Unleashing dynamic task scheduling at rack-scale

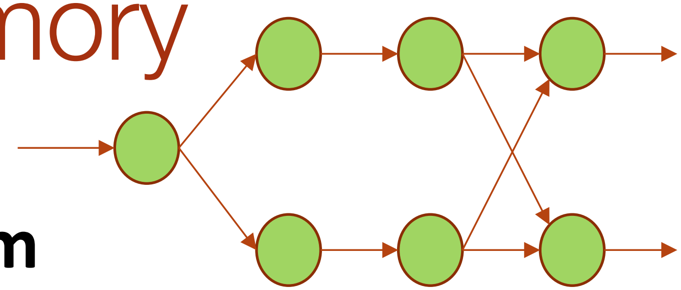
Magnus Norgren,

Andra Hugo (DDN Storage),

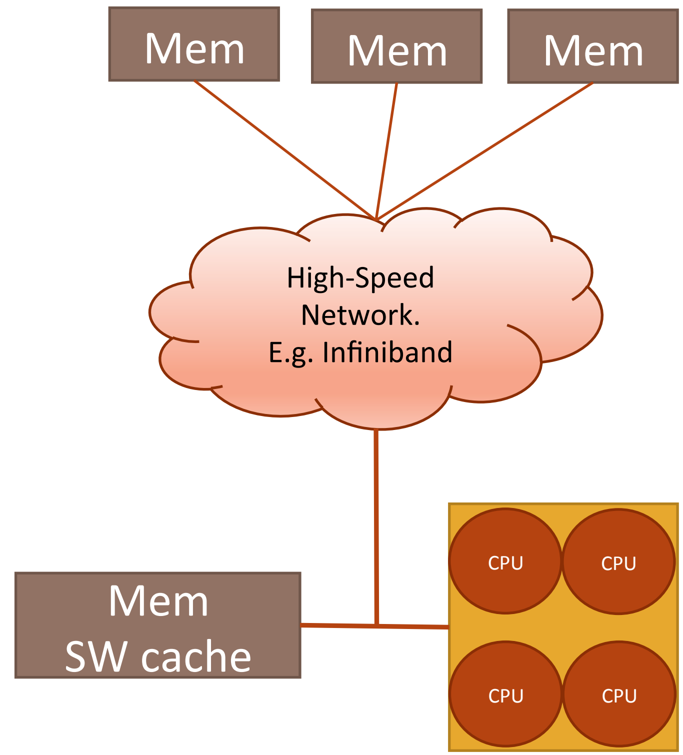
Stefanos Kaxiras, Konstantinos Sagonas

Uppsala University

Task based programming on distributed shared memory



- **Can we scale a task based system at cluster scale?**
- **… on a Software DSM ?!**
  - **→ Yes, If we co-design scheduling & caching**

