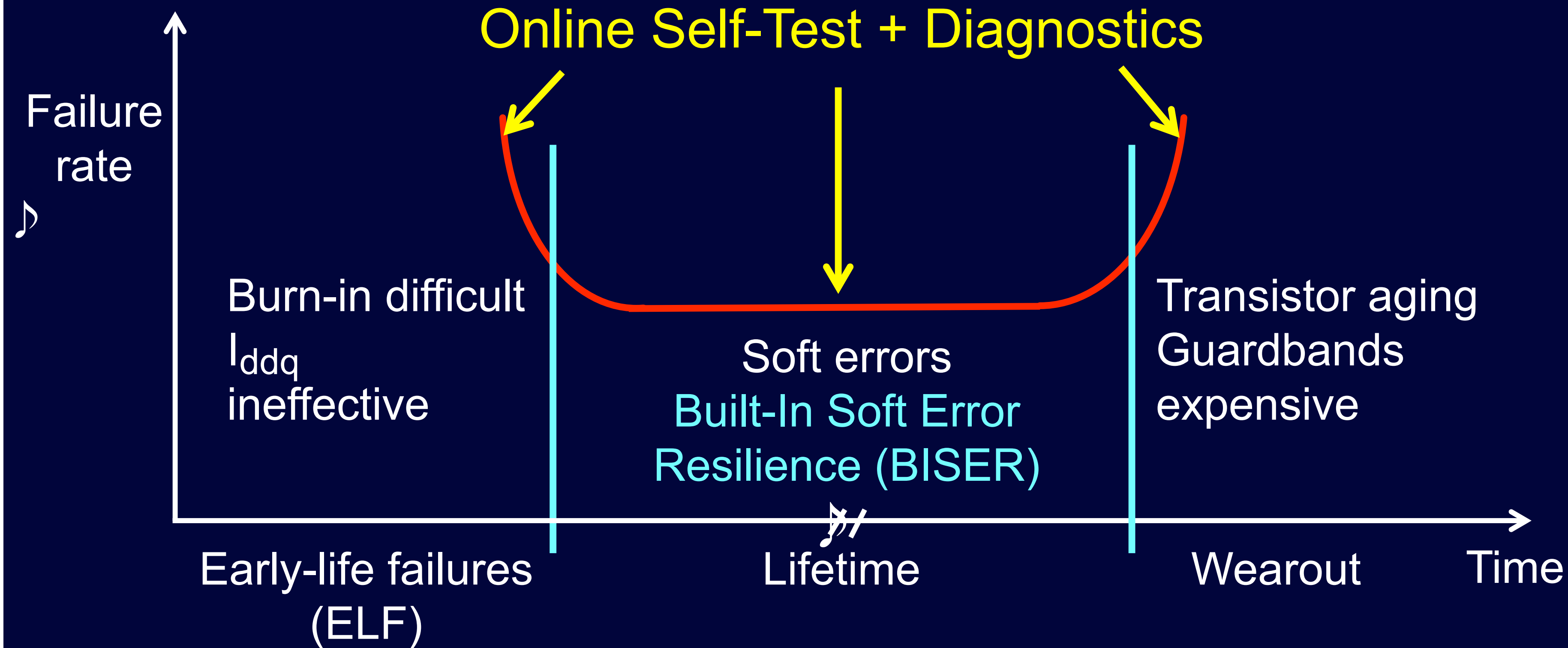
Operating System Scheduling for Efficient Online Self-Test in Robust Systems

Yanjing Li
Stanford University

Onur Mutlu
Carnegie Mellon University

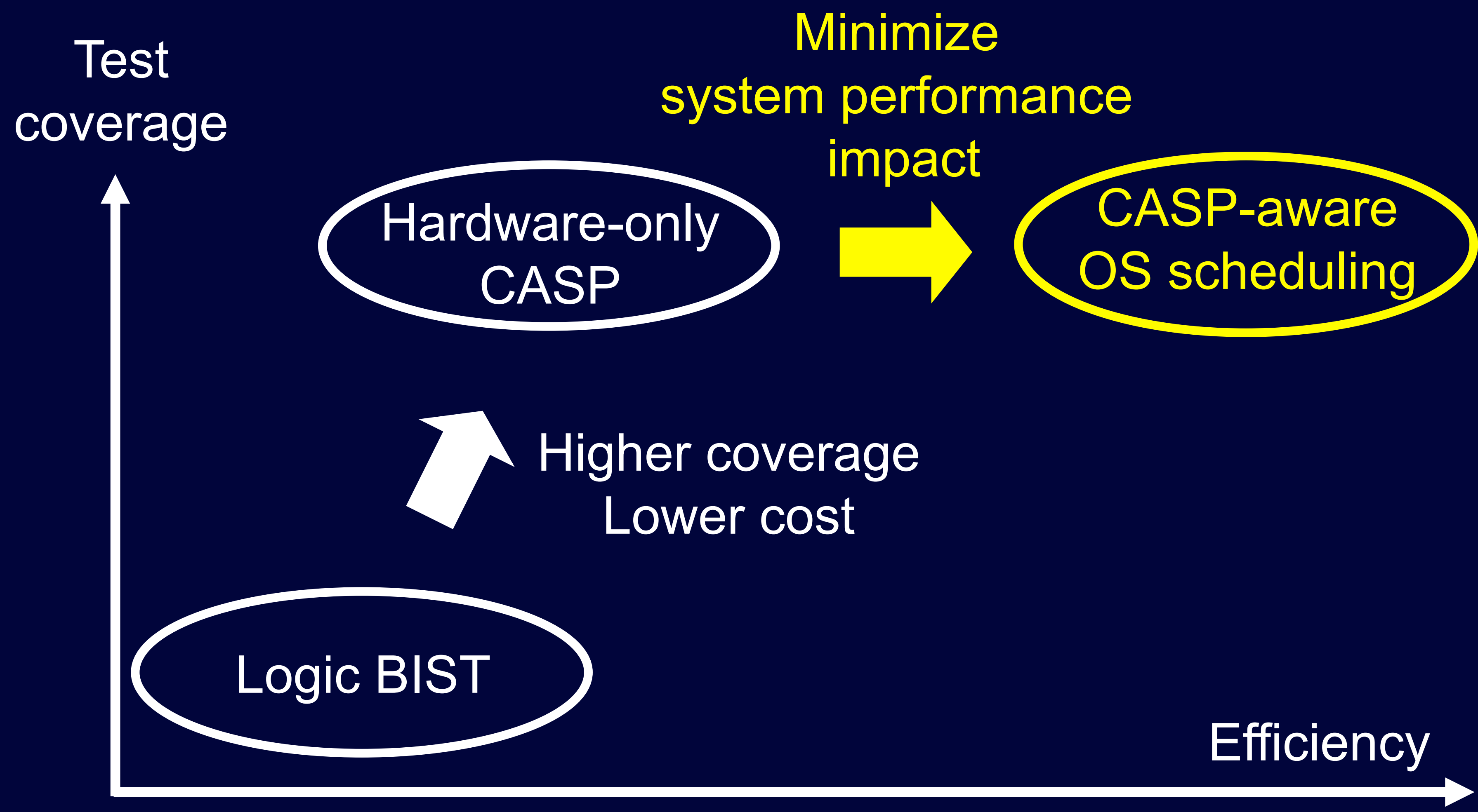
Subhasish Mitra
Stanford University

Why Online Self-Test & Diagnostics?



