

Latency Hiding, Work Stealing Scheduling

Vladimir M.

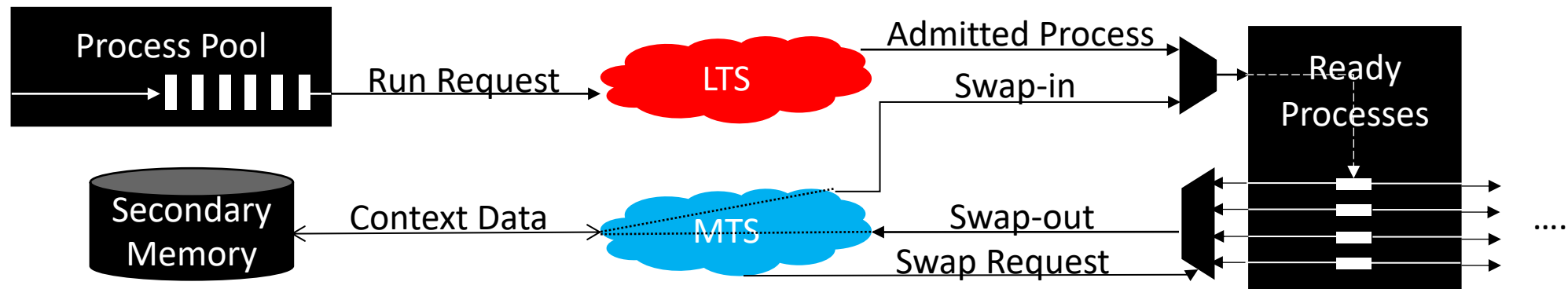
What is Scheduling

- Overall: **map** resources to tasks, service **multiple** tasks, make **efficient** use of resources.
- Throughput \uparrow , turnaround time \downarrow , response time \downarrow , resource utilization \uparrow , wait time \downarrow , fairness.
- Preemptive (scheduler may force context switch) and,
- Cooperative (non-preemptive).

Scheduling = deciding which tasks should make progress...

System Level: Long/Medium-Term

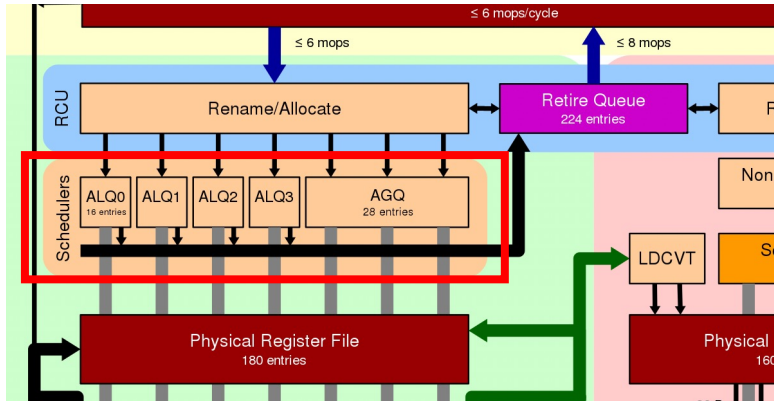
- **Long-term scheduling**: choose processes to admit into execution.
 - E.g., a process requests to be run. LTS reviews: concurrent process count, characterization of running/requested processes, process co-scheduling request.



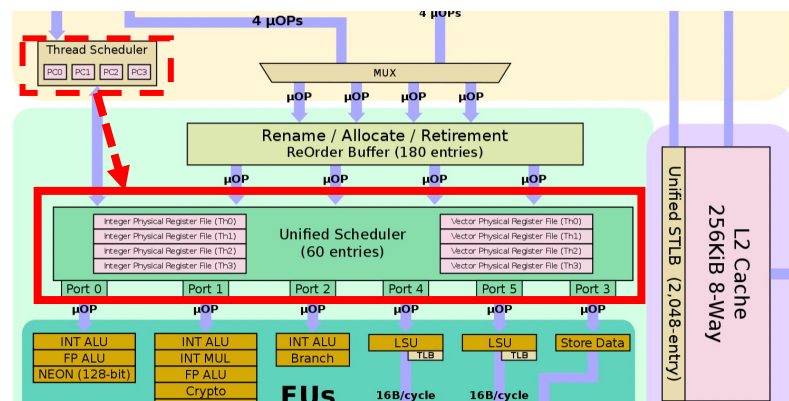
- **Medium-term**: control availability between processes and resources.
 - E.g., a process is waiting very long for IO resp., MTS may swap processes based on idle, priority, memory access patterns (faulting), jumbo memory use.