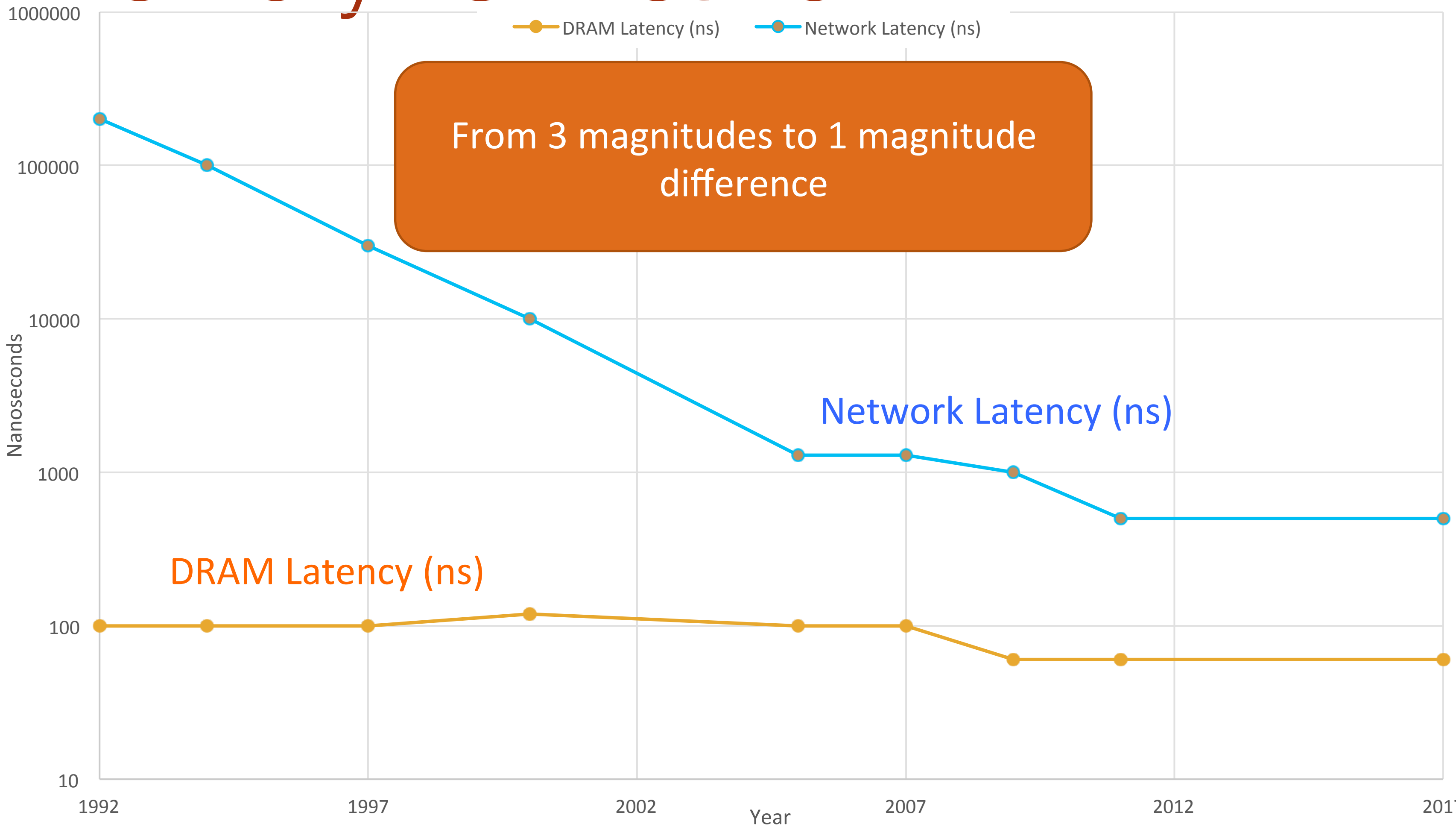


Early results: Running tasks on remote memory at >90% of native execution on local memory