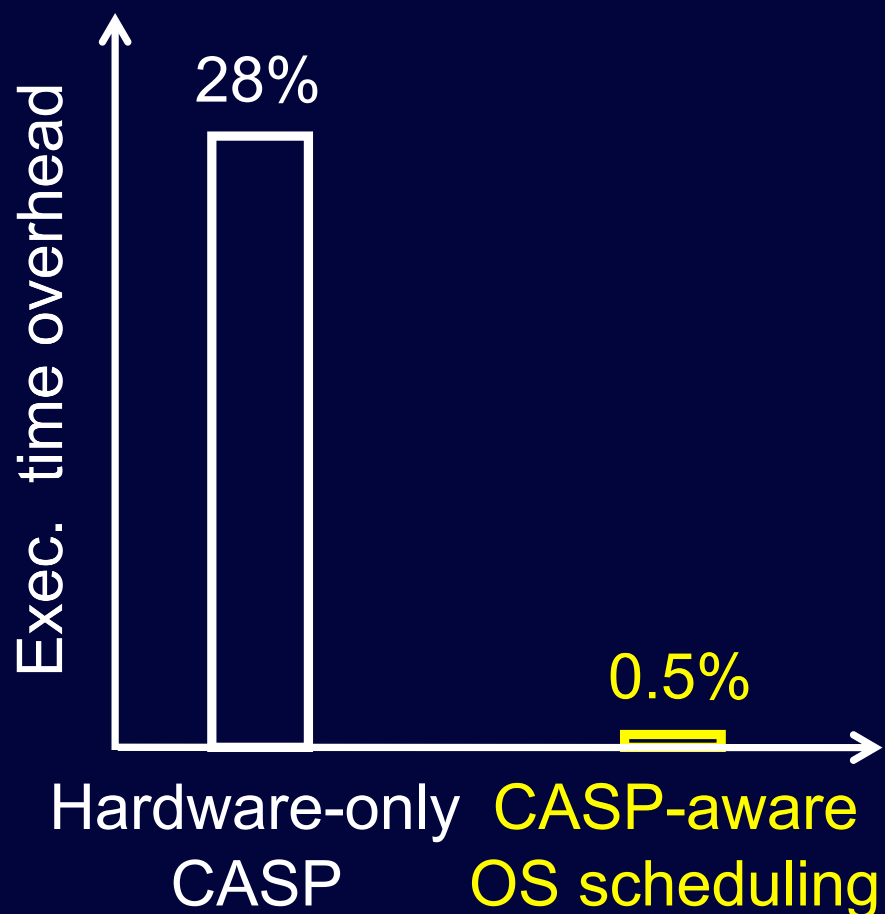
- Application: Failure prediction & detection
- Global optimization → software-orchestrated

Key Message


Test
coverage
Minimize
system performance
impact
Hardware-only
CASP
CASP-aware
OS scheduling
Higher coverage
Lower cost
Logic BIST
Efficiency

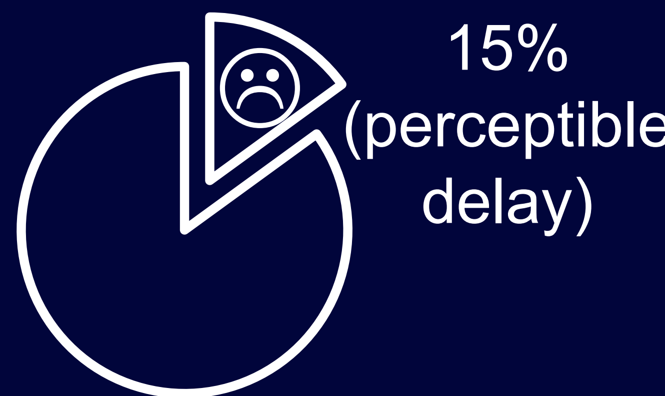
Results from Actual Xeon System

PARSEC performance impact



Text editor “vi” response time

Hardware-only CASP



CASP-aware OS scheduling

CASP runs for 1 sec every 10 sec.

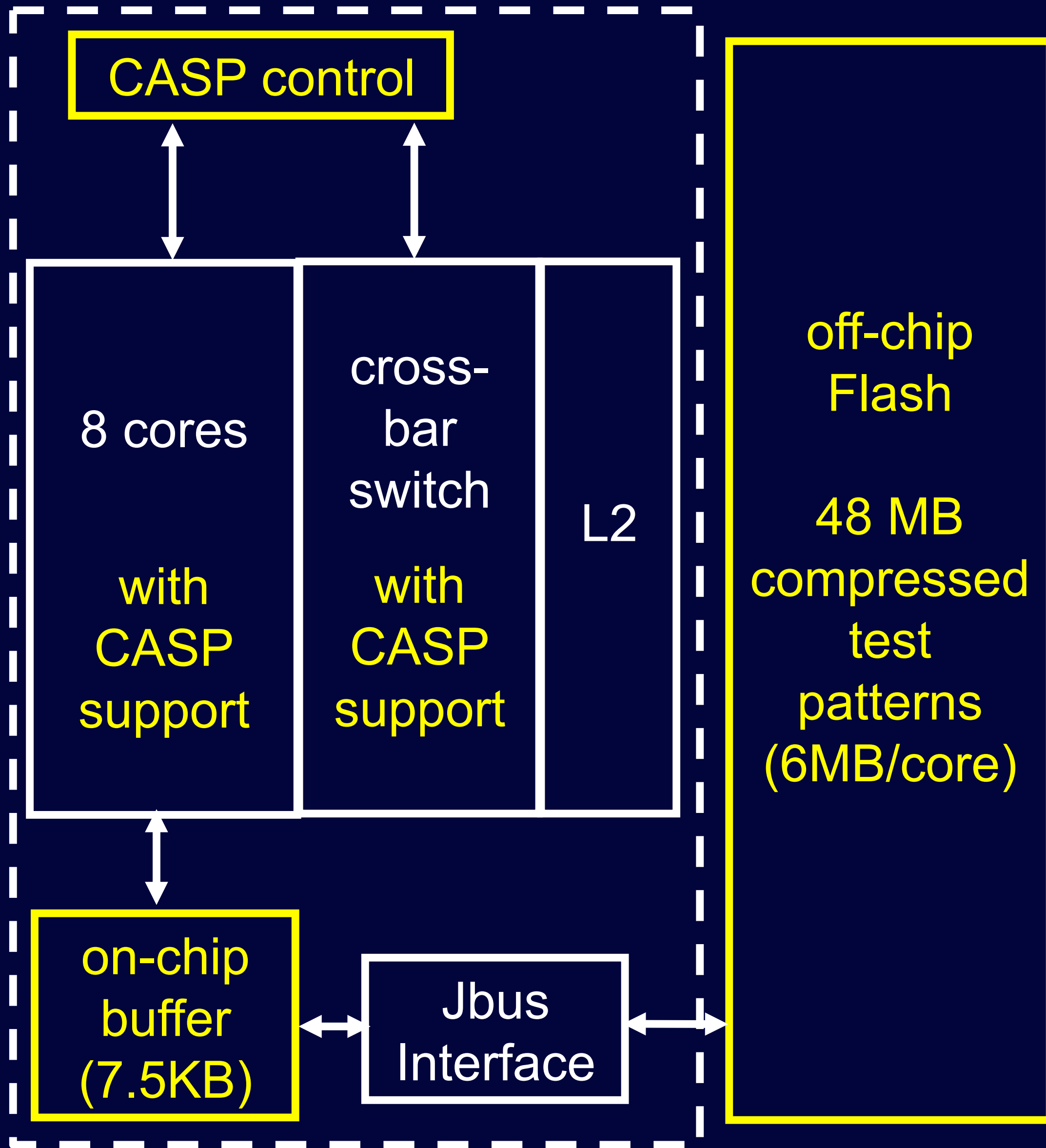
CASP Idea

- [Li DATE 08]
- Concurrent with normal operation
  - ☺ No system downtime
- Autonomous: on-chip test controller
- Stored Patterns: off-chip FLASH
  - ☺ Comparable or better than production tests
  - ☺ Test compression: X-Compact

