Palirria:
Accurate On-line Parallelism Estimation for Adaptive Work-Stealing

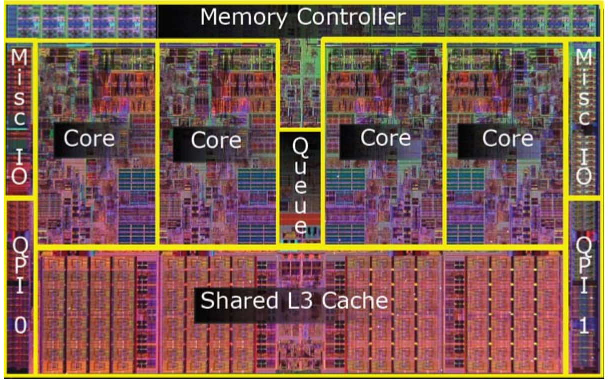
Georgios Varisteas, Mats Brorsson

PMAM, February 2014

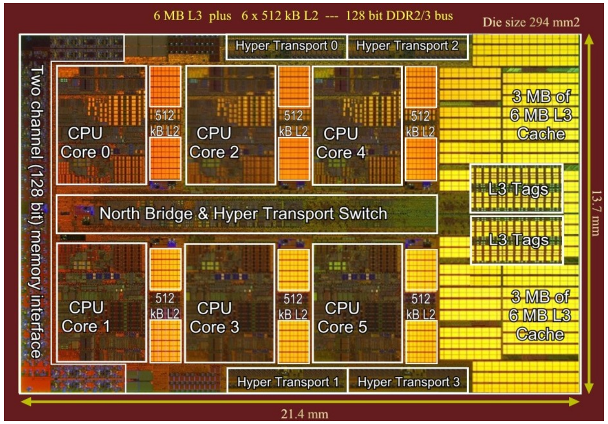
KTH
Royal Institute of Technology

Motivation

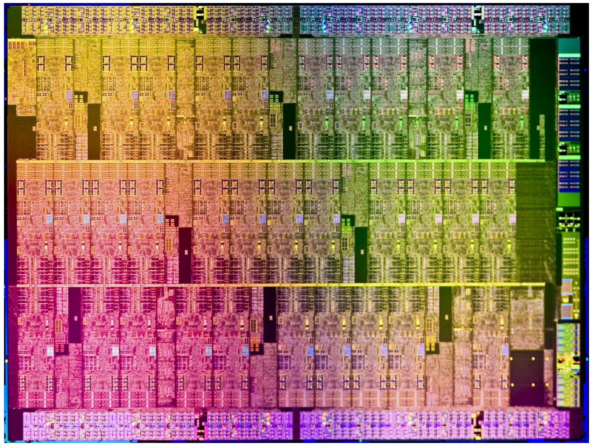
- Increasing number of cores per die
  - Worrisome power budget
  - Unequipped OS resource management



Intel i7



AMD Phenom II



Intel Xeon Phi

Motivation: Scheduling

- Keep the system utilized just enough to lower the power budget
  - Conservative core allotment
- Allot cores so that application performance is maximized
  - Liberal core allotment

Dynamic Multiprogramming

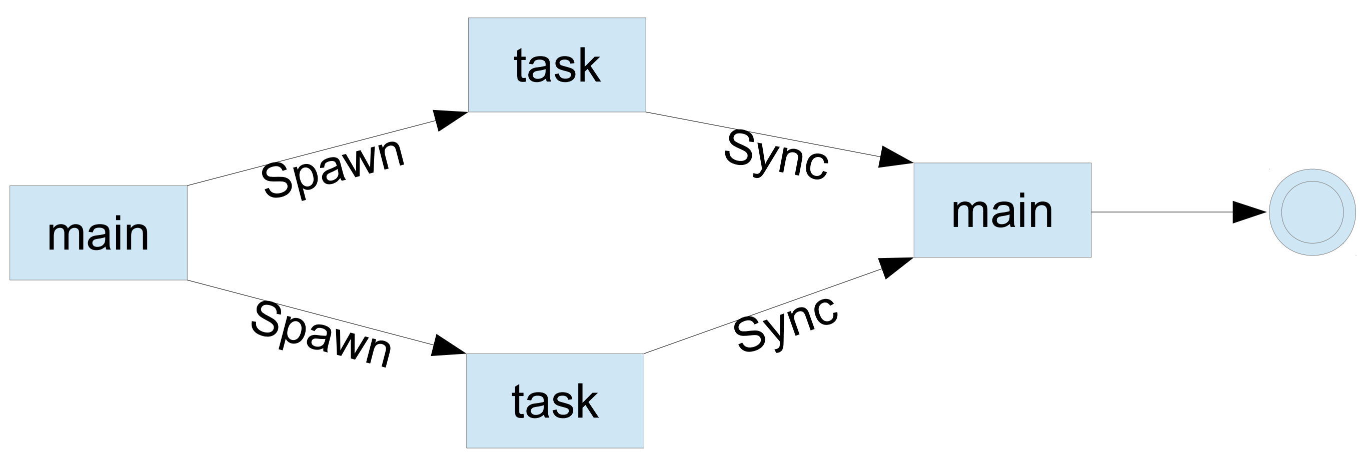
- **Adapt** allotment size to actual application processing requirements
  - Each application must provide knowledge on its exposed parallelism
  - The OS can intelligently partition available resources

Summary

- Palirria
  - Method for estimating a task-based workload's concurrency
    - Accurate, lightweight, online, no training
  - Built upon a variation to traditional work-stealing
    - Deterministic Victim Selection (**DVS**) replaces victim selection in any work-stealing scheduler
  - ➔ Good performance with less worker threads for workloads of irregular parallelism

Task-centric programming models

- Expose independent computations, executable in parallel
- Adapt easily
  - Logical, not bound to hardware



Work Stealing

- Pre created pool of worker threads
- Local task queue per worker thread

- Workers place spawned tasks in their queue
- If worker idle:
    1. Steals from its own task-queue
    2. Steals from a remote task-queue (victim)
- **Victim selection**: find a non-empty remote queue
    - Traditionally employs some randomness

From Estimation to Adaptation

- Estimate a workload's parallelism
  - Metric for quantifying parallelism
- Decide adequate allotment size
  - Conditions for requesting change

Parallelism Estimation: Metrics

- Traditional black box approaches
  - ➔ Measure cycles or other perf. counters
  - ✗ Estimate based on past behavior
  - ✗ Hardware dependent
- Could we exploit the scheduling?
  - ➔ Parallelism currency: task-queue size
  - ✓ Estimate based on future processing needs
  - ✓ Hardware agnostic

Parallelism Estimation: Decision

- Maybe add more workers
  - **Over-utilized** allotment
  - Non empty task queues
- Probably need less workers
  - **Under-utilized** allotment
  - Empty task-queues

Parallelism Estimation: Issues

- **Threshold:** What queue size should decide over-utilization?
- **Overhead:** How many workers should qualify this condition?
- **Balance:** What if some workers are over- and others under- utilized?

- Random victim selection hinders estimation

Scheduling Support for Parallelism Estimation

- Must normalize work discovery latency
  - Predictable distribution of tasks among workers
- Must infer global status from some workers
  - Uniform distribution of tasks among workers

DVS: Deterministic Victim Selection

- Completely non-random victim selection

➔ Uniformly distributes tasks to all workers

➔ Reduces worst latency for task discovery

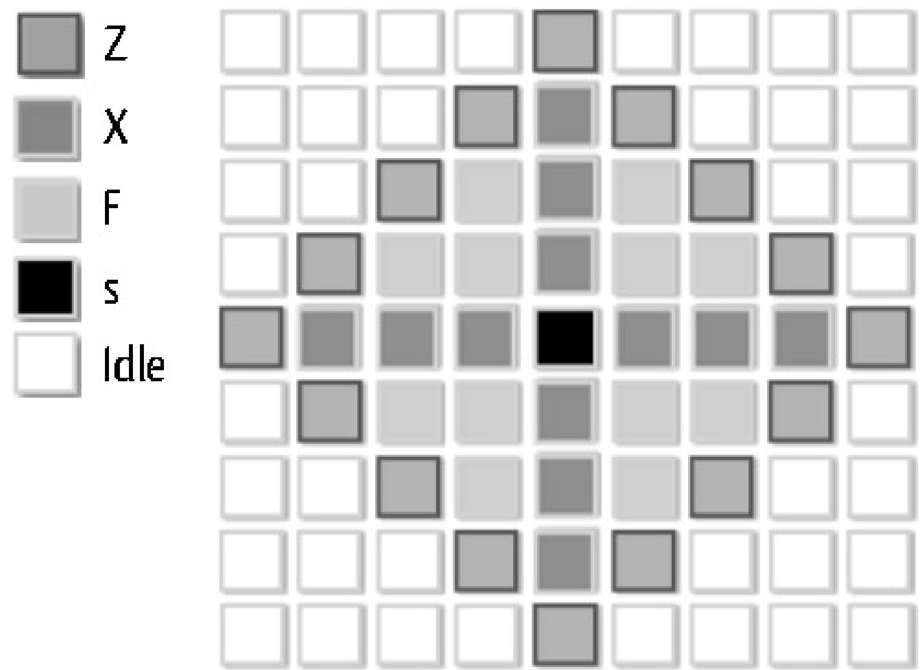
➔ Maintains performance

Paper: G. Varisteas, M. Brorsson. ***DVS: Deterministic Victim Selection to Improve Performance in Work-Stealing Schedulers***. MULTIPROG 2014, Vienna