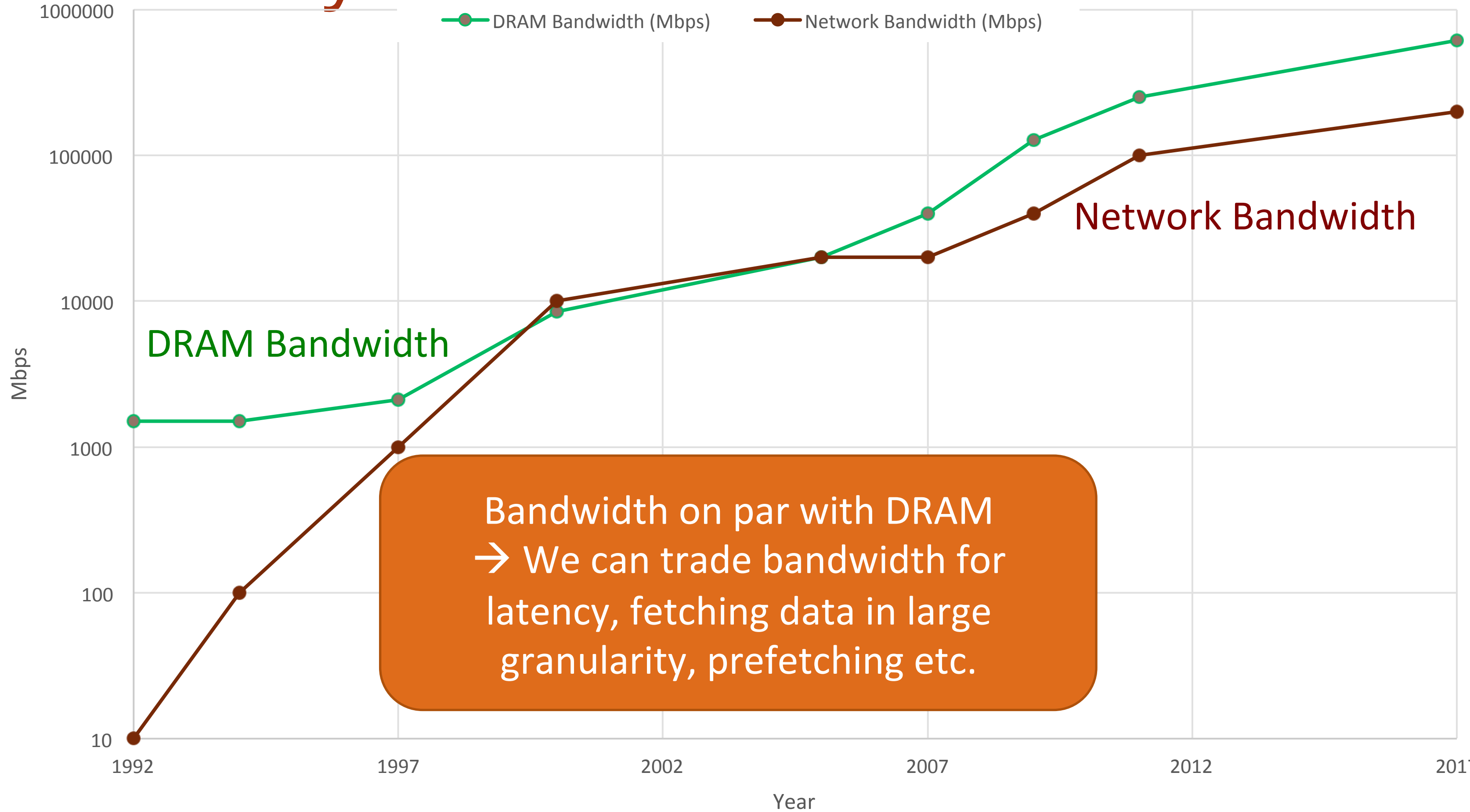
Distributed Shared Memory

- Coherent Distributed Shared Memory (DSM) gives us:
  - Transparent Caching (takes advantage of *any* locality)
    - Even if locality is tiny, its benefit is a function of latency!
    - Avoids hybrid SM (intranode)-MP (internode) programming
  - Freedom from data distribution
    - Demand fetching, prefetching can largely hide bad distribution
  - Natural and familiar synchronization constructs (locks)
  - Extend memory beyond one node
- But isn't this quite costly?
  - Yes – Used to be, but times have changed
    - Latency only one magnitude worse than DRAM
    - …And bandwidth is on par with DRAM

Trends: Latency
Memory vs. Network



Trends: Bandwidth Memory vs. Network



Bandwidth on par with DRAM
→ We can trade bandwidth for latency, fetching data in large granularity, prefetching etc.

ARGO DSM (www.argodsm.com)