Major Technology Trends Favor CASP

CASP Study: SUN OpenSPARC T1

- Test coverage
  - Stuck-at: 99.5%
  - Transition: 96%
  - True-time: 93.5%
- Test power
  - ≈ normal operation
- 0.01% area impact



~ 8K Verilog LOC modified (out of 100K+)

Hardware-only CASP Limitations

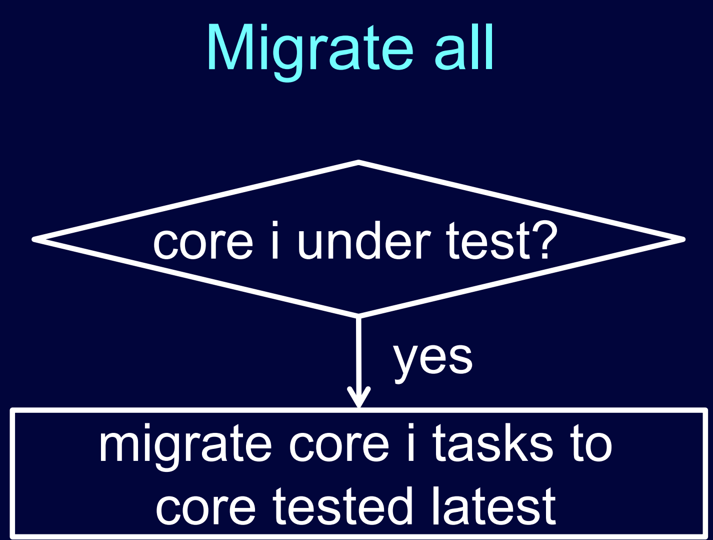
- Hardware-only
  - No software interaction (e.g., OS)

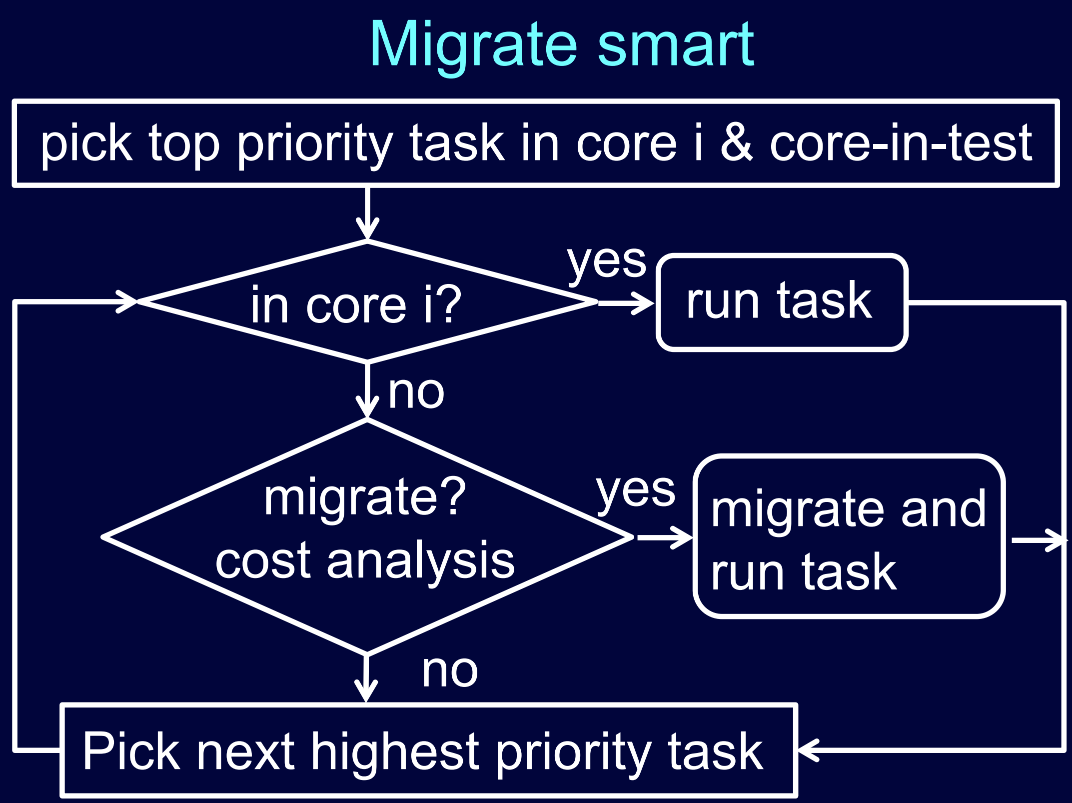
☹ Visible performance impact

- Core unavailable during CASP → task stalled
  - Scan chains for high test coverage
    - Comprehensive diagnostics
    - Required for acceptable reliability