http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-139400

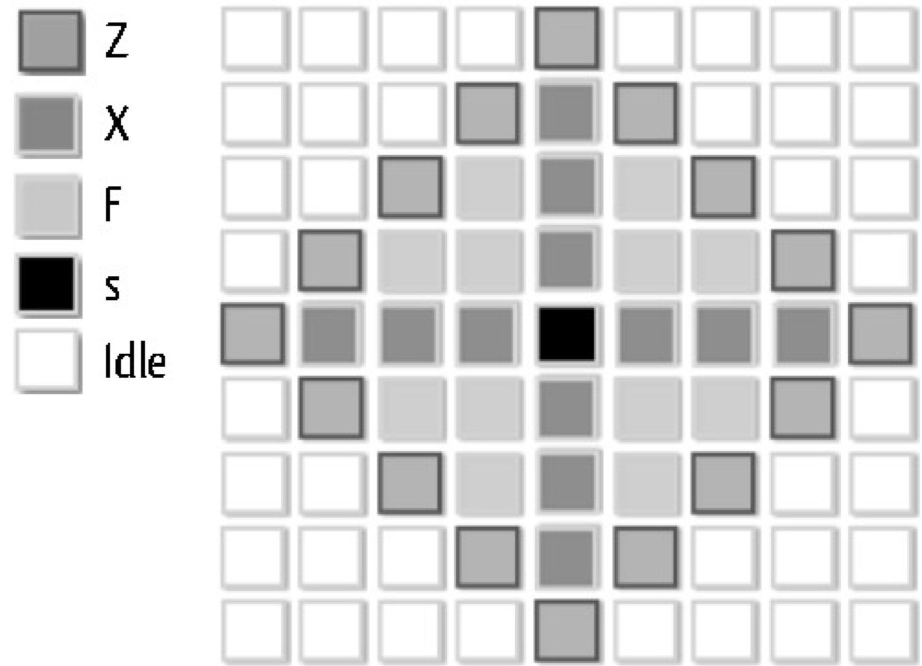
DVS: Worker Classification

- Model available workers as a virtual mesh grid
- Classify workers based on location
  - **X**: vertically & horizontally from the source
  - **Z**: at maximum distance from the source
  - **F**: what remains



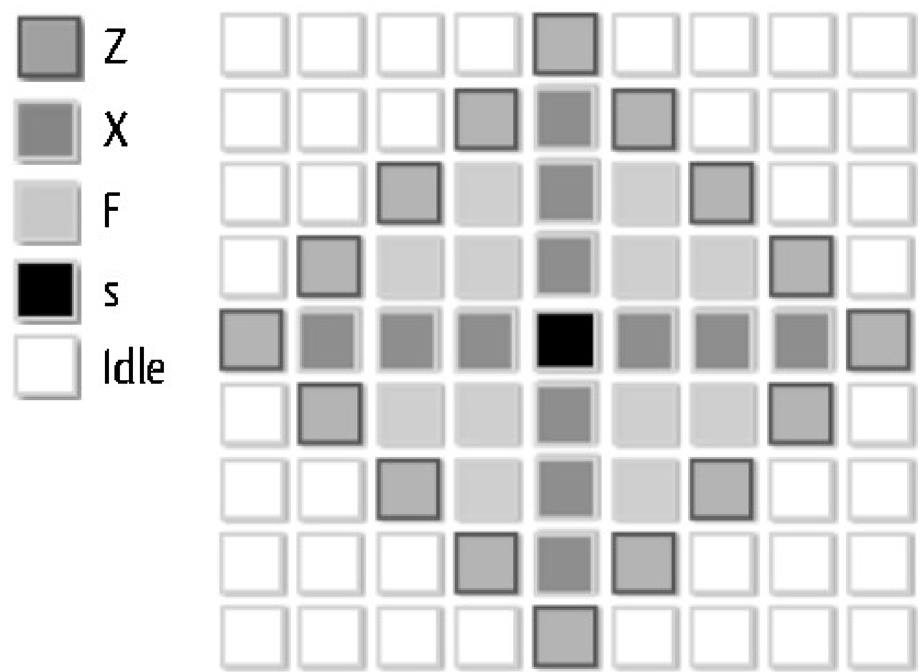
Palirria: Decision Policy

- **Under-utilized**: decrease
  - All workers in Z
    have empty task-queue
- **Over-utilized**: increase
  - All workers in X
    have more than L tasks in
    their task-queue
- **Balanced**: no change
  - If otherwise



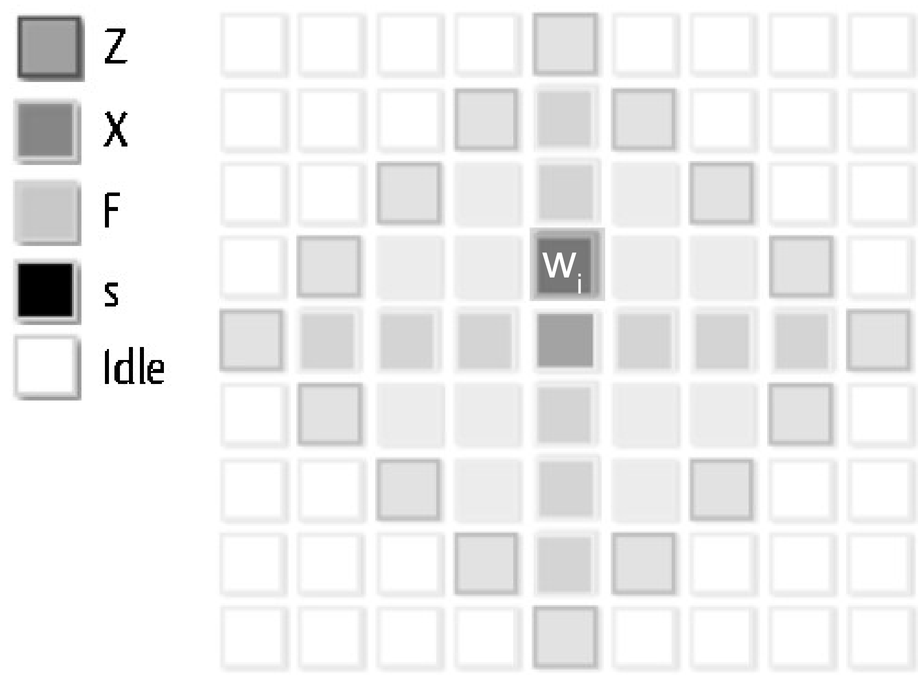
Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims



Palirria: Over-utilization condition

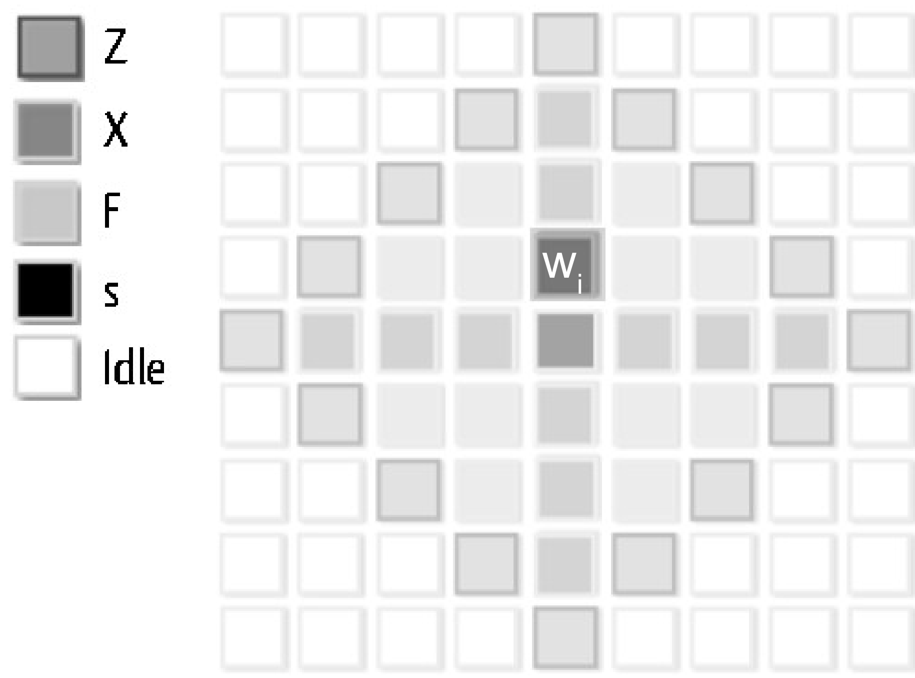
- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims