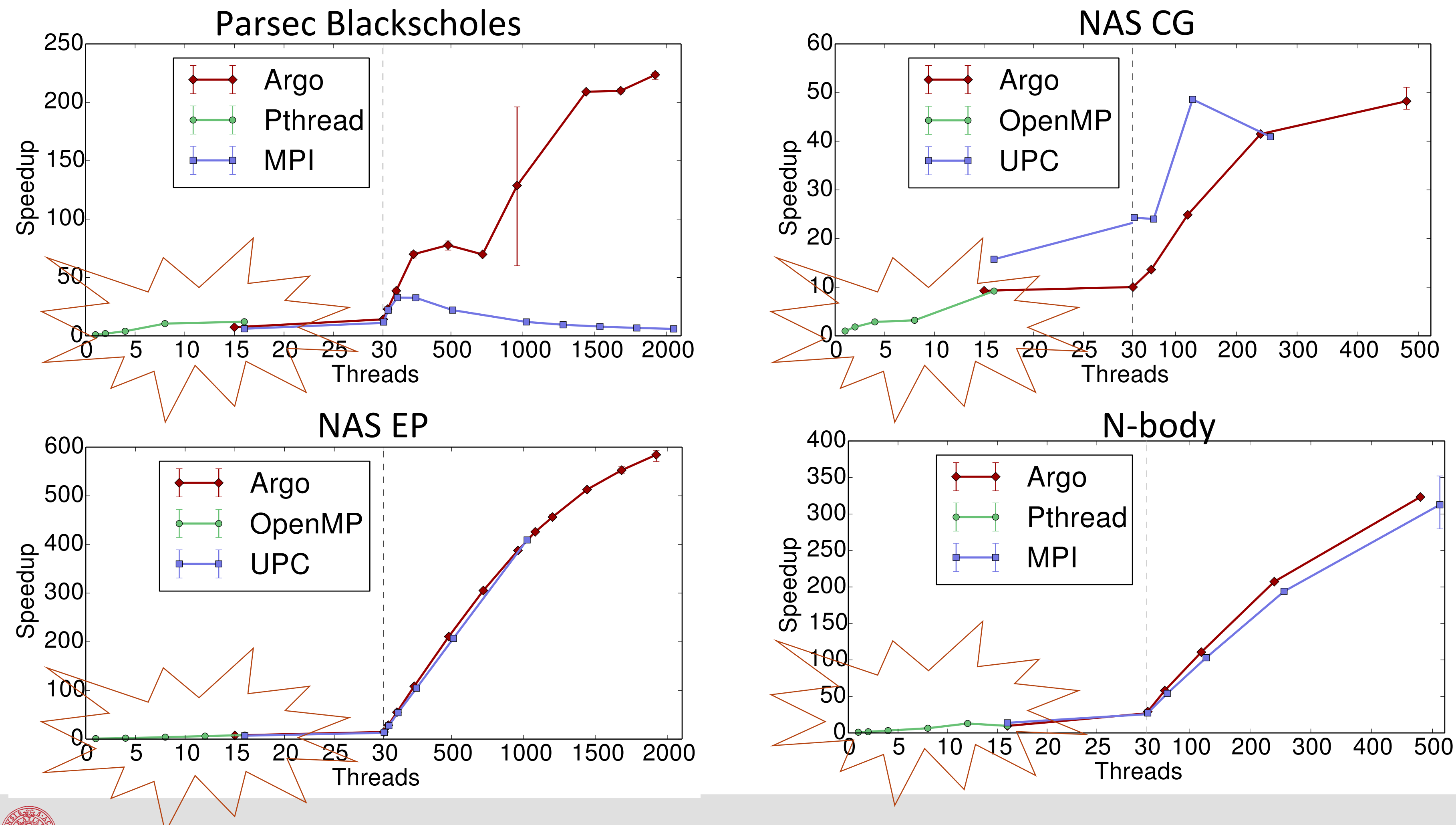
- A DSM based on current trends [HPDC'15]:
  - User-space implementation
  - Page-based DSM (uses virtual memory faults for misses)
  - MPI is the "network layer" (but only need RDMA)
  - Runs
    - Pthreads – Data race free programs
    - OpenMP - Manually ported (Minor modifications)
    - Compile and link with ArgoDSM library → MPI program that implements DSM

ARGO DSM (www.argodsm.com)