CASP-Aware OS Scheduling

- Key idea: make OS aware of CASP
  - ❖ Tasks scheduled / migrated around CASP



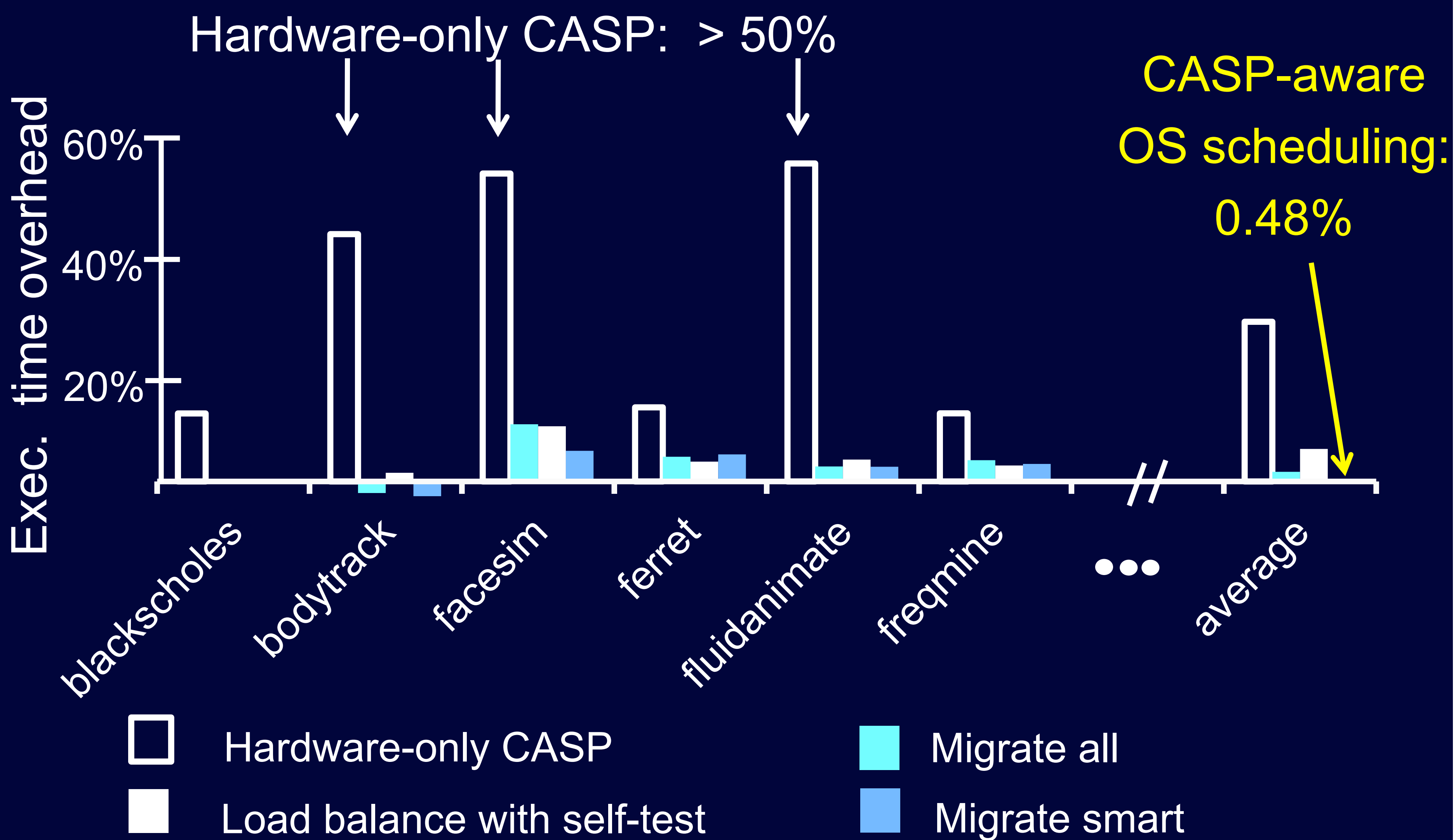


- Scheduling for interactive / real-time tasks: see paper

Evaluation Setup

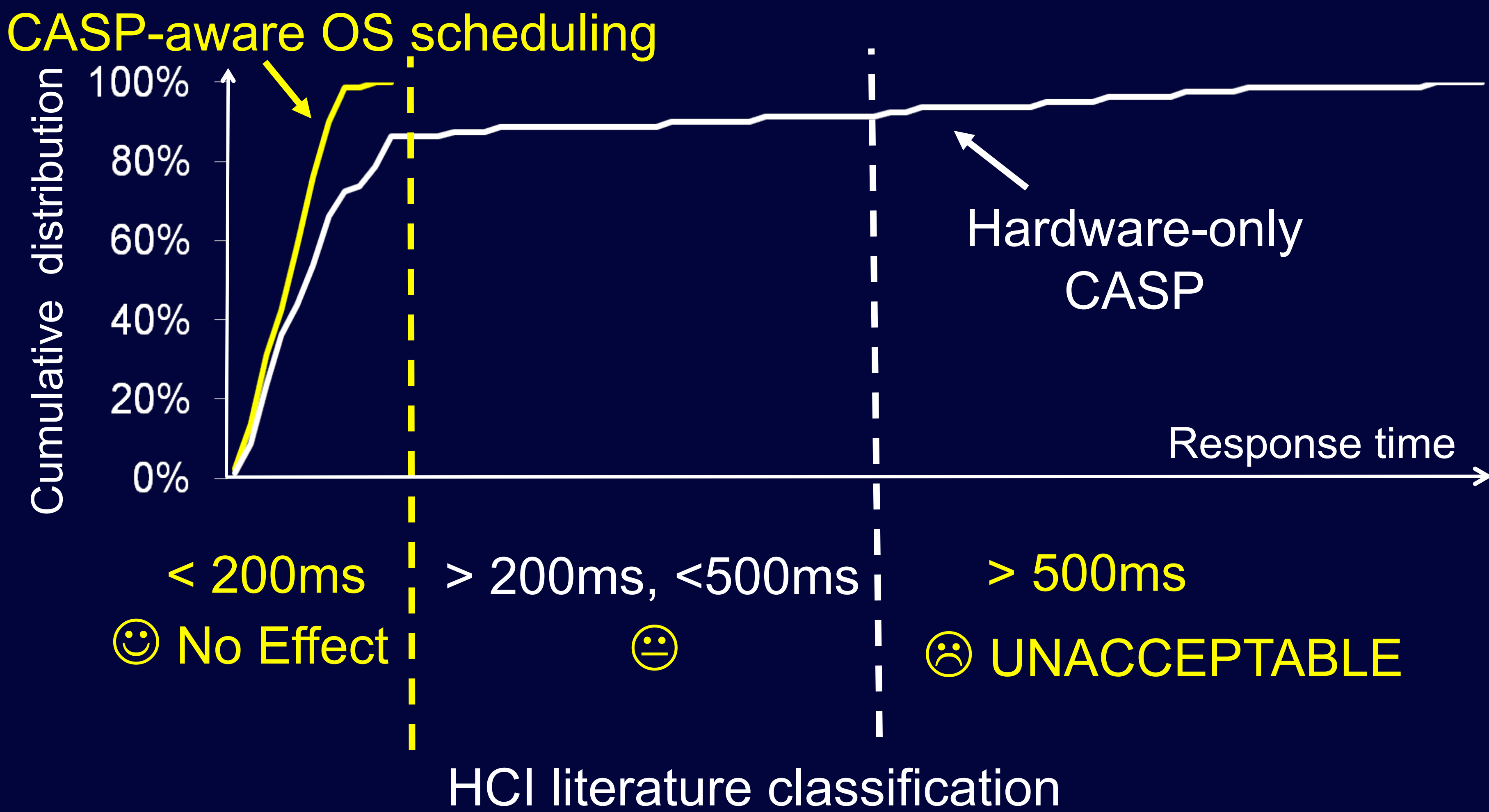
- Platform
  - 2.5GHz dual quad-core Xeon
  - Linux 2.6.25.9 (scheduler modified)
- CASP test program: idle test thread
  - Sufficient for performance studies
- CASP configuration
  - Runs 1 sec every 10 sec
  - More parameters in paper