Palirria: Over-utilization condition

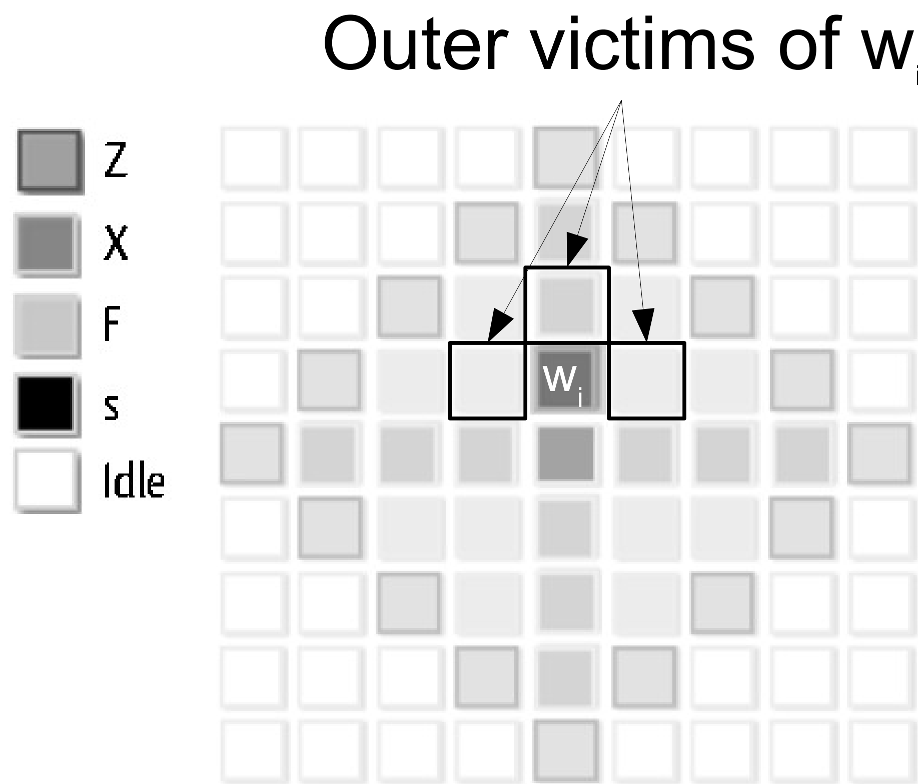
- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims

Outer victims of w_i



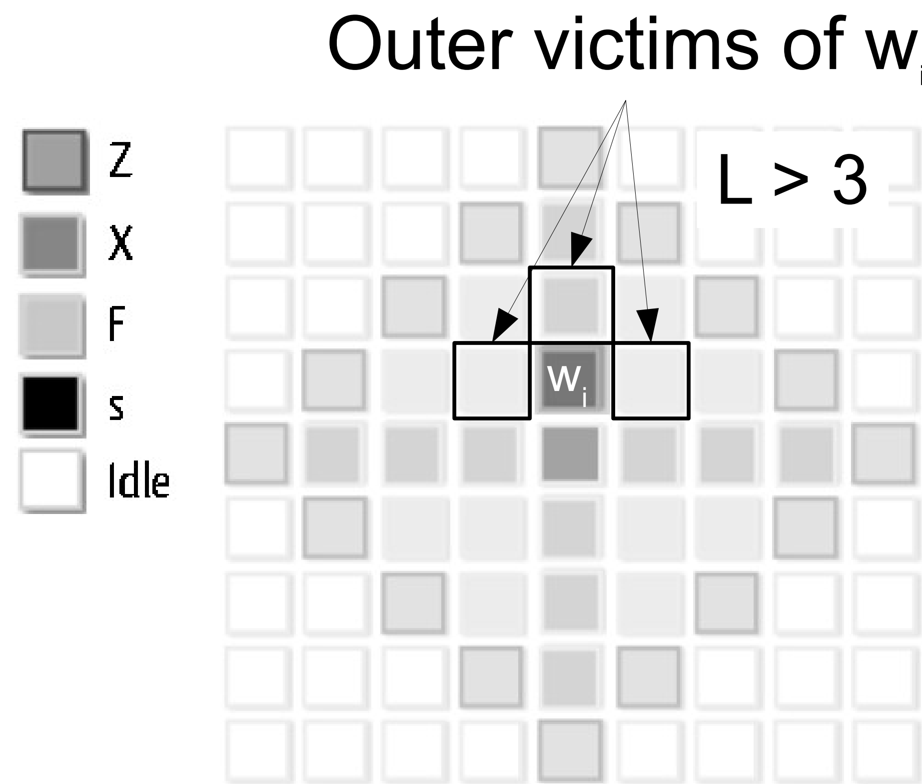
Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims



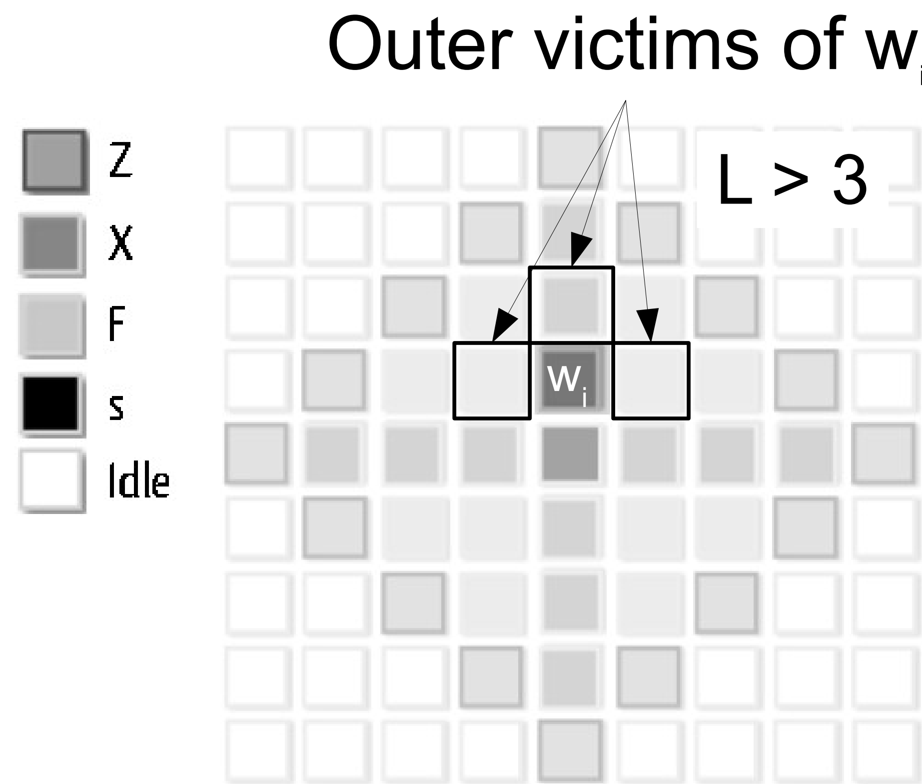
Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims



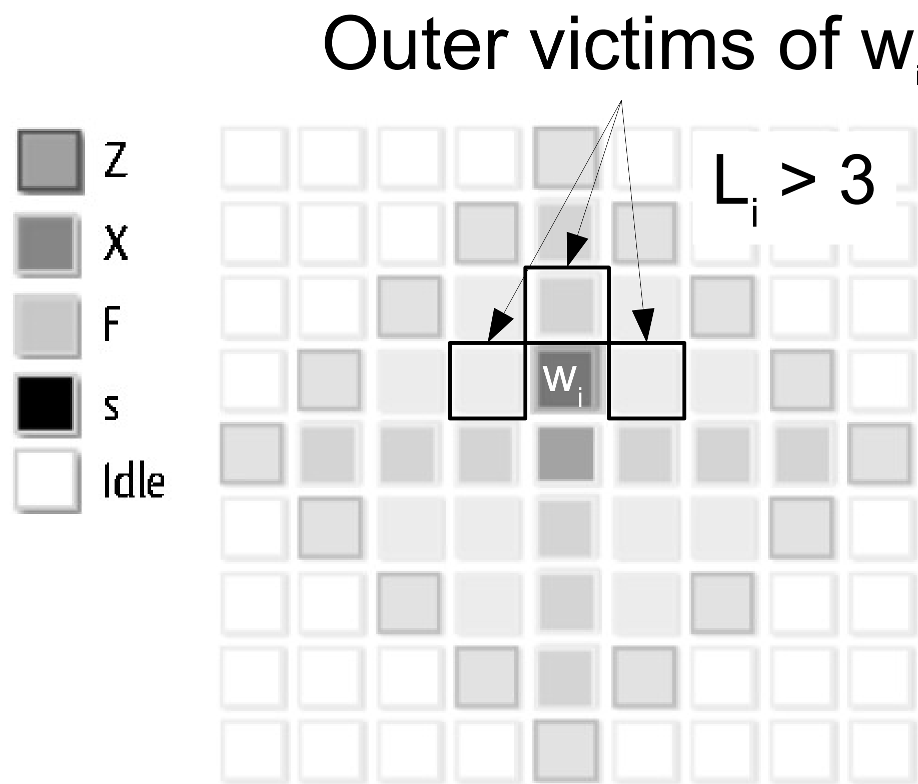
Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims
- O_i: workers that have w_i as their primary victim



