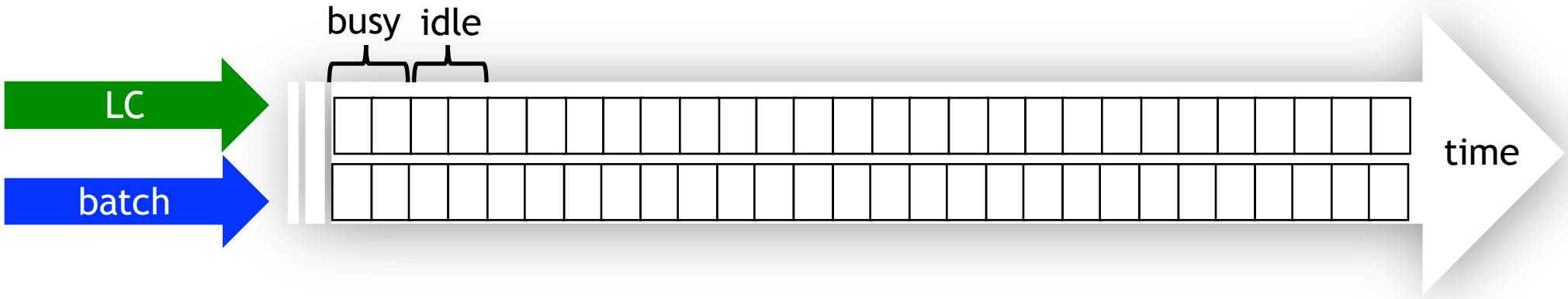
Even IPC 0.01 violates SLO at low load!

# Even Low IPC Co-Runner is Destructive




Critical resources are shared by active SMT lanes.

# Principled Borrowing

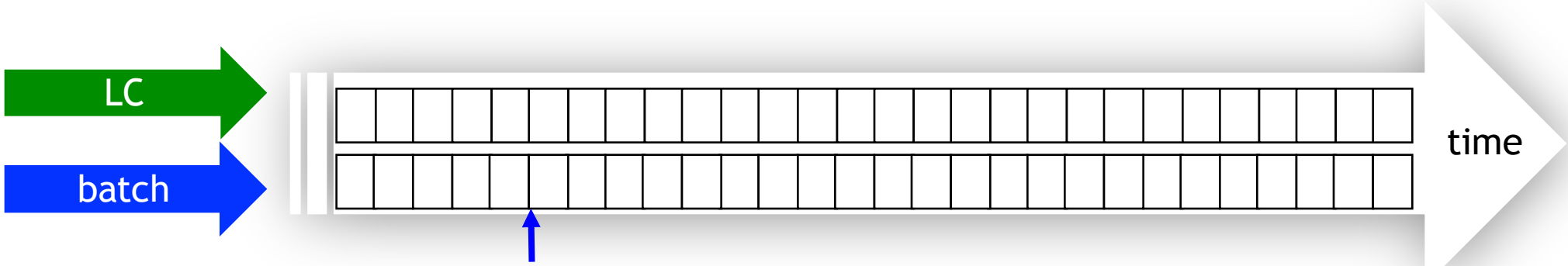



Batch lane borrows hardware resources when LC lane is idle

Batch lane releases hardware resources when LC lane is busy

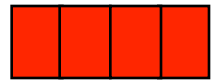
Can we implement principled borrowing on current hardware?

# Hardware is Ready — Software is Not




Batch lane calls “mwait”

Thread sleeps, releasing hardware to OS (~2K cycles)

OS schedules batch lane with any ready job 

When LC lane is busy, batch lane must do nothing (*nap)*

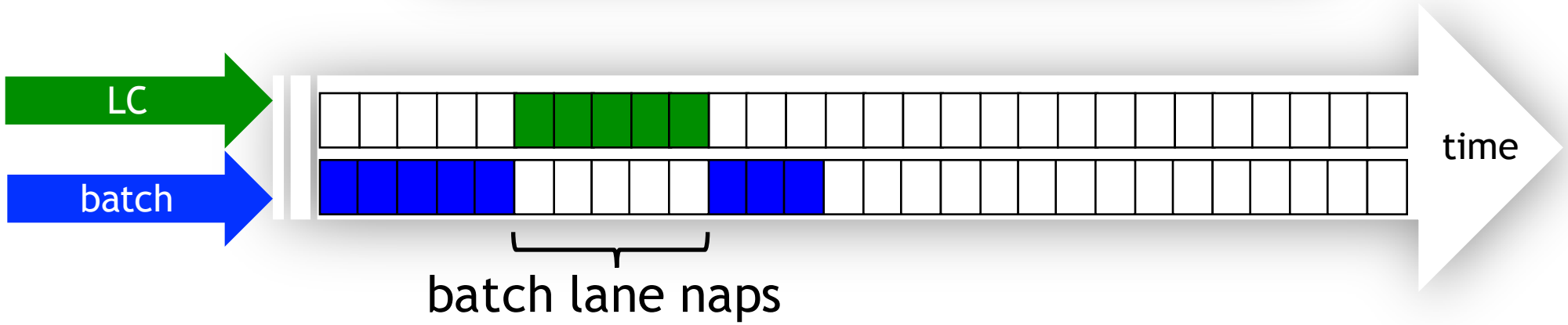
# nanonap()