Results: Computation-Intensive Applications



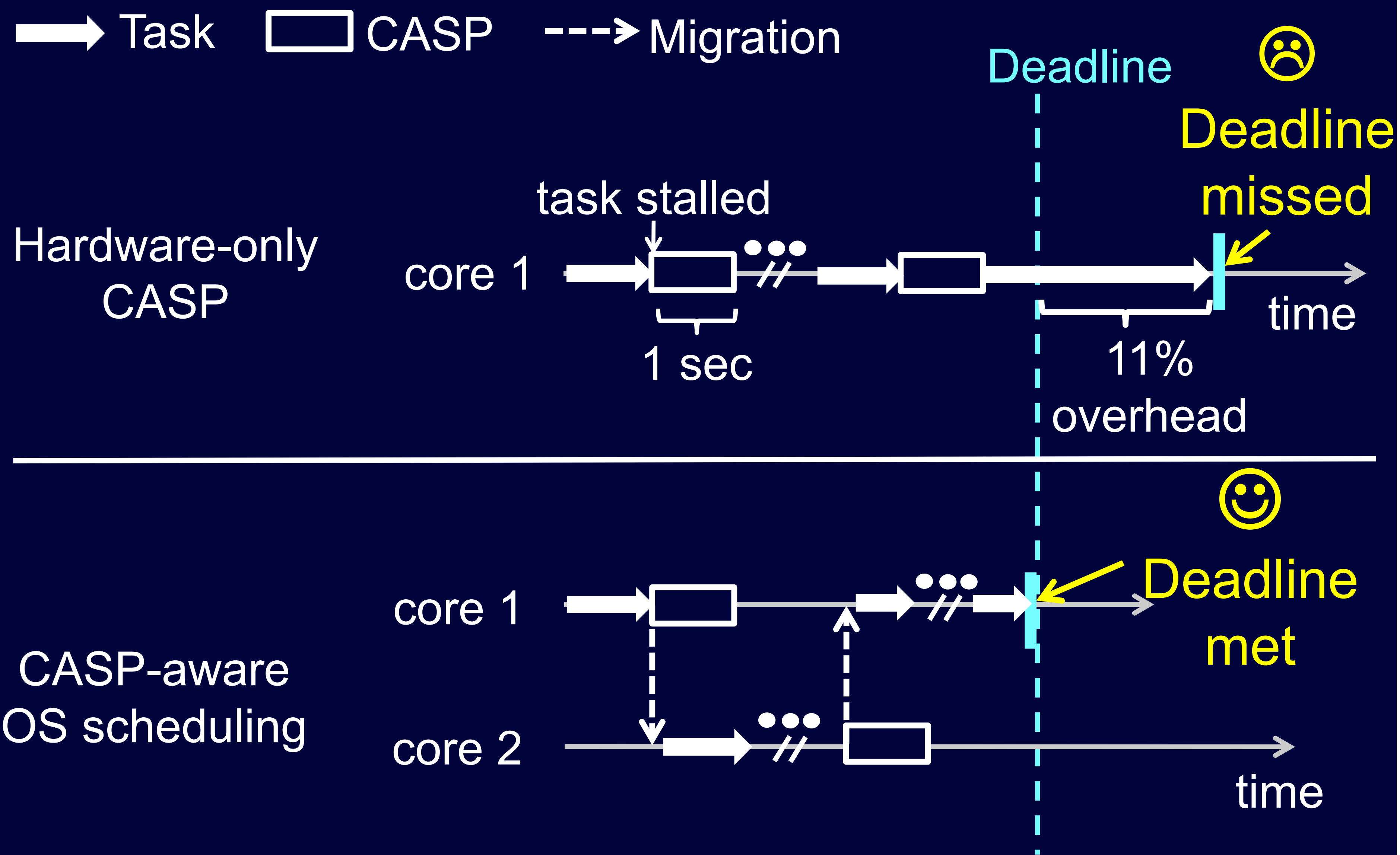
Workload: 4-threaded PARSEC

Results: Interactive Applications



Workload: firefox

Results: Soft Real-Time Applications



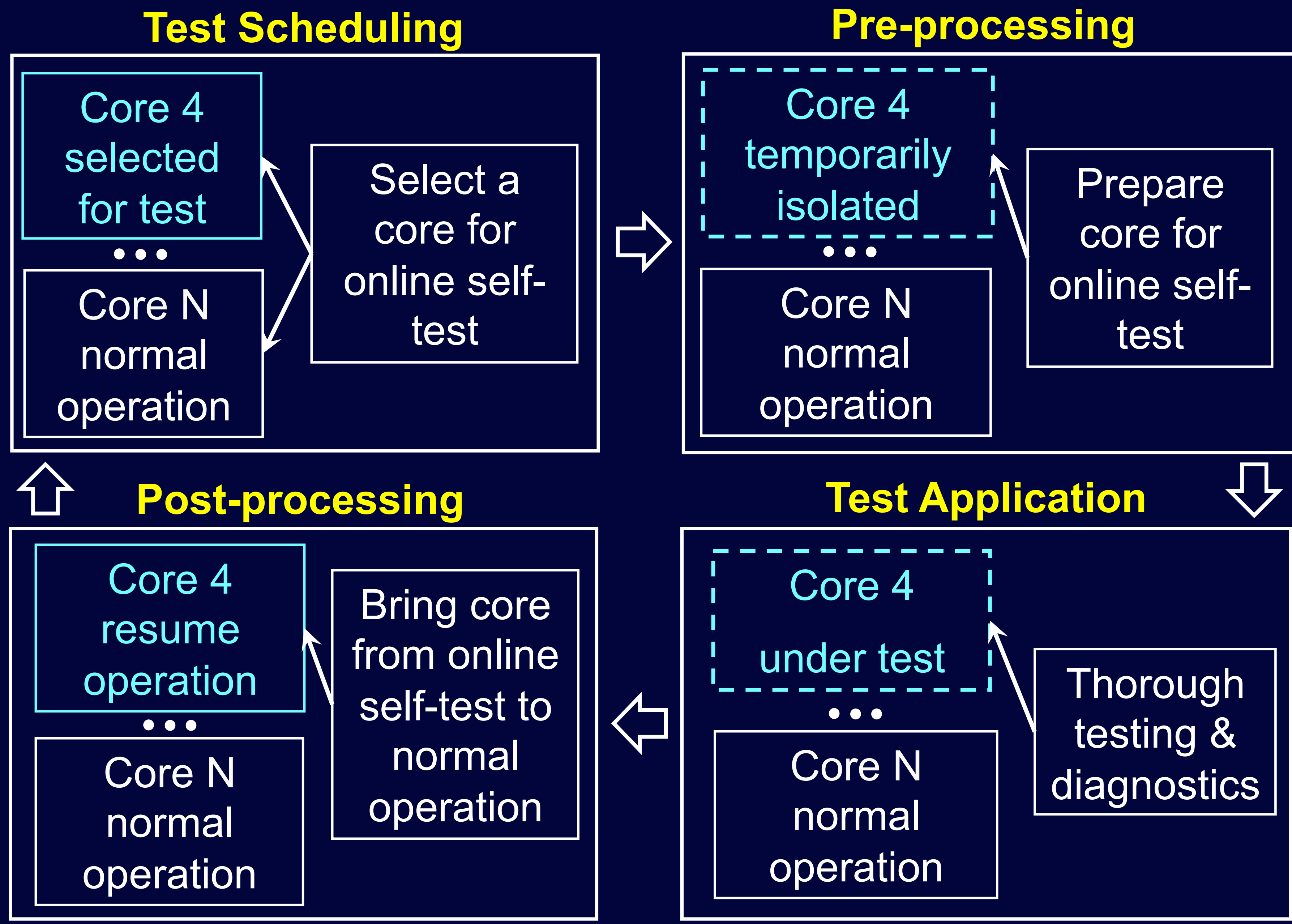
Workload: h.265 encoder

Conclusions

- CASP: efficient, effective, practical
- Hardware-only CASP inadequate
    - Visible performance impact
        - Shown in real system
- CASP-aware OS scheduling
    - Minimal performance impact
        - Wide variety of workloads
        - Shown in real system