Thread Level: Short-Term & Dispatching

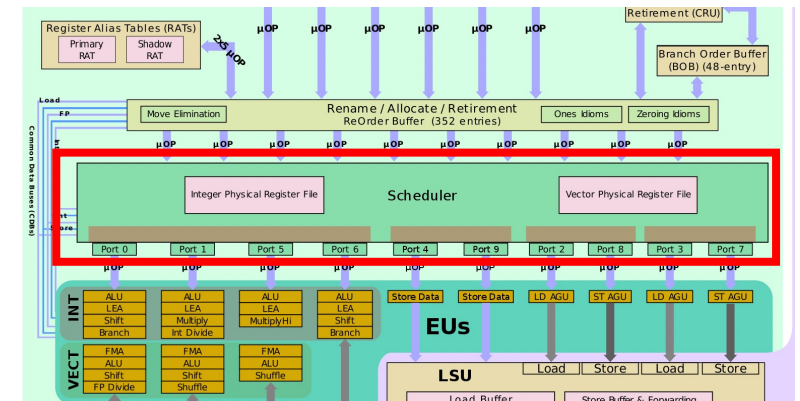
- Short-term scheduling: choosing ready processes to map onto HW resources.
 - Signaled by the host OS, or after task completion (internal HW).
 - Processor affinity (context) & process ganging (comm.) optimizations.
 - What the selected paper focuses on.
- Process Dispatch
 - HW support for load/store of process descriptor, registers.



AMD Zen2 (x86-64) Core Microarchitecture
2-thread FCFS Scheduler (fine-grained simultaneous multithreading)





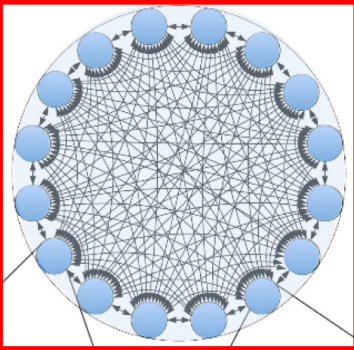
Cavium Vulcan (ARM-v8) Core Microarchitecture
4-threads (fine-grained simultaneous multithreading)

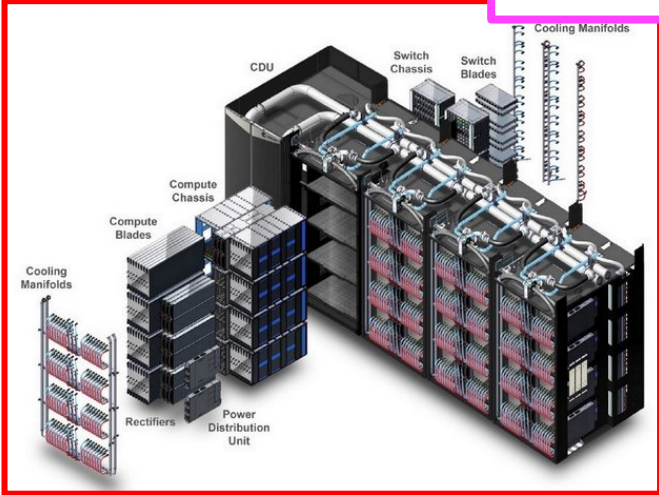
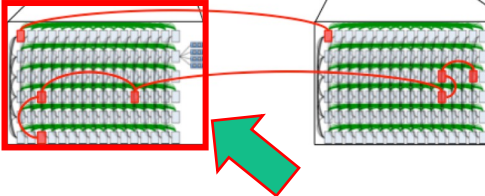


Intel Sunny Cove (x86-64) Core Microarchitecture
2-thread FCFS Scheduler (fine-grain temporal multithreading)

Aside: Supercomputer Scheduling Problem


HPE Cray: Aries Interconnect





5	Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
---	--	---------	----------	----------	-------