Thread nap semantics

- Put SMT hardware lane in idle state
- OS cannot schedule hardware context
- OS can interrupt and wakeup thread

# nanonap()

```
per_cpu_variable: nap_flag;
void nanonap() {
    enter_kernel();
    disable_preemption();
    my_nap_flag = this_cpu_flag(nap_flag);
    monitor(my_nap_flag);
    mwait();
    enable_preemption();
    leave_kernel();
}
```



# Elfen Scheduler

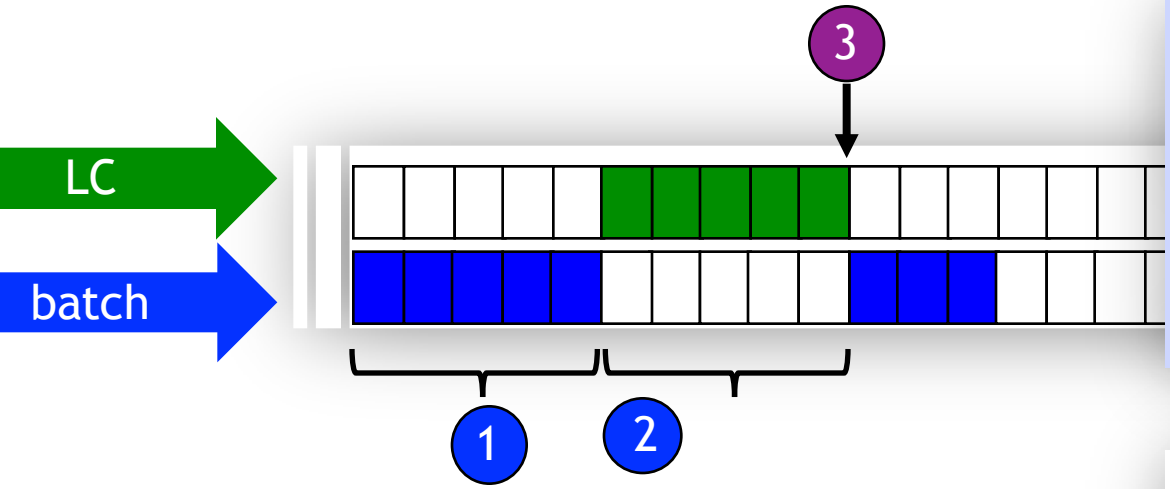
No change to latency critical threads

Bind latency critical threads to LC lane

Bind batch threads to batch lane