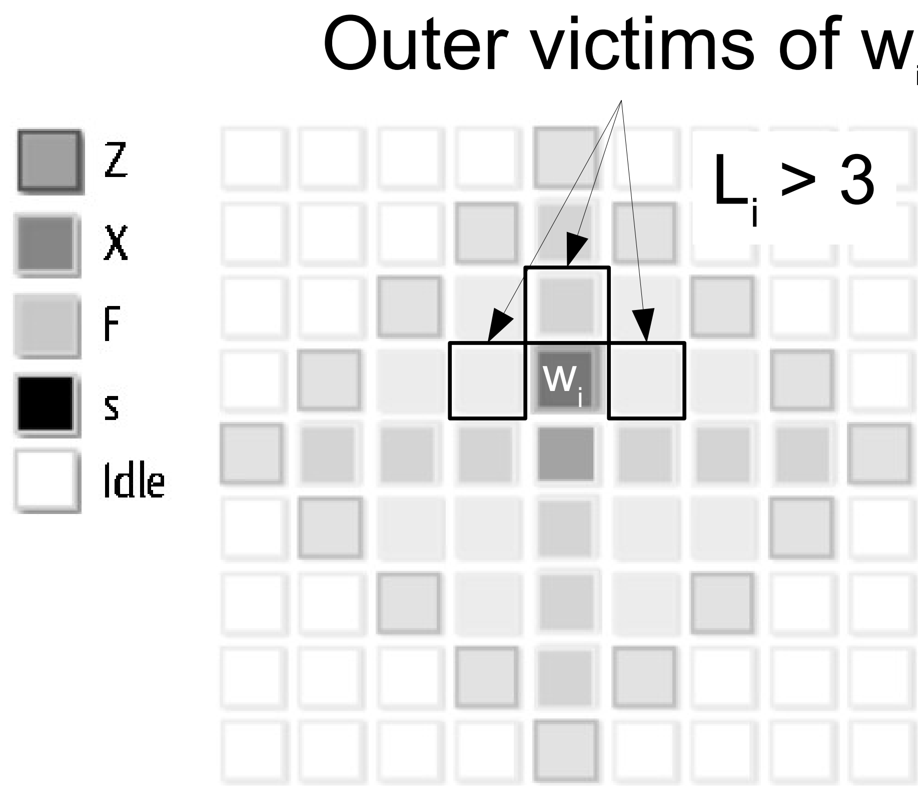
Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims
- O_i: workers that have w_i as their primary victim
- L tunes tolerance



Palirria: Over-utilization condition

- $L_i > |O_i|$
  - $|O_i|$: Number of Outer victims
- O_i: workers that have w_i as their primary victim
- $L = |O_i| + 1$
- L is calculated when constructing the victim-set

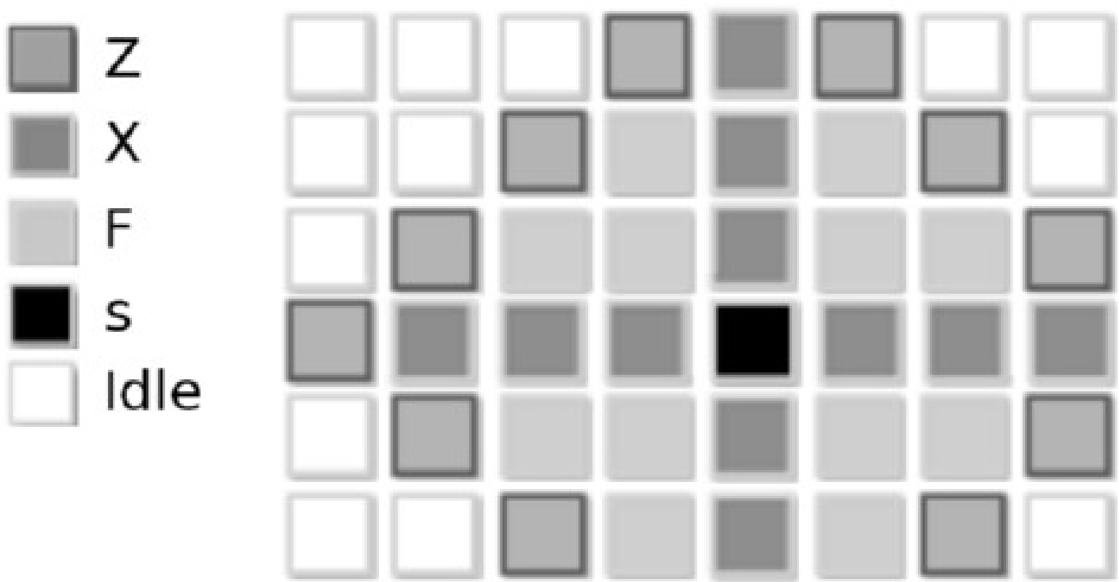


ASTEAL: prominent related work

- **Metric**: cycles spent on wasteful actions
  - Failed steal attempts
- Samples the cycle counter of all workers

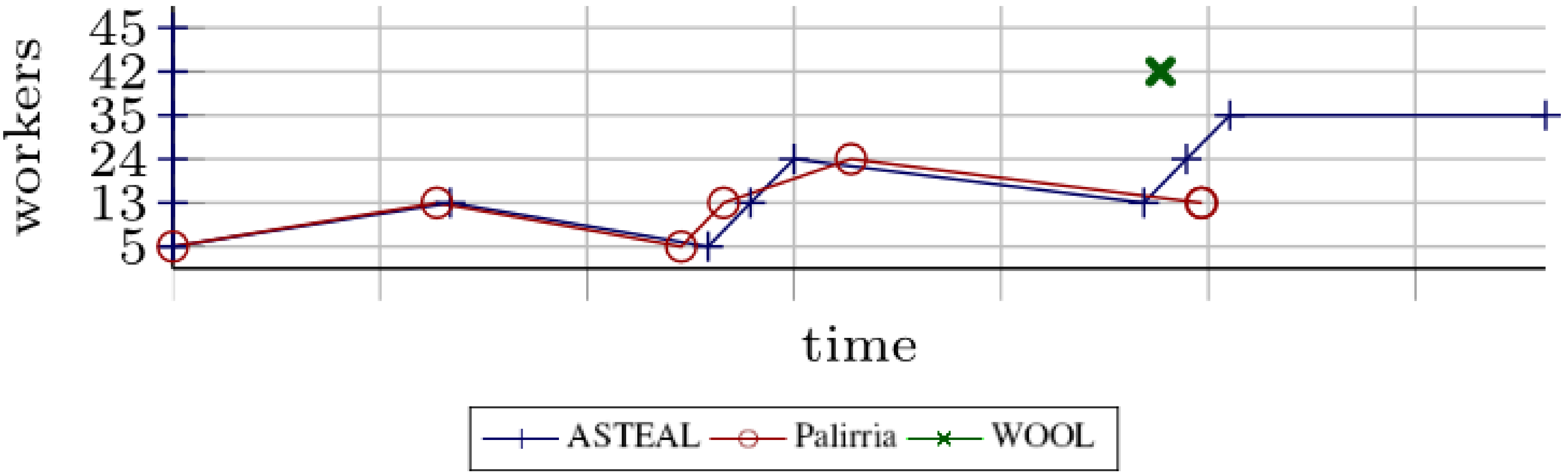
Palirria Evaluation

- All implementations using the same WOOL scheduler
- Linux on a 48-core Opteron Numa system