Backup Slides

Hardware-only CASP Test Flow



Test Flow with CASP-Aware OS Scheduling

Test Scheduling ⇨ **CASP-Aware OS Scheduling Starts**

1. Informs OS test begins by interrupted	⇨	2. OS performs scheduling around tests

⇩

Pre-processing

⇩

Test Application ⇦ **Post-processing**

⇧

Informs OS test completes by interrupt

CASP-Aware OS Scheduling Ends

⇧

Test Scheduling

Algorithms for Tasks in Run Queues

- Migrate_all
  - Migrate all tasks from test core to be tested
- Load_balance_with_self_test
  - Workload balancing considering self-test
- Migrate_smart
  - Migrate tasks based on cost-benefit analysis

Scheduling for Run Queues: Scheme 1

- Migrate_all
- Migrate all tasks from core-under-test
  - Except for non-migratable tasks
    - e.g., certain kernel threads
- Destination
  - core that will be tested furthest in the future

Scheduling for Run Queues: Scheme 2

- Load_balance_with_self_test
- Online self-test modeled as highest priority task
  - weight of workload ~90X of normal tasks
- Load balancer automatically migrates other tasks
- Bound load balance interval
  - smaller than interval between two consecutive tests
  - Adapt to the abrupt change in workload with test

Scheduling for Run Queues: Scheme 3

- Migrate_smart: migrate based on cost-benefit analysis
  - Cost: wait time remaining + cache effects
- When test beings
  - Migrate all tasks to idle core (if exists)
- During context switch for cores not under test
  - Worthwhile to “pull” task from core(s) under test?
    - Yes: migrate and run task from core under test
    - No: don’t migrate

Scheduling for Wait Queues

- Task woken up: moved from wait queue to run queue
  - Run queue selection required
- Follow original run queue selection
  - If queue selected is not on a core under test
- O/W pick a core tested furthest in the future
- Quick response for interactive applications
- Used with all three run queue scheduling schemes

Scheduling for Soft Real-Time Applications

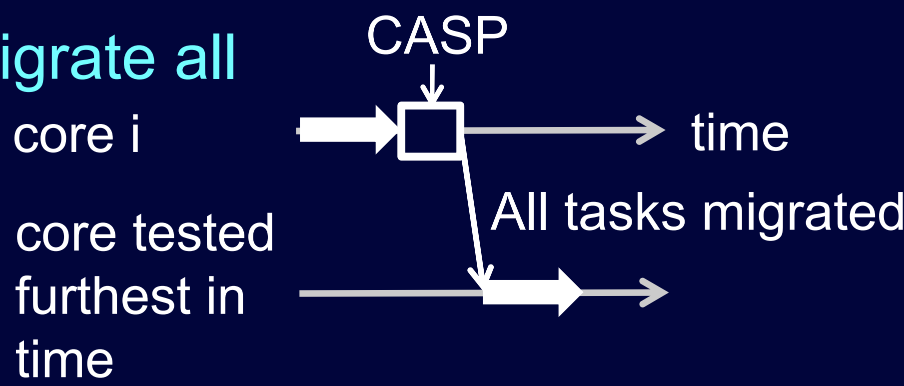
- Separate scheduling class for real-time applications
  - Higher priority than all non real-time apps
  - More likely to meet real-time deadlines
- Migrate real-time tasks from core to be tested to
  - core that has lower-priority tasks

    and
  - core that will be tested furthest in the future
- Used with all three run queue scheduling schemes