Instrument batch workloads to detect LC threads quickly & nap

# Elfen Scheduler

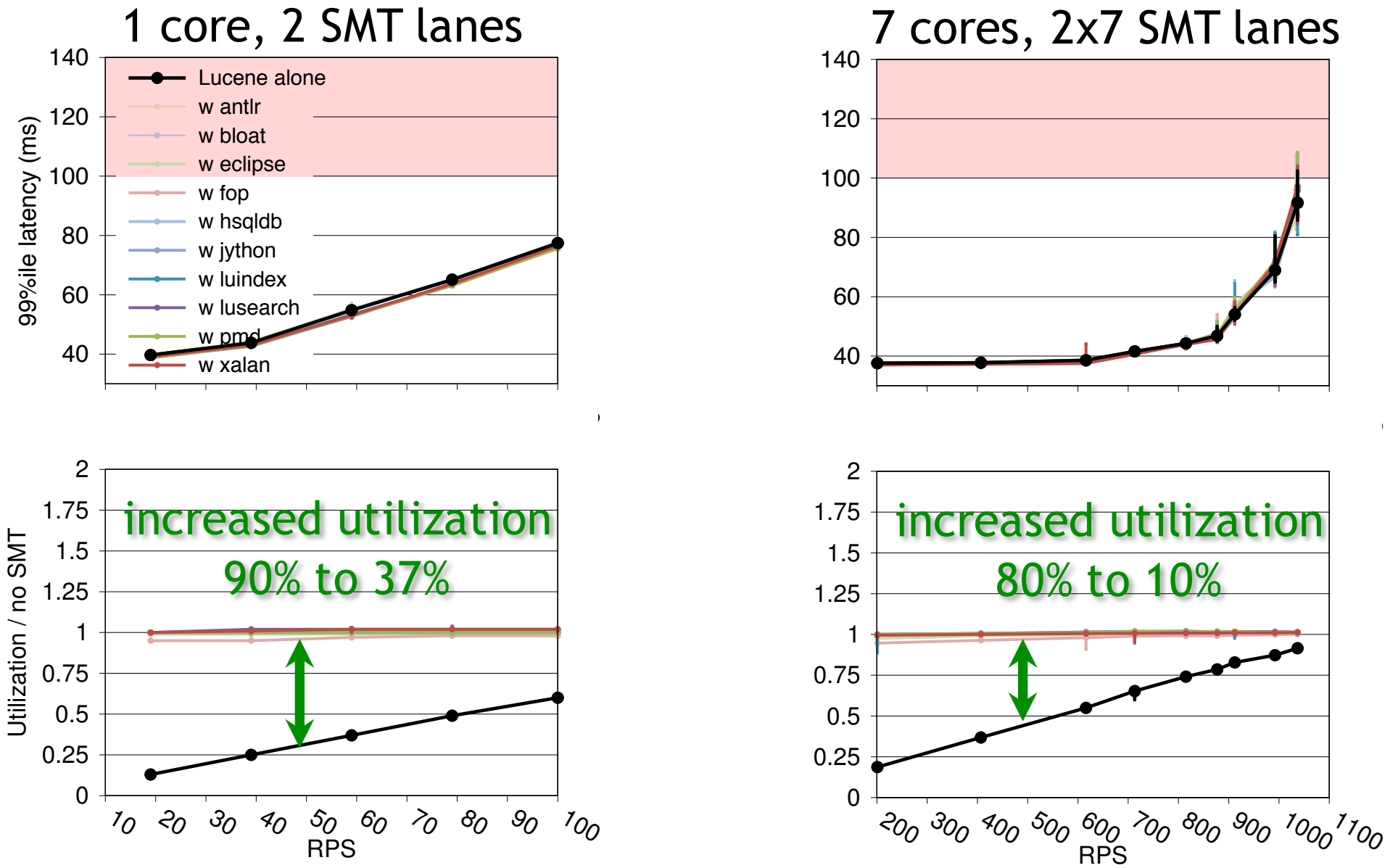



```
/* fast path check injected into method body */
check:
if (!request_lane_idle)  1
    slow_path();

slow_path() { nanonap(); }  2
```

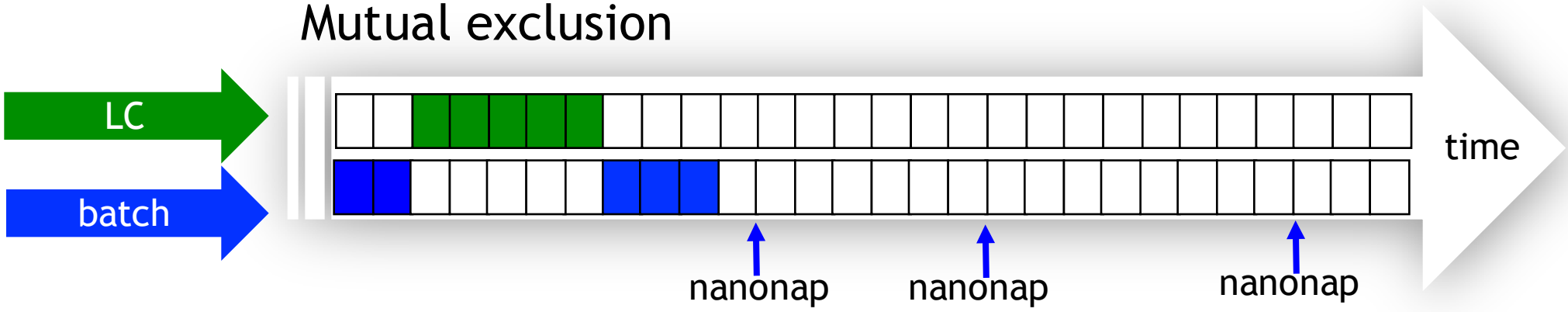
```
/*     maps lane IDs to the running task     */
exposed SHIM signal: cpu_task_map
task_switch(task T) {
   cpu_task_map[thiscpu] = T;
}
idle_task() {
  // wake up any waiting batch thread
  update_nap_flag_of_partner_lane();  3
   ......
}
```