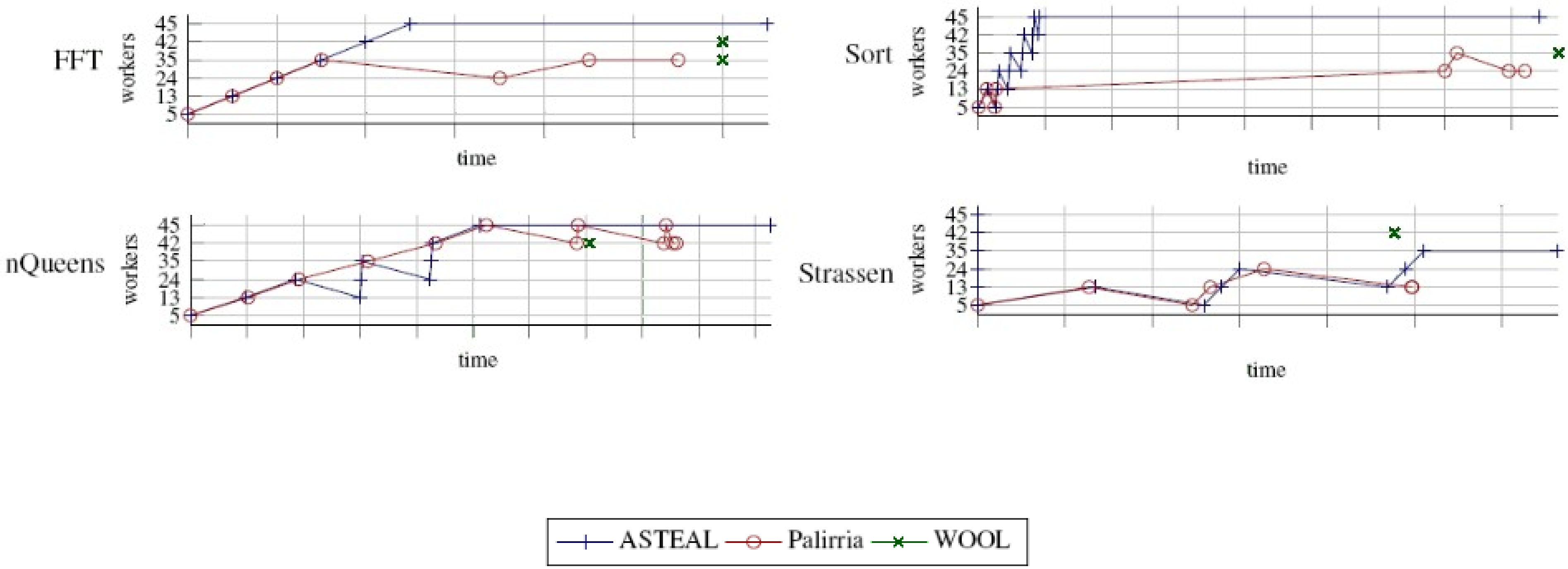


Accuracy

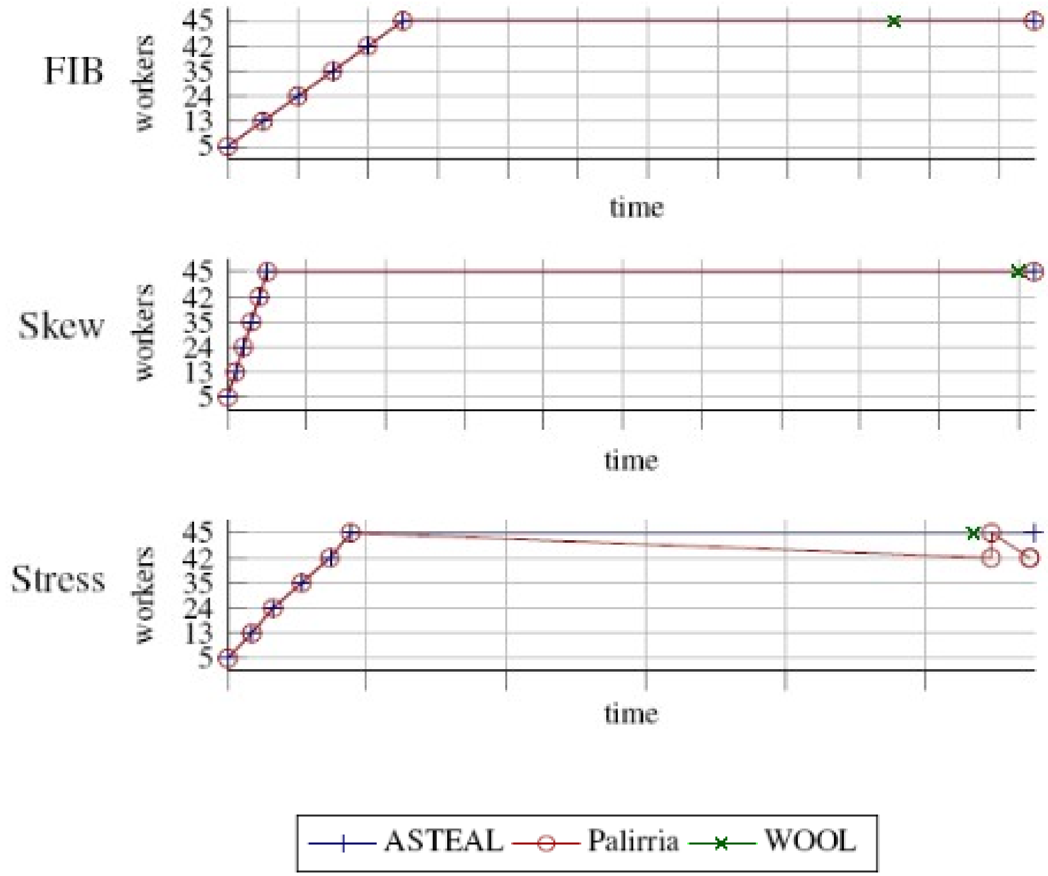
- Dynamically changed allotment size over time
- WOOL: best fixed size execution time



Accuracy: irregular workloads



Accuracy: regular workloads



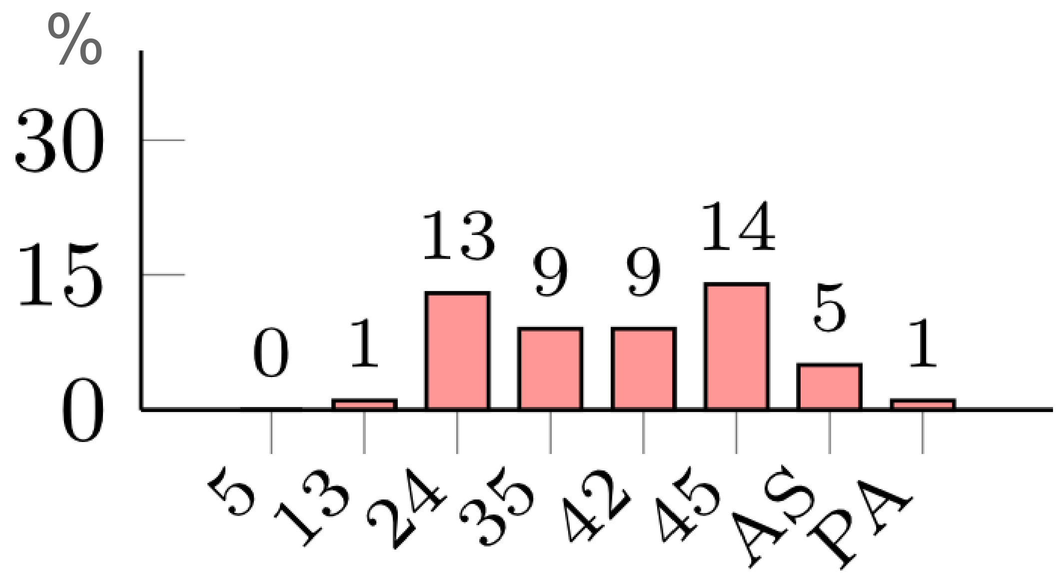
Wastefulness

- Percentage of the avg per worker execution time spent:
  - idling
  - on failed steal attempts