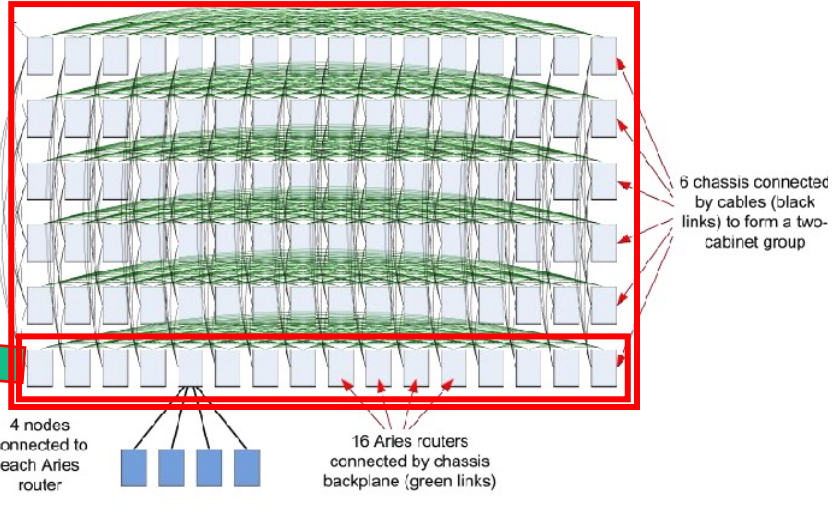
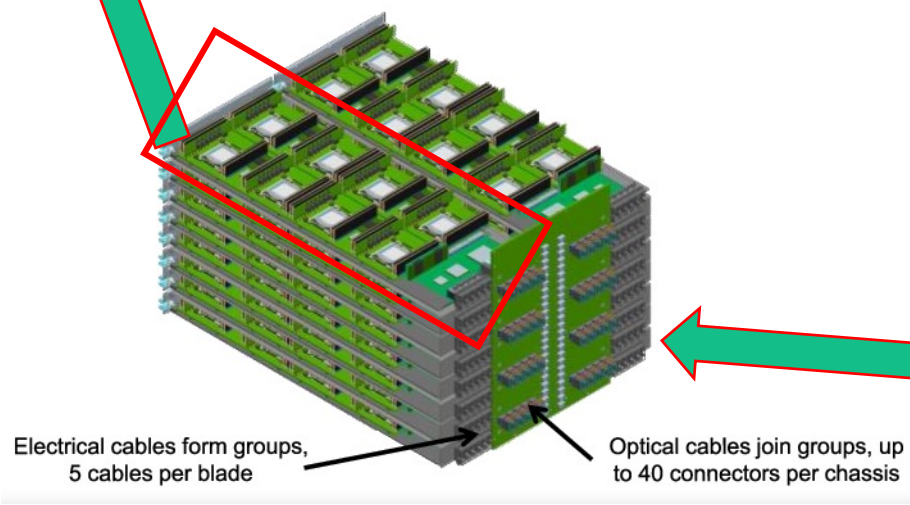
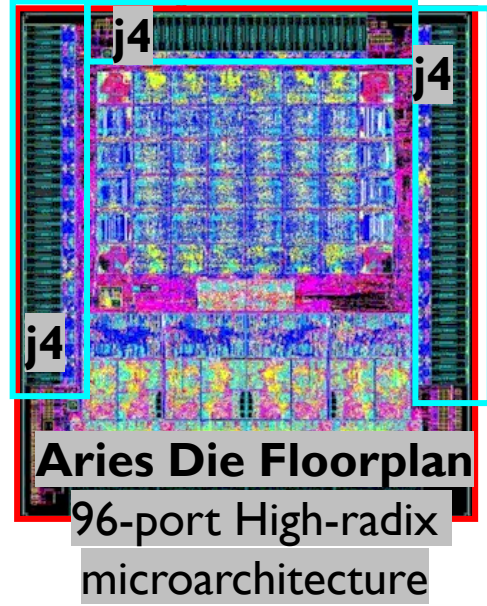
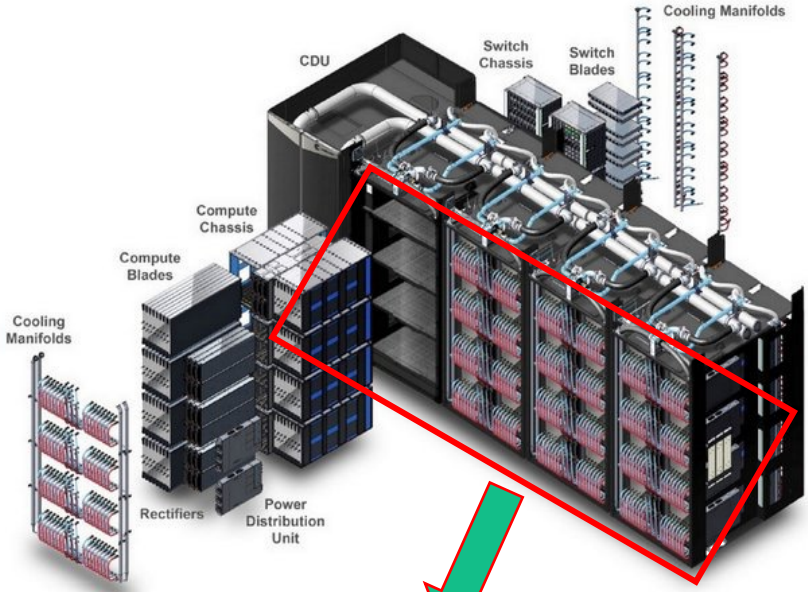
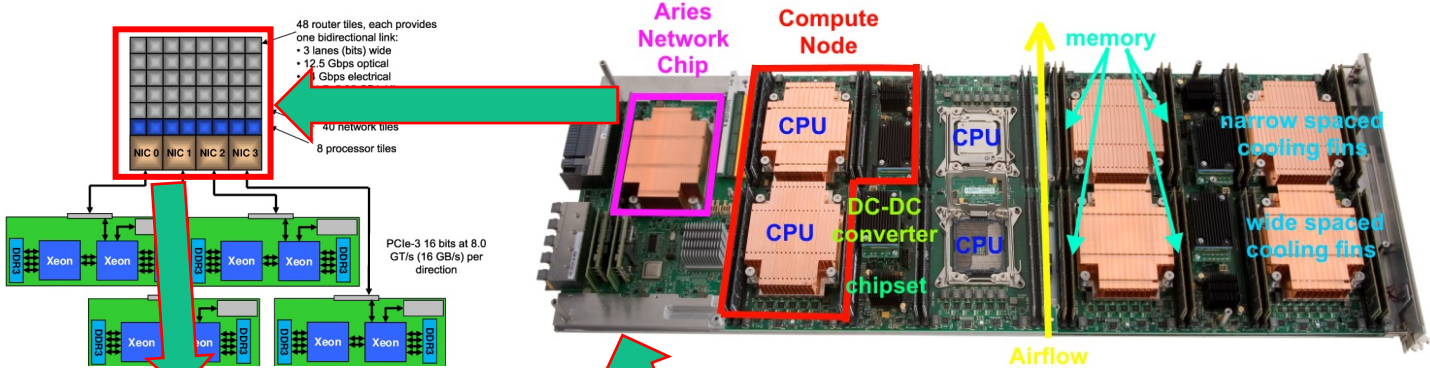
>750K cores where