1. Batch thread borrows resources continuously checks LC lane status
2. LC starts, batch calls nanonap() to release SMT hardware context
3. OS touches nap_flag to wake up batch thread

# Results: Borrow Idle



# Beyond Mutual Exclusion




Concurrent co-running with a budget

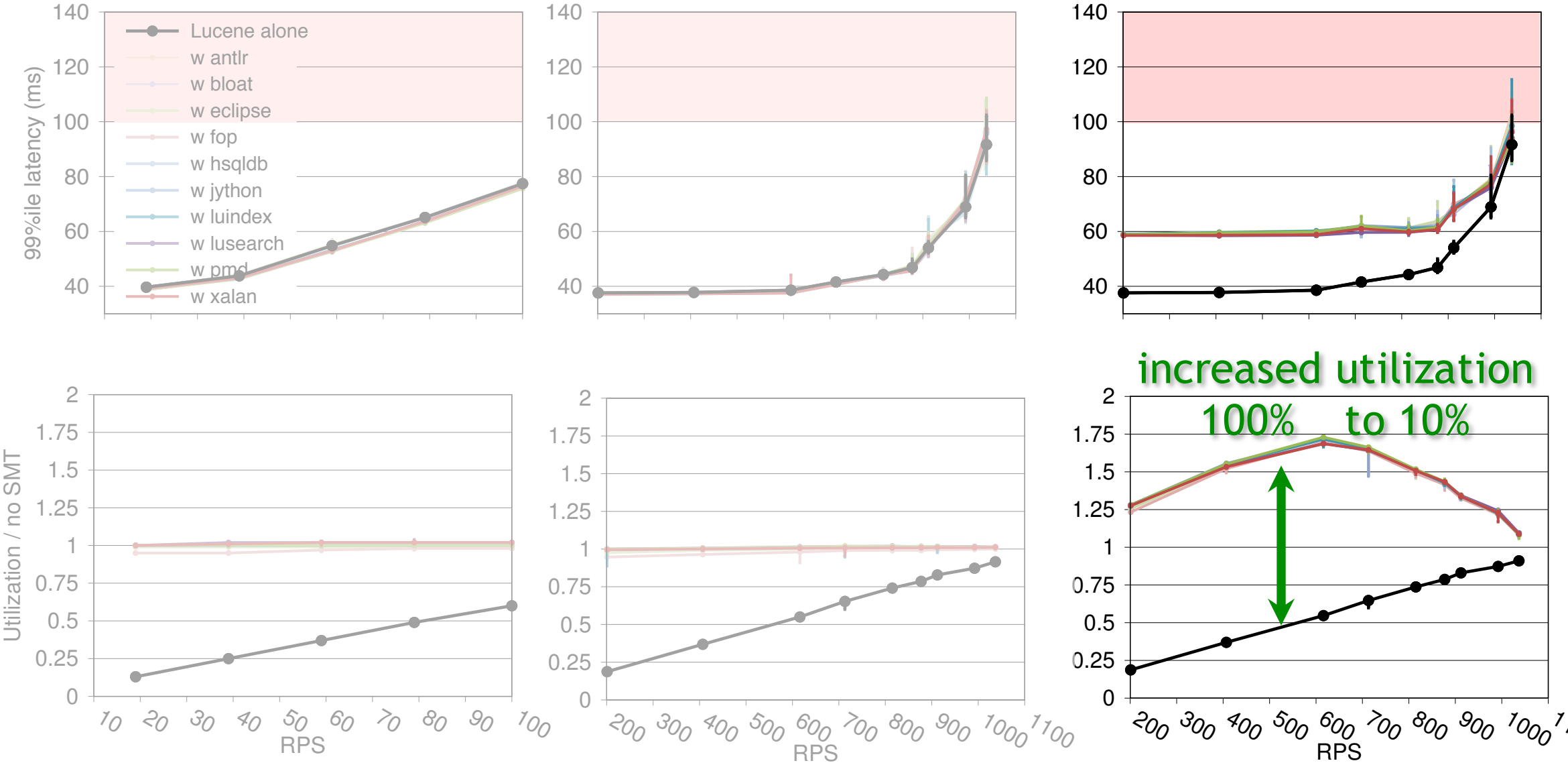
- Borrow idle policy

- Fixed budget policy
- Refresh budget policy
- Dynamic budget policy

## Borrow idle on 1 cores

## Borrow idle on 7 cores

## Dynamic budget 7 cores



# Conclusion

Principled borrowing for SMT

- nanonap() releases STM hardware *without* giving it to OS
- Elfen scheduler implementation & policies
- 90% to 25% increase in utilization
- Same tail latency!?

`https: //github.com/elfenscheduler`

Thank you!