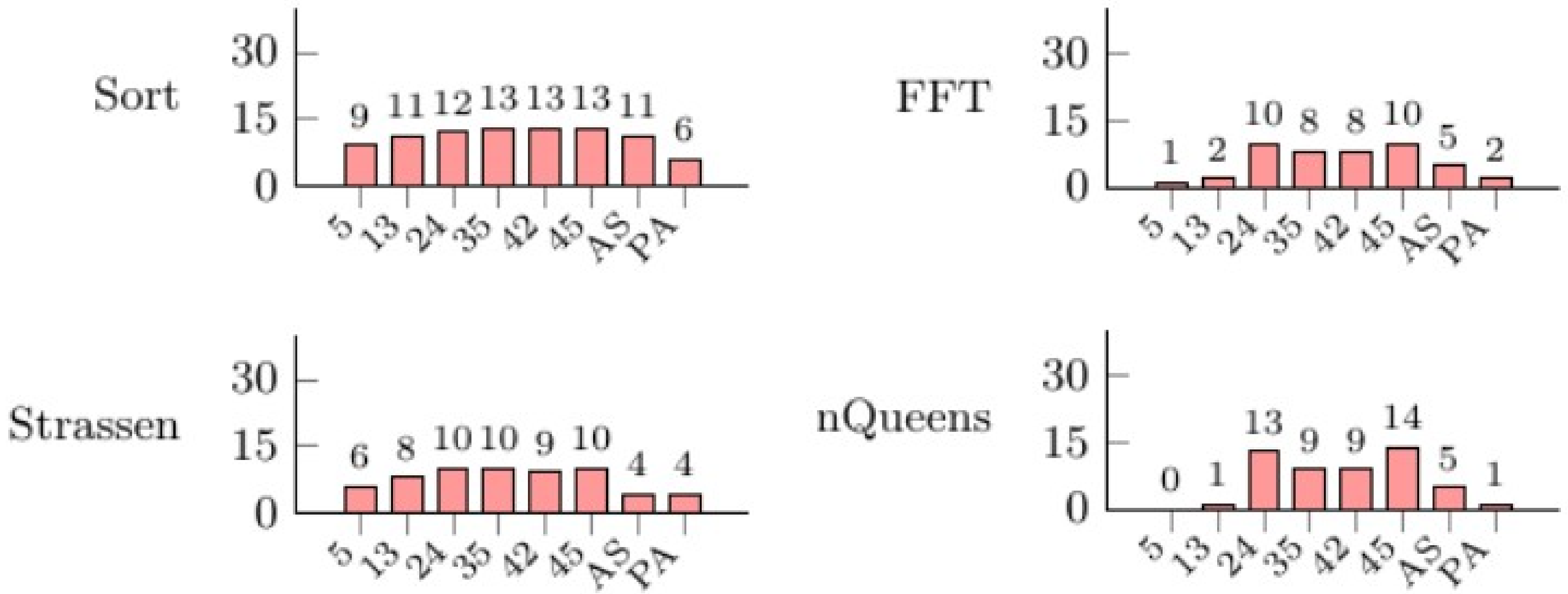
n: fixed n-workers

AS: Asteal adaptive

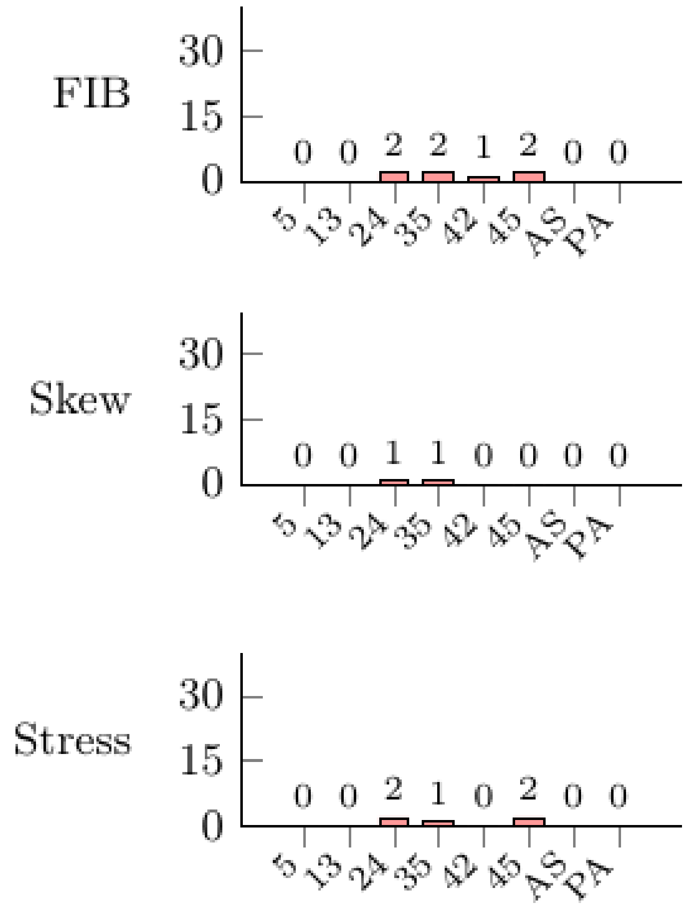
PA: Palirria adaptive



Wastefulness: irregular workloads



Wastefulness: regular workloads



Conclusions

- Non-random workload distribution techniques
  - Are efficient
  - Enable accurate estimation of parallelism
- Task-queue size
  - Quantifies future parallelism
  - Is hardware agnostic

Summary

- Palirria
  - Method for estimating a task-based workload's concurrency
    - Accurate, lightweight, online, no training
  - Built upon a variation to traditional work-stealing
    - Deterministic Victim Selection (**DVS**) replaces victim selection in any work-stealing scheduler
  - ➔ Good performance with less worker threads for workloads of irregular parallelism

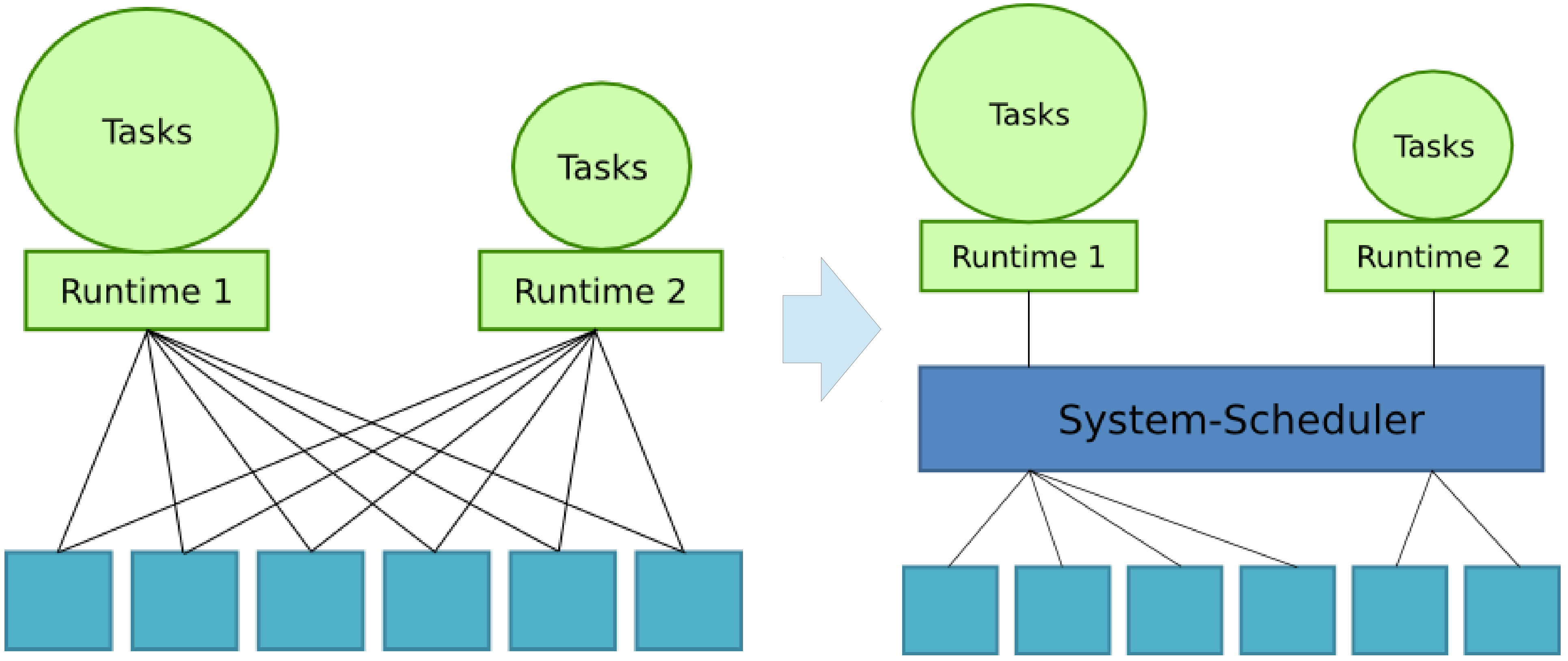
Thank you

Dynamic Resource Allocation

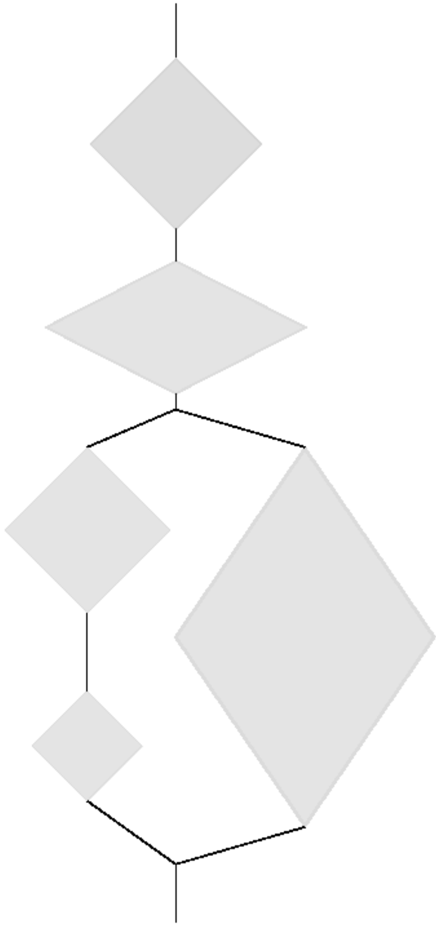
- The operating system knows resource availability