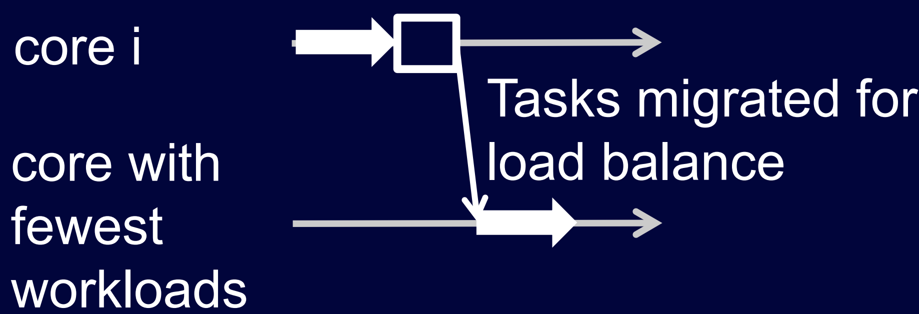
CASP-Aware OS Scheduling Summary

Computation-Intensive Tasks

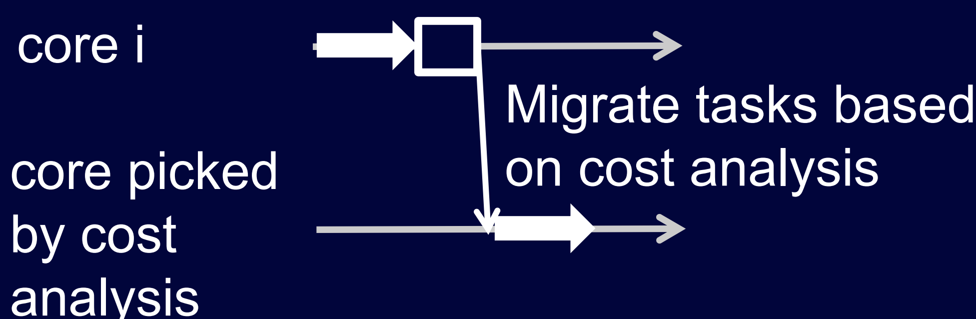
Migrate all



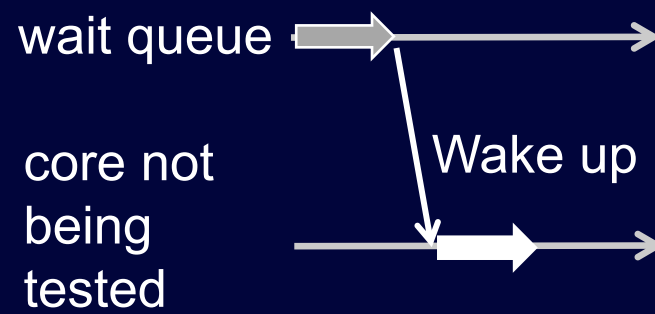
Load balance with self-test



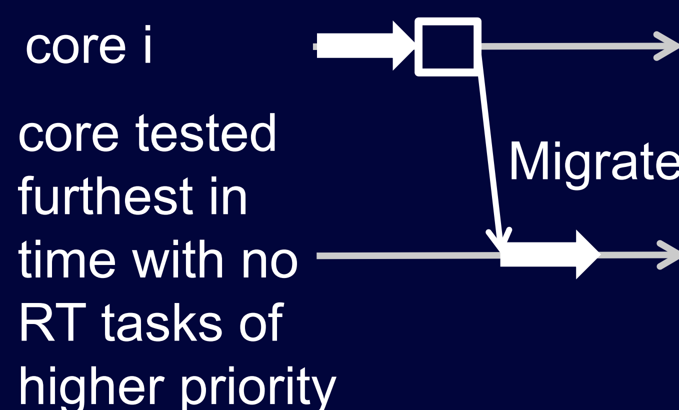
Migrate smart



Interactive Tasks



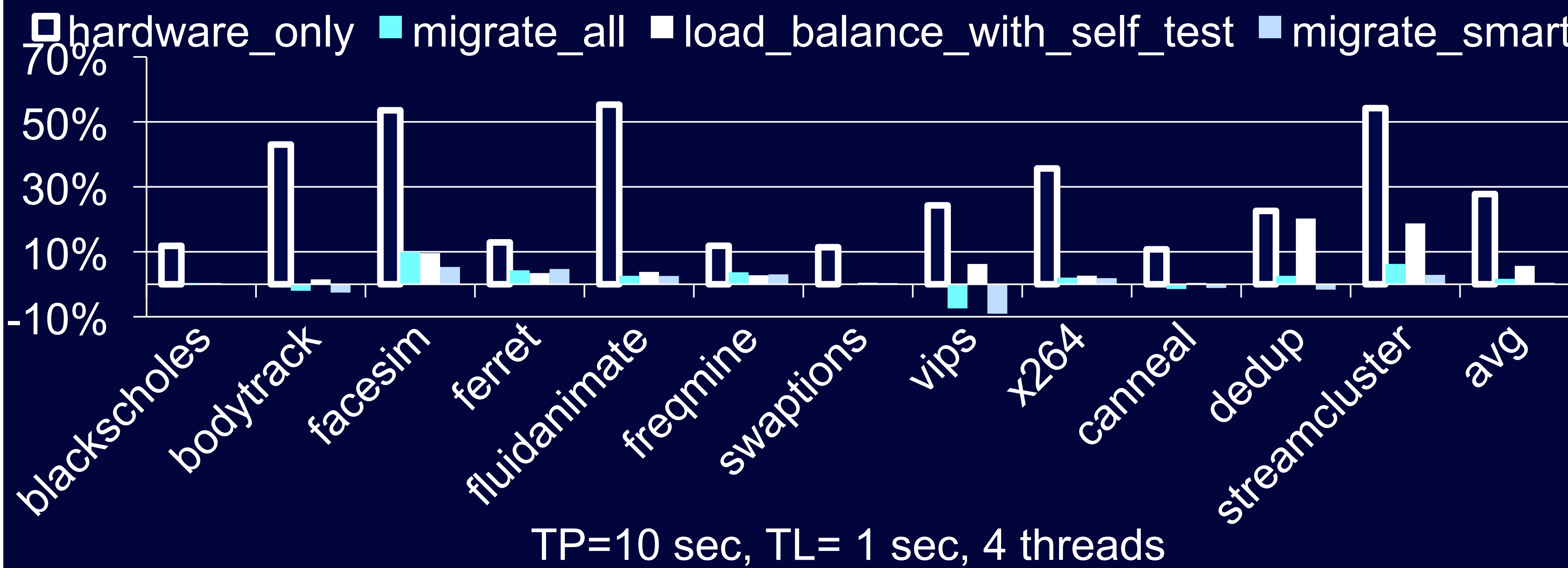
Soft Real-Time (RT) Tasks



Workloads Evaluated

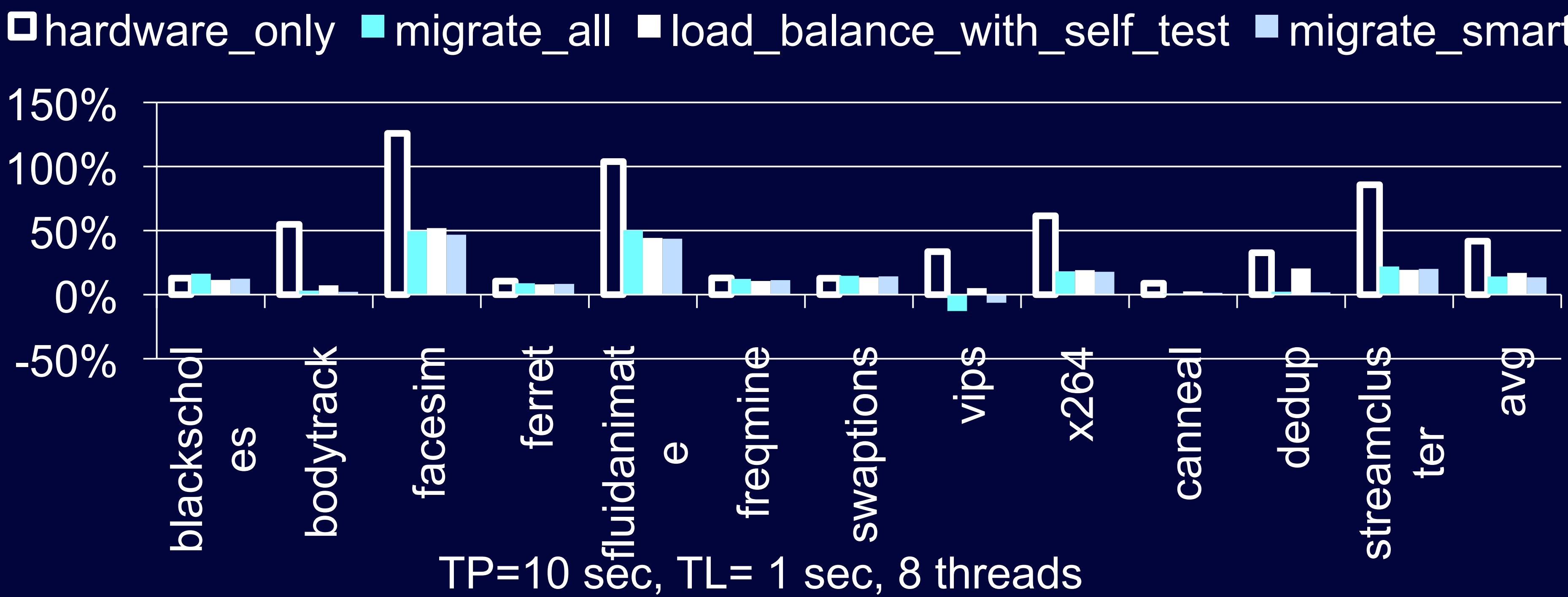
- Computation-intensive (PARSEC)
  - Tasks in run queues
- Interactive (vi, evince, firefox)
  - Tasks in wait queues
- Soft real-time (h.264 encoder)
  - x264 from PARSEC with RT scheduling policy

Results: 4-threaded PARSEC Applications



- ☹ Hardware_only: significant performance impact
- Migrate_smart: best approach
  - 0.48% overhead on average; ~5% max
- Migrate_all: comparable results