Aside: Supercomputer Scheduling Problem

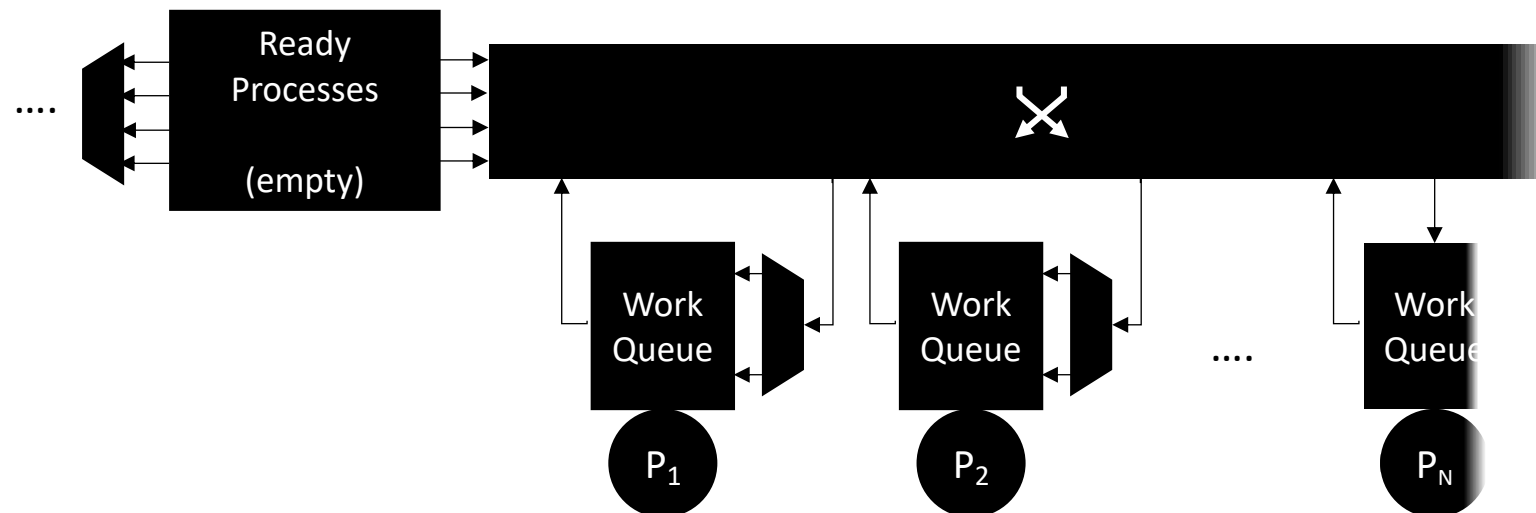
HPE Cray: Aries Interconnect

AMD EPYC (Zen 2 x86-64) [64C/CPU] * 8 [CPU/ Blade] * 6 [Blade/Chassis]
* 6 [Chassis/Cabinet] * 42 [Cabinet] = 774K cores



Work Stealing and Latency Hiding Techniques

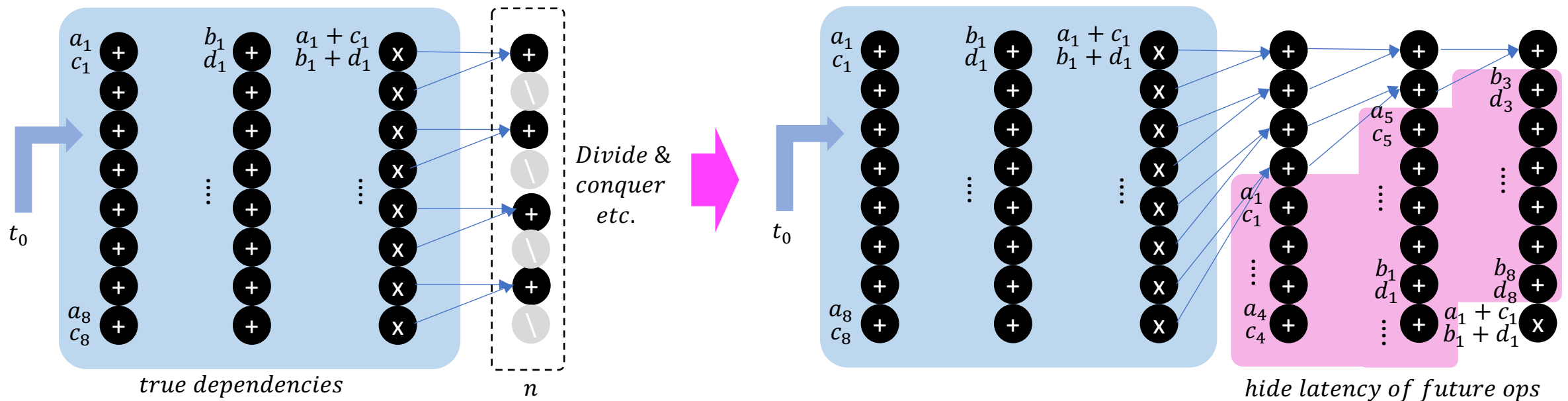
- **Work Stealing:** underutilized resources attempt to “steal” processes (work) from other resources.
- Not *Work Sharing*: migration of spawned processes across resources.
- Allows resources to take the initiative of “stealing” work from other resources.