- The application runtime knows resource requirements

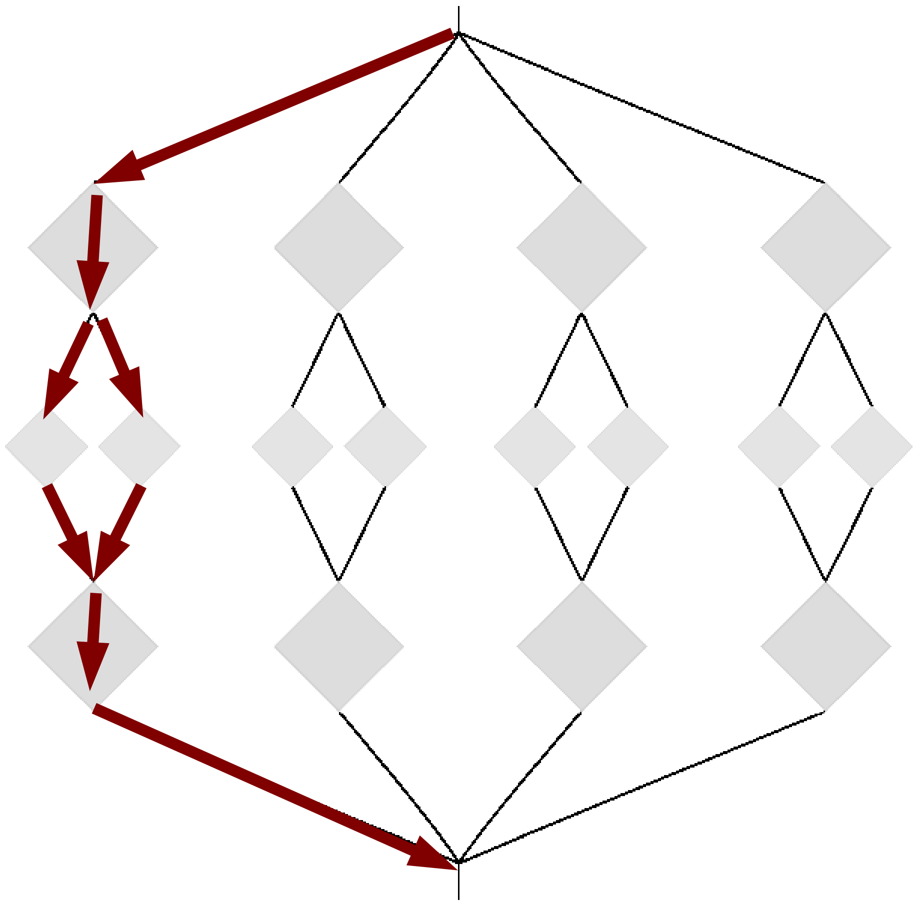
Two Level Scheduling Scheme



Flow of Tasks

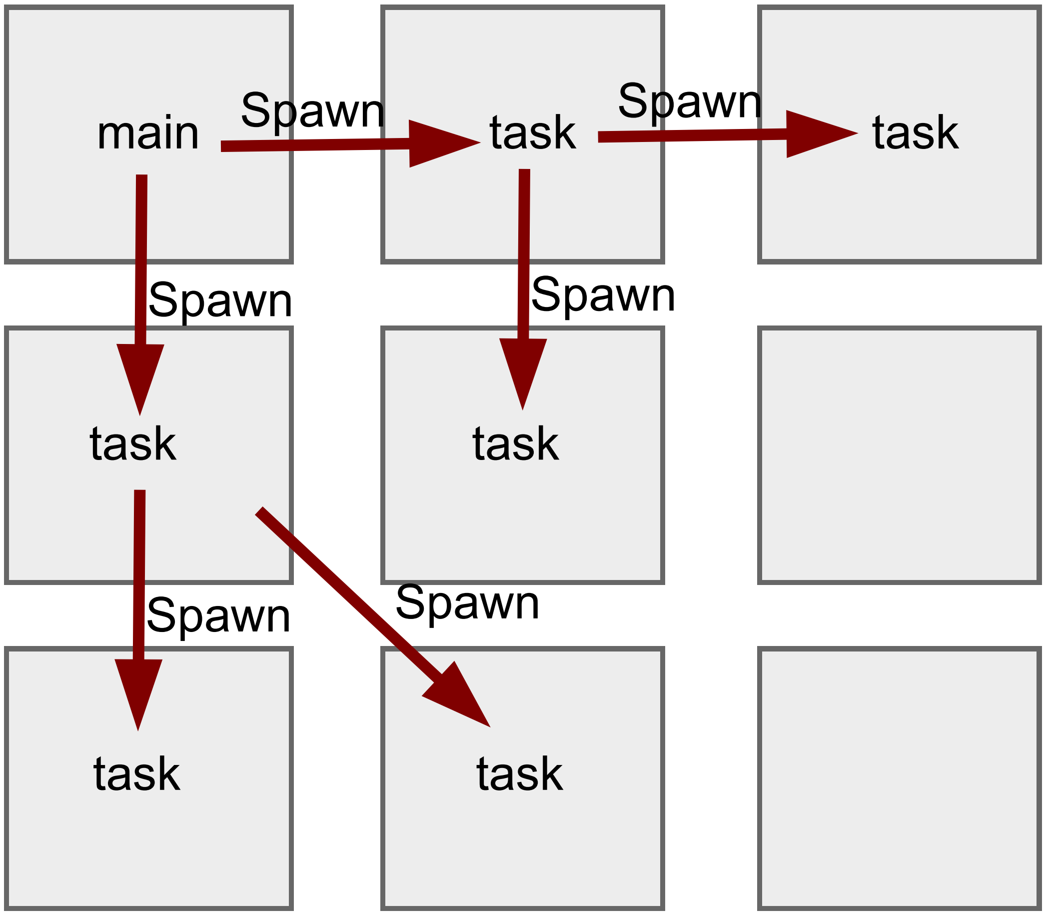


Parallel program
sequence of parallel sections



One parallel section

Flow of Tasks



Task Scheduling Issues

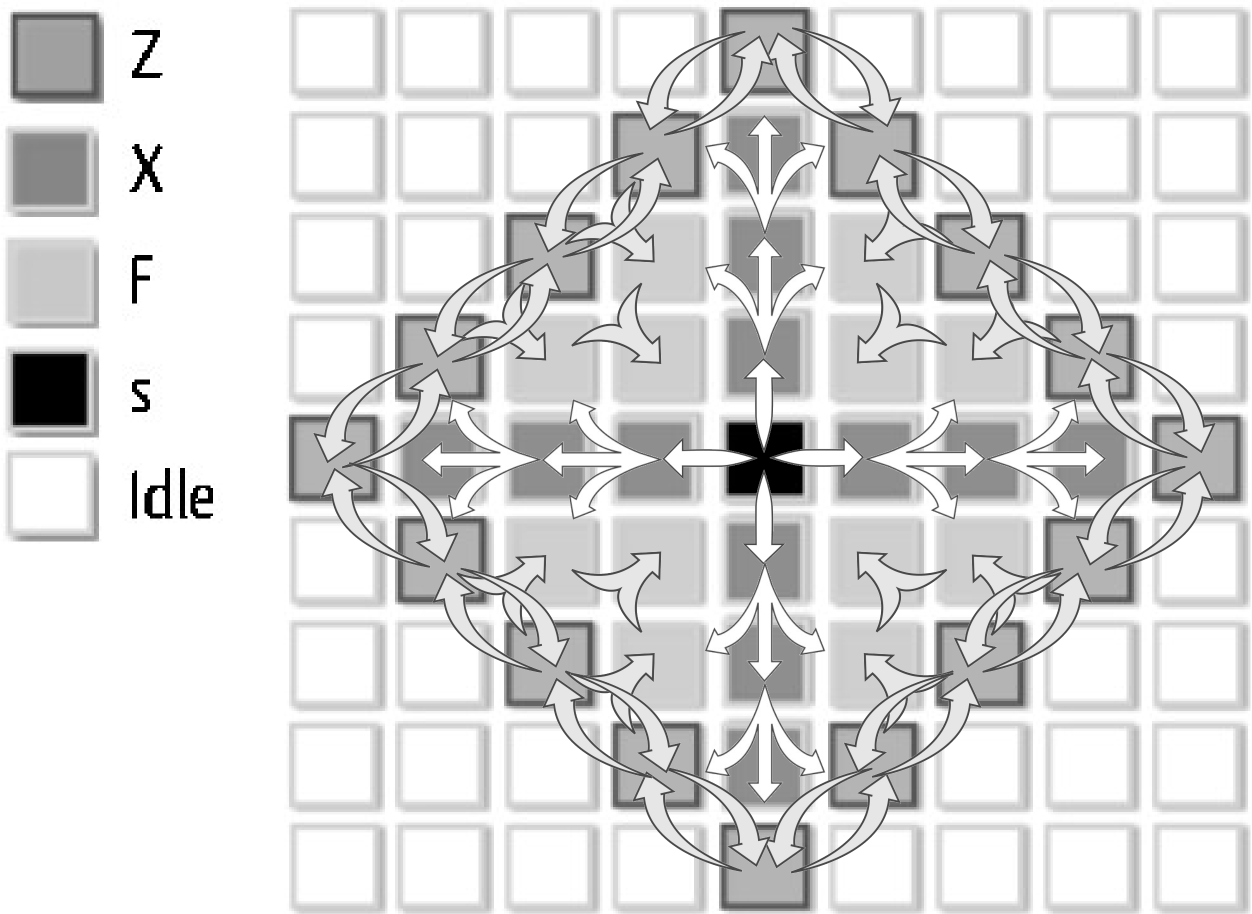
- Adaptation of allotment size
  - Dynamically estimate actual parallelism
  - → Predictable distribution of tasks

- Uniform distribution
  - Available tasks equally distributed
  - → Controllable distribution of tasks