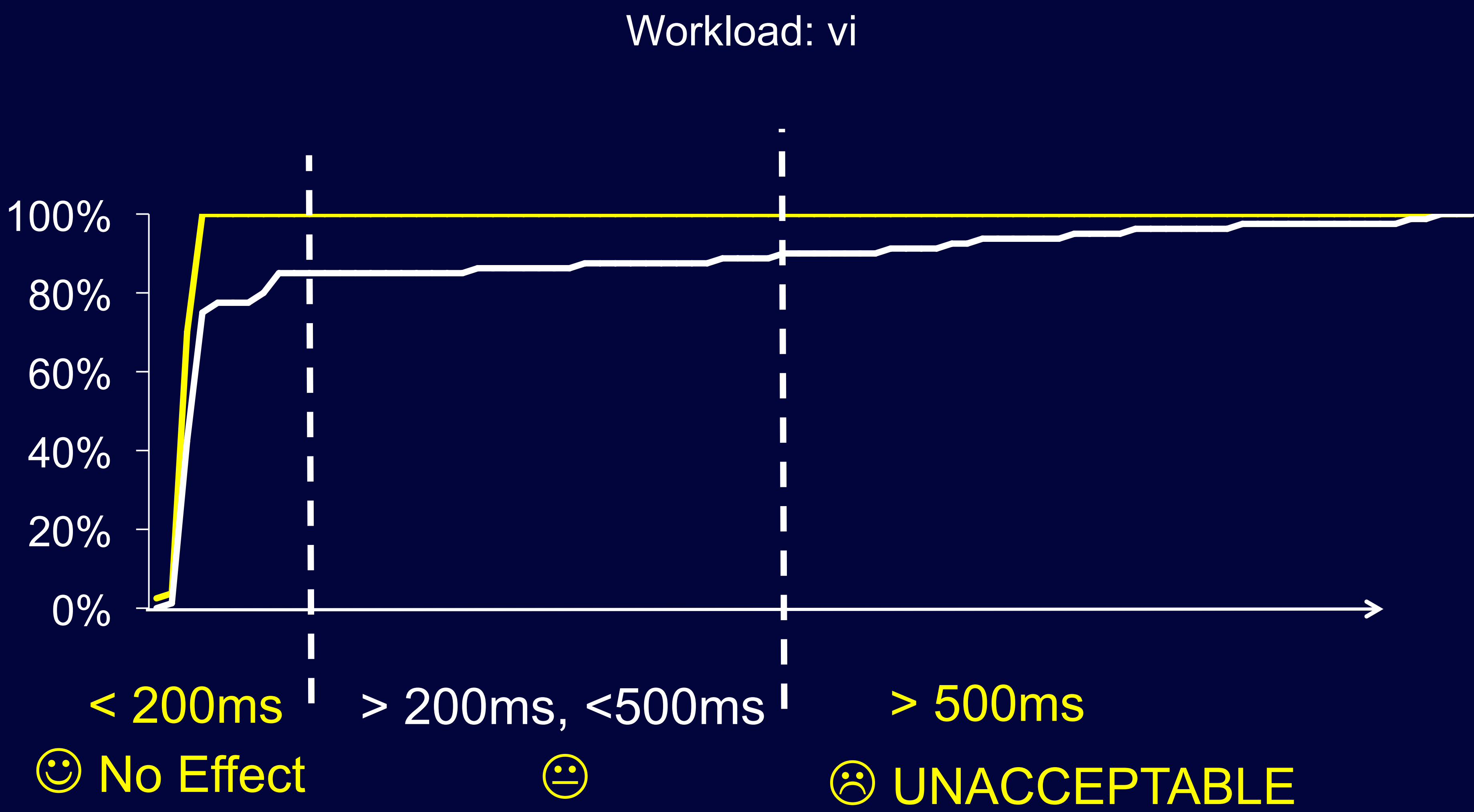
Results: 8-threaded PARSEC Applications



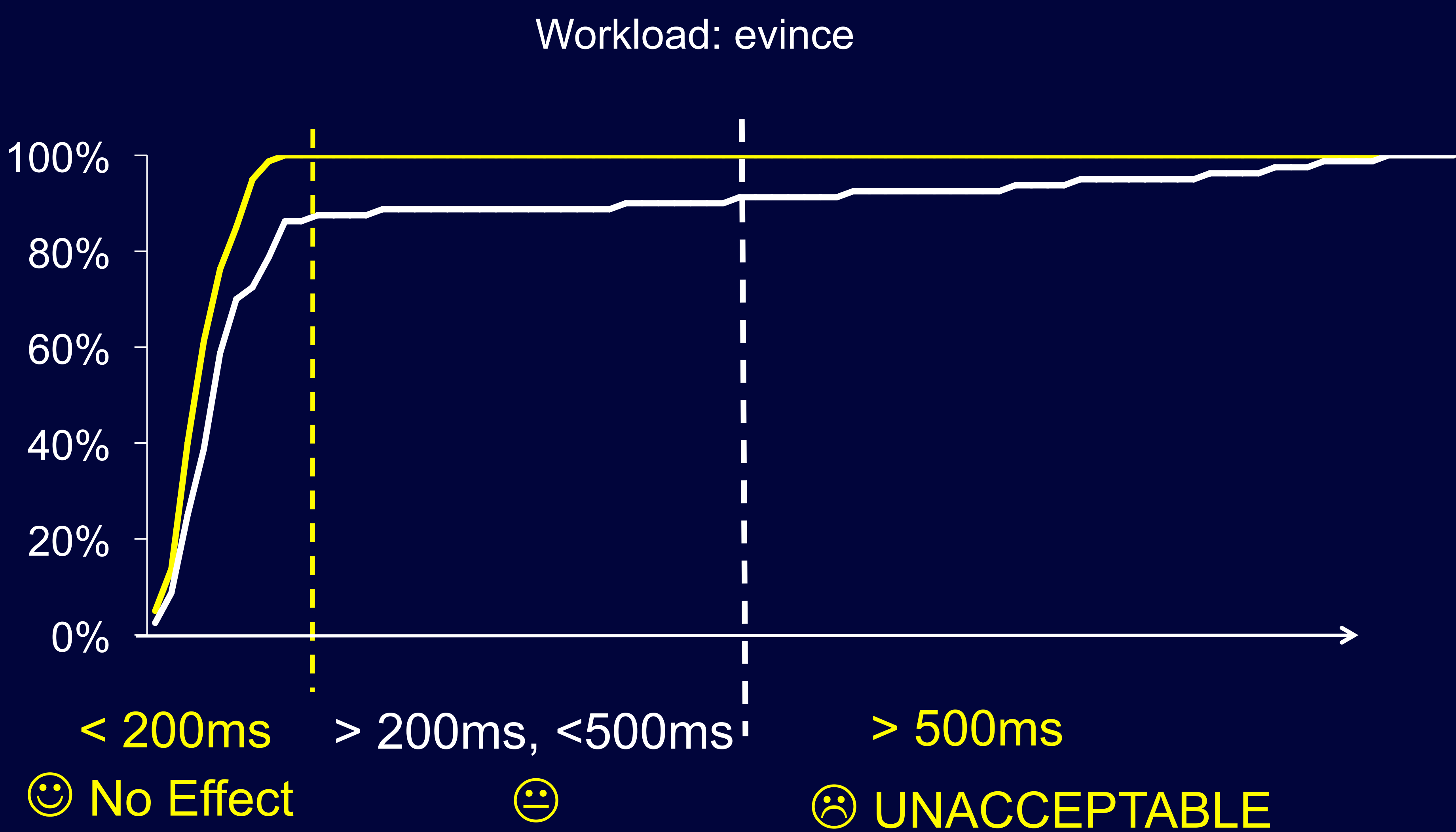
☹ hardware-only: significant performance impact

- Our schemes
  - ~ 11% (i.e. TL/(TP-TL))
  - Inevitable due to constraints in resources

Results: Interactive Applications



Results: Interactive Applications (2)



Results: Soft Real-Time Applications

- 8 single-threaded h.264 encoder
  - 7 high priority: real-time priority level 99
  - 1 low priority: real-time priority level 98

TP=10 sec, TL= 1 sec

Configuration	hardware-only	Our schemes
Not fully loaded	11% for 7 apps.	No penalty for 7 apps.
Fully loaded	11% for all 8 apps.	0% 7 higher-priority apps. 87% for low-priority app.

☹ hardware-only: deadlines missed

- Our schemes: Deadlines met