Work Stealing and Latency Hiding Techniques

- Latency: time requirement to perform an action (i.e., an operation).
 - Stalls dependent instructions, could mask latency by overlapping other concurrent operations...
- **Latency Hiding:** perform concurrent operation(s) while waiting for latency operation.
 - make progress on overall work while waiting.

E.g., biased MAC: $f(x) = \sum_{i=1}^n (a_i + c_i)(b_i + d_i) \mid n \text{ processors}$

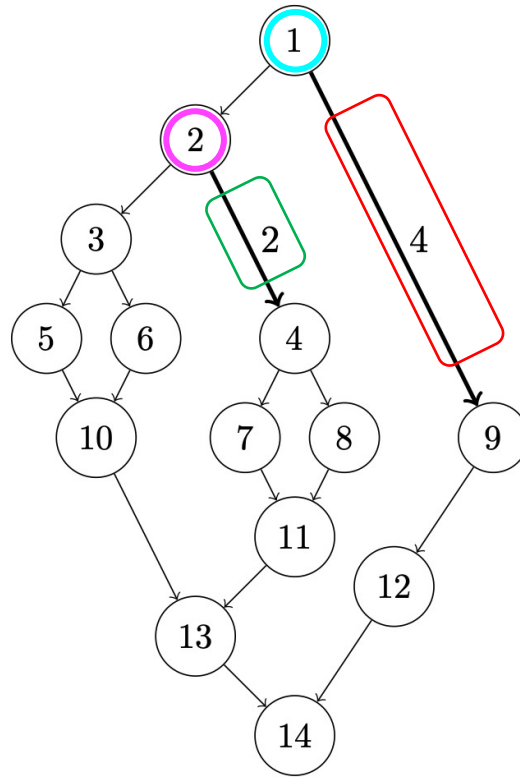
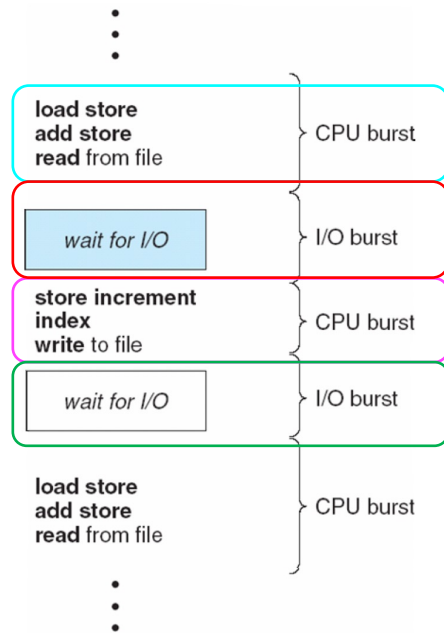