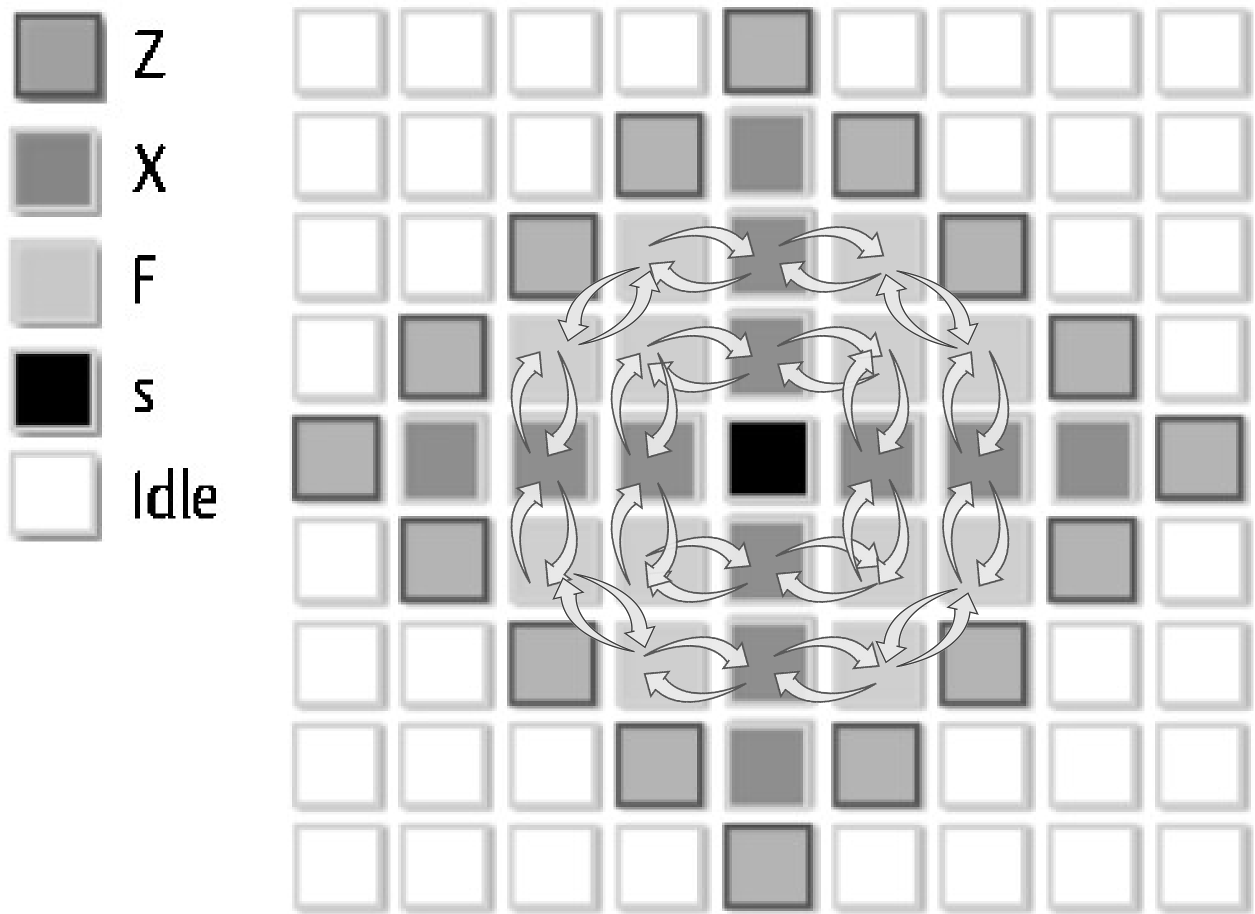
Work-stealing

- Victim selection
  - Random
    - Uncontrollable distribution
  - Semi-random (leap-frogging)
    - Unpredictable distribution
  - Non-random?
    - Controllable and predictable distribution
    - Can it be as fast?

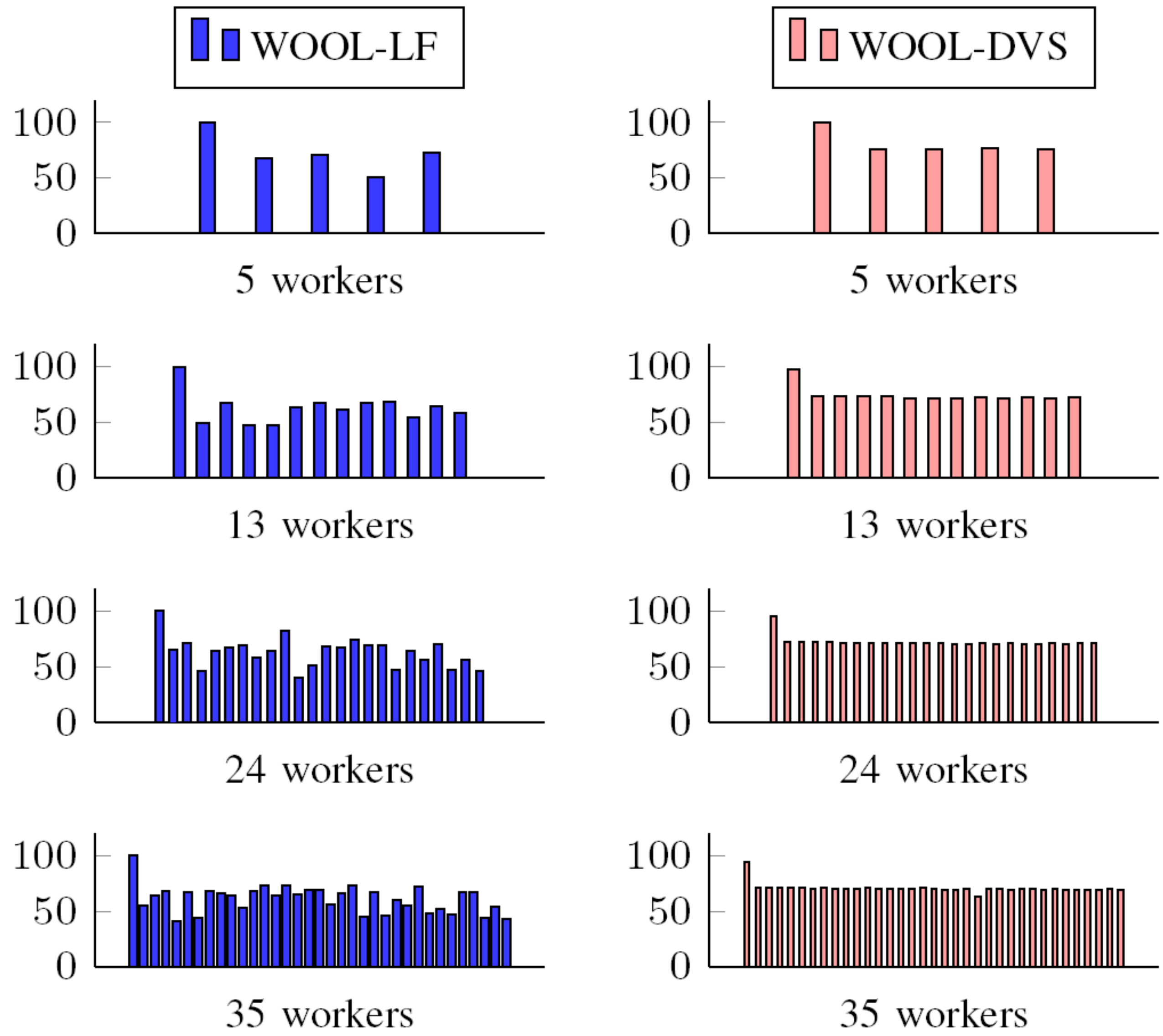
DVS: Deterministic Victim Selection



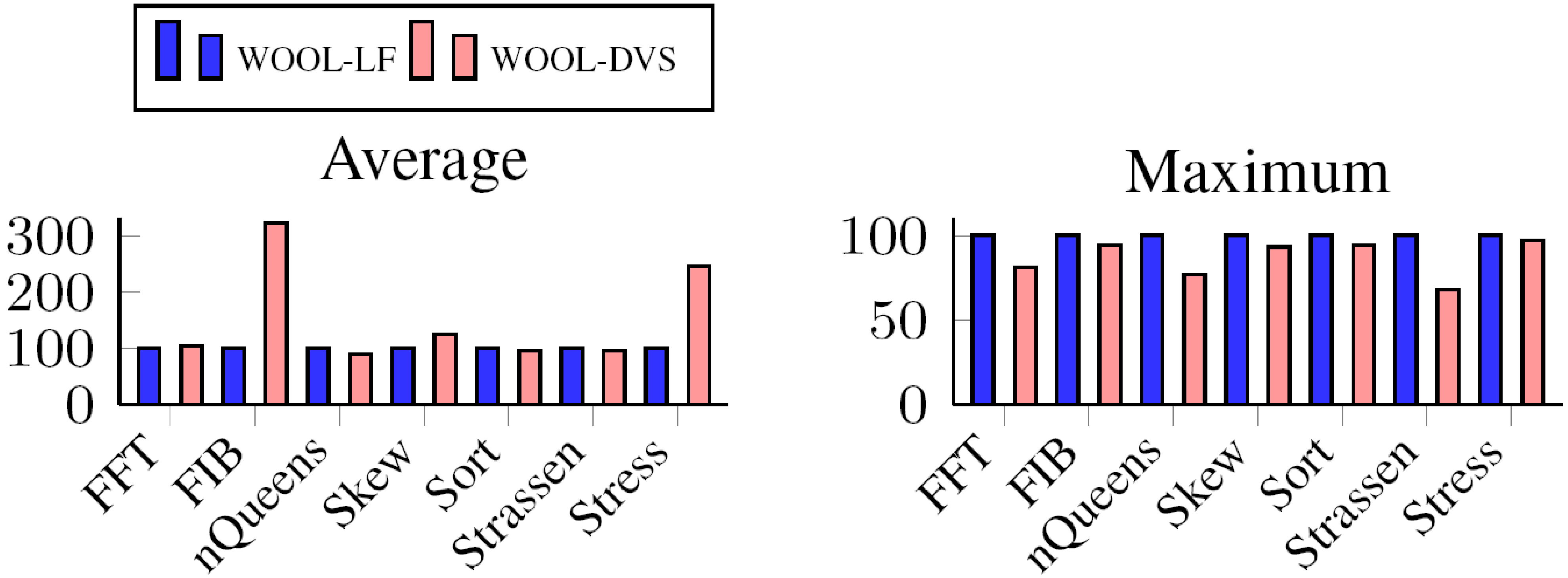
DVS: Deterministic Victim Selection



DVS: Workers' Useful Time



DVS: First successful steal latency



DVS: Execution time

DVS: Execution time