Paper: Introduction

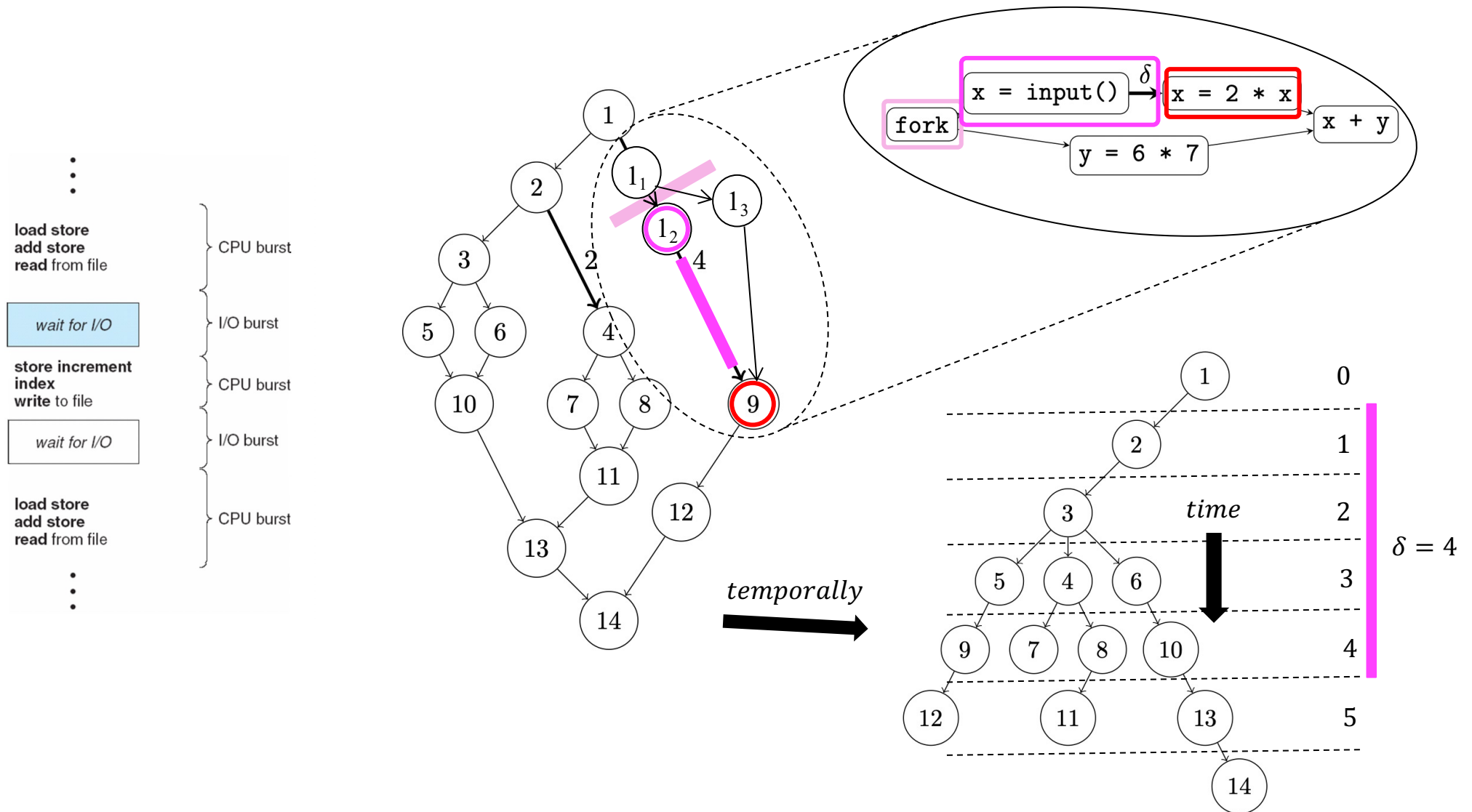
*“Latency Hiding Work Scheduling:
Scheduling Interacting Parallel Computations with Work Stealing”
Stefan K. Muller, Umut A. Acar (SPAA '17)*

- Extends the Directed Acyclic Graph (DAG) model for parallelism to account for latency, present new work-stealing algorithm for hiding latency.
- Allows threads to suspend without blocking the underlying resource, allowing the scheduler to switch to another thread.
- For workloads which access external networks, secondary storage, remote procedure calls etc., significant performance improvement.
- Introduces multiple deques per-worker, scheduler coalescing of resuming tasks, execution of resumed tasks parallel, thieves' ability to target individual deques – not individual workers.

Paper: Model of Computation



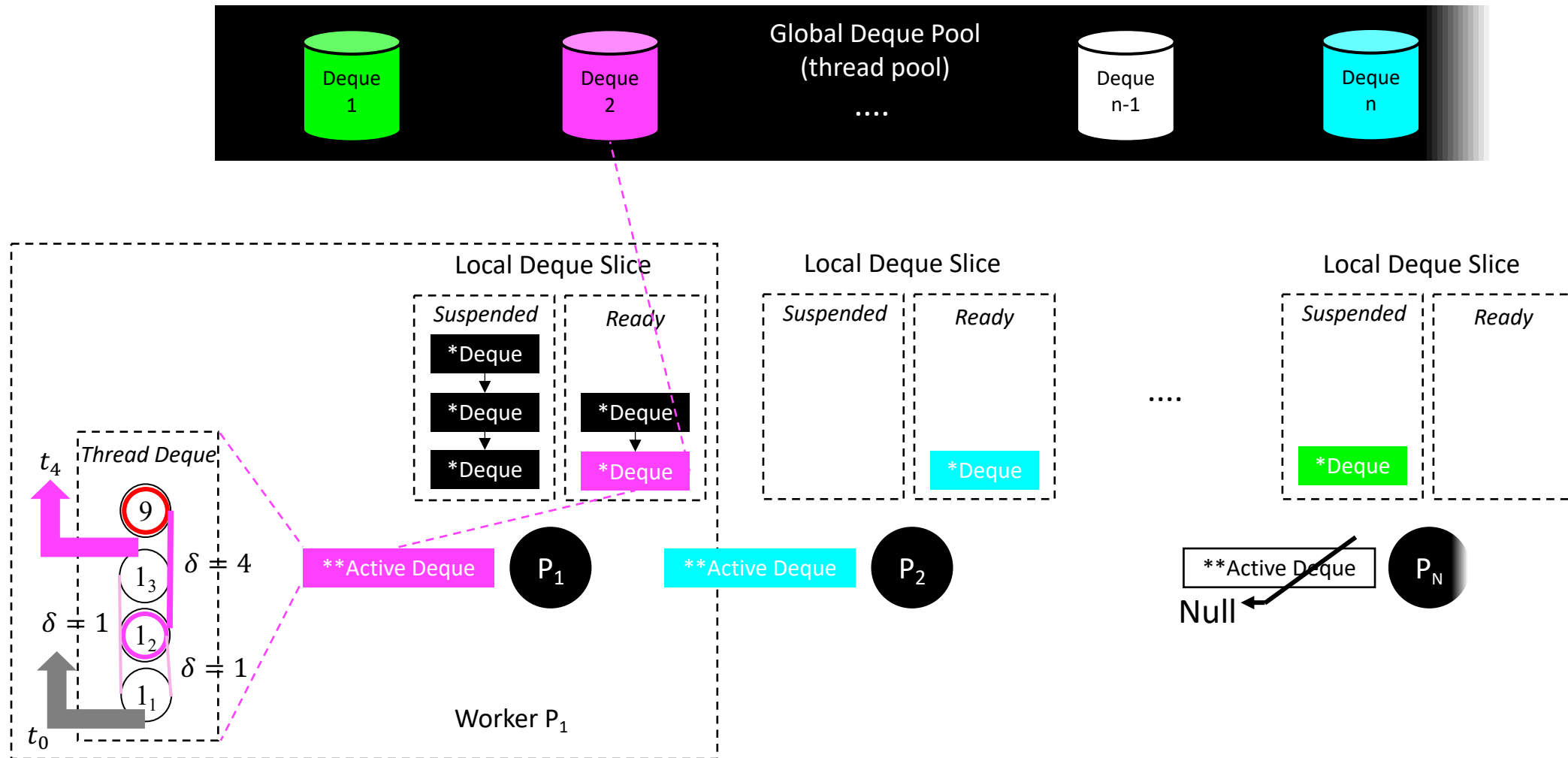
Paper: Model of Computation



Paper: Scheduling Problem

- Schedule the dag as its structure is revealed during the execution.
- Operating in rounds, execute an instruction and learn what vertices become enabled and possible ready as a result.
- Executing an instruction may enable the children of the instruction, but not all children may be immediately ready ($\delta > 1$).
- We do not know how the weight of a heavy edge (i.e., only evaluate $\delta > 1$).
- The latency hiding work stealing scheduler works in parallel to solve the online scheduling problem of weighted dag computation.

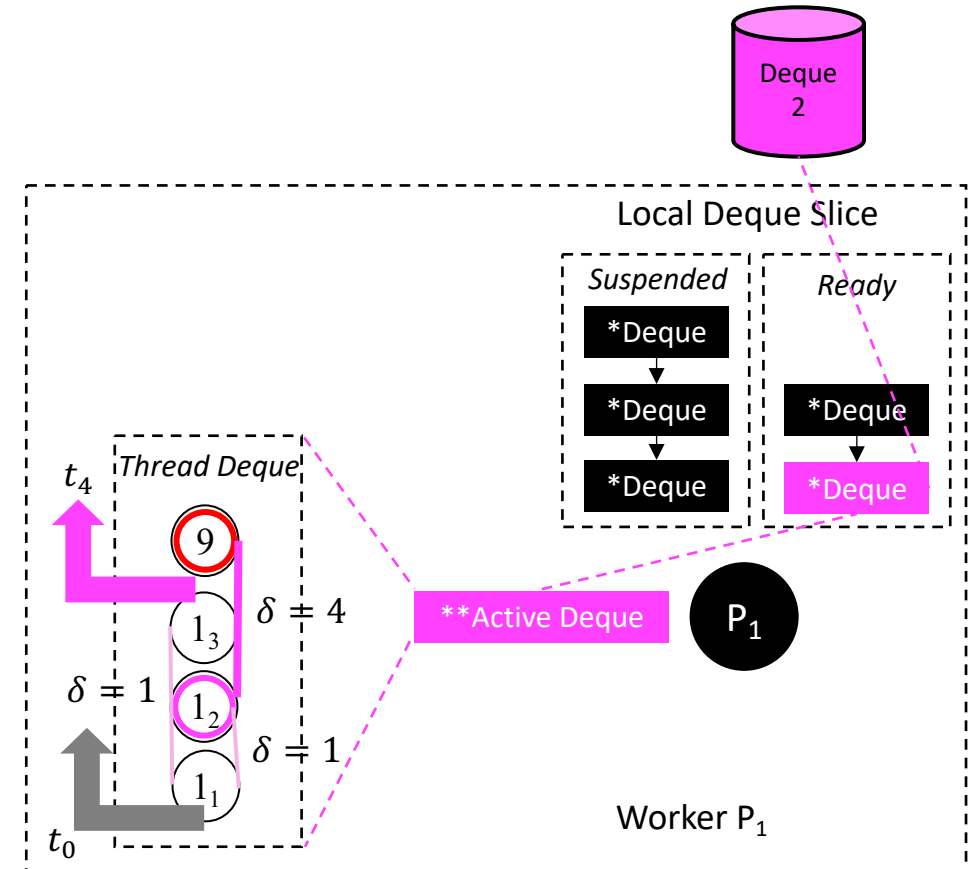
LHWS Scheduler: Structure



LHWS Scheduler: Algorithm

- Deques (double-ended queues) used to store work.
- Workers may have multiple deques, one of which is active.

1. Worker tries to pop vertex out of deque.
 1. IF successful, execute vertex + push children into deque
 2. ELSE suspended vertex, continue looking for work.
 1. IF work found, continue executing vertices.
 2. ELSE check owned deques for work
 1. IF work found, switch active Deque, continue executing.
 2. ELSE attempt to steal vertex from another Deque
 1. IF successful, start new Deque with vertex.
 2. ELSE idle.
2. Scheduler initializes computation, then
 1. When suspended vertices resume, partitions them by deque and executes them in parallel.
3. Suspended vertices install a callback that signals resumption to the scheduler.

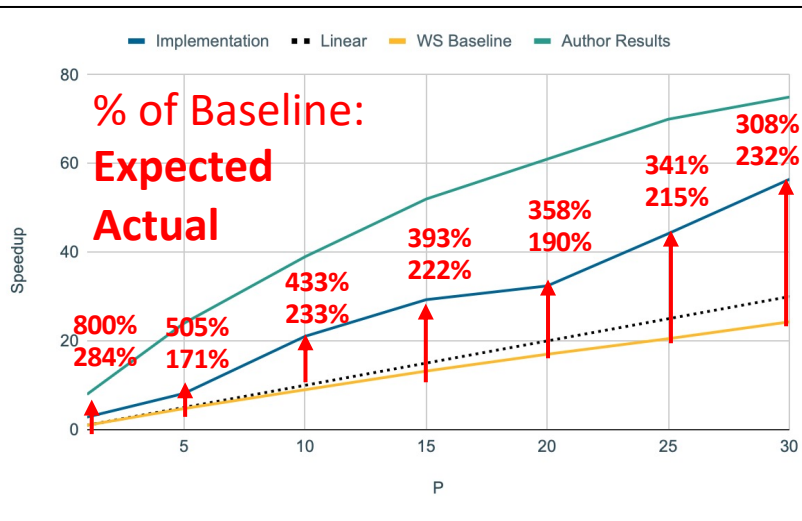
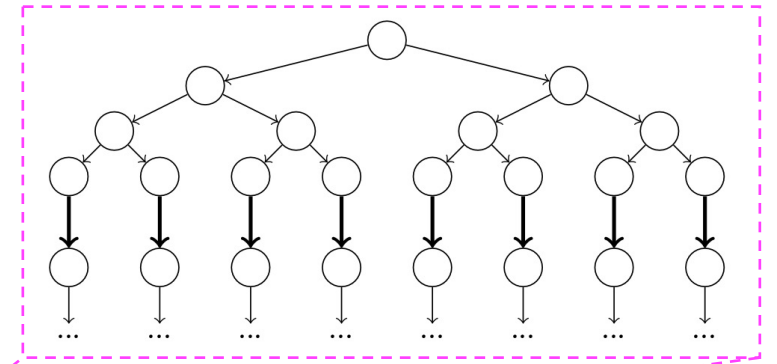


LHWS Scheduler: Analysis

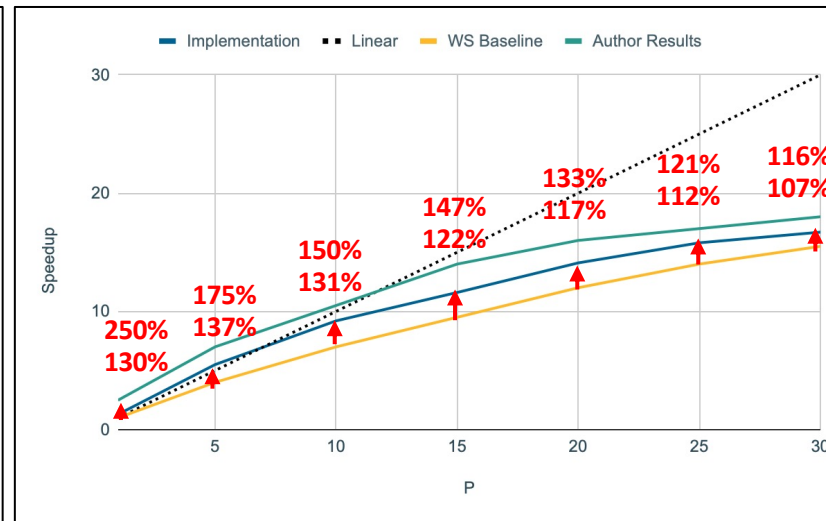
- Scheduling takes (amortized) time $O(1)$
- Worker runtime bound: $O\left(\frac{W}{P} + SU(1 + \log U)\right)$
- Where:
 - W = total work
 - P = worker count
 - S = span
 - U = suspension width
- Notice $U = 0$ results in $O\left(\frac{W}{P}\right)$ rounds of computation.

LHWS Scheduler: Results

- Map Reduce workload with $n=5000$ (leaves).
- Below $< 20\%$ of the dag.



$\delta = 50ms$



$\delta = 5ms$



$\delta = 1ms$

Questions

1. Why does exploiting latency hiding improve overall performance?
2. Why store work in dequeues (double-ended queues)?
3. Why are workers able to begin a chain-reaction performing work on the computation dag – without the explicit use of the scheduler?