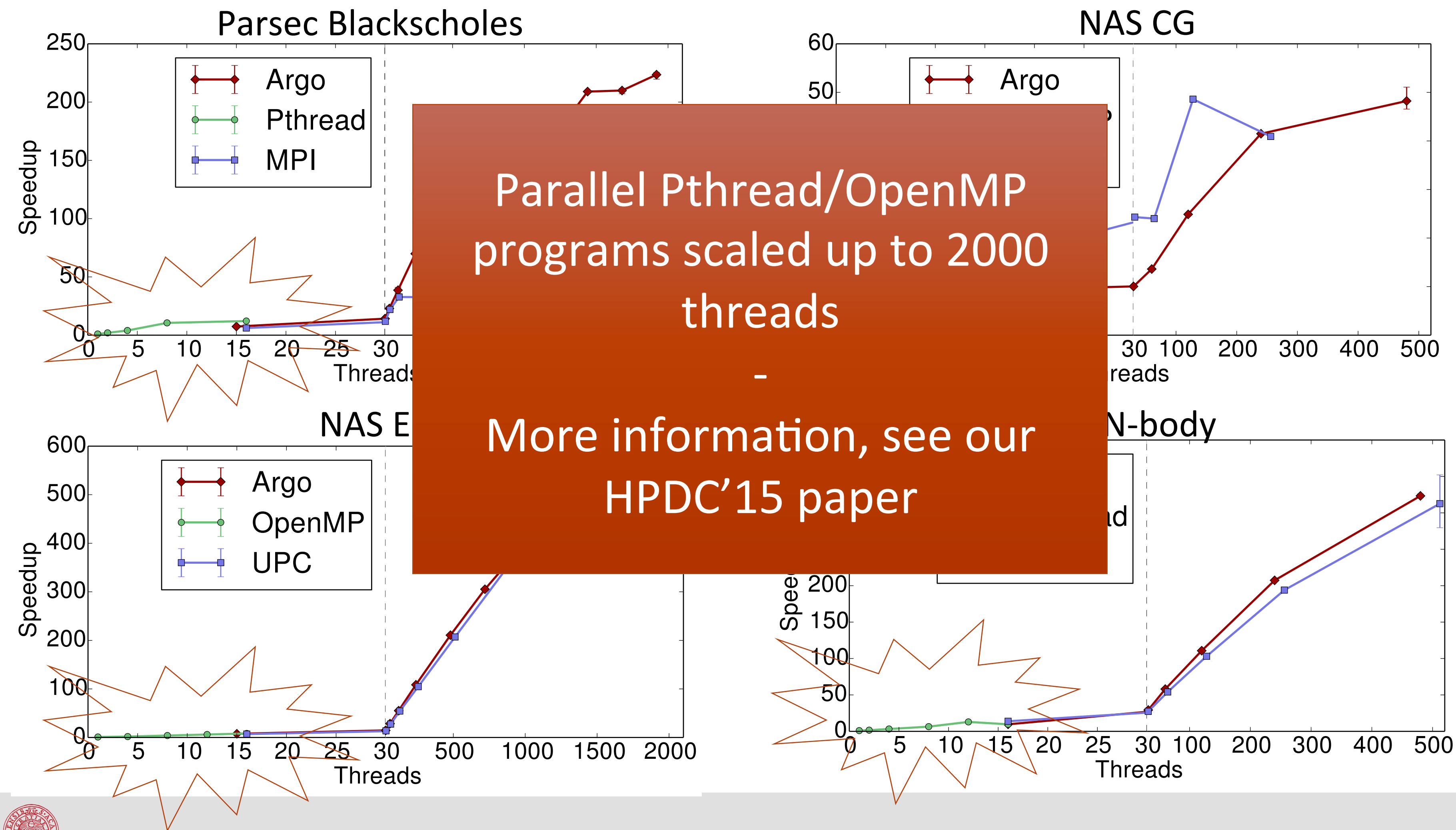
- Under the hood [HPDC'15]
  - SC for DRF (Sequential Consistency for Data Race Free)
    - Full relaxation of memory ordering between synchronization
  - Builds on very simple, completely distributed coherence (VIPS) [PACT'15, ISCA'13]
  - Queue Delegation Locking primitives [IEEE TPDS'18]

- Take decisions locally & Trade bandwidth for latency

Benchmark scaling
Pthread / OpenMP on ArgoDSM



Benchmark scaling
Pthread / OpenMP on ArgoDSM

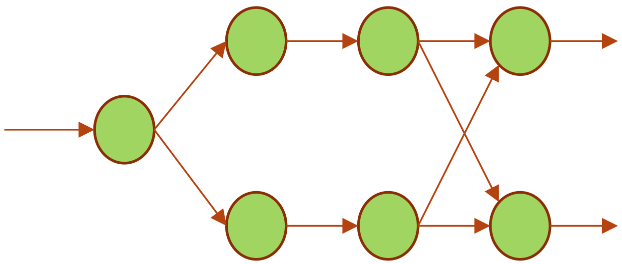


Parallel Pthread/OpenMP programs scaled up to 2000 threads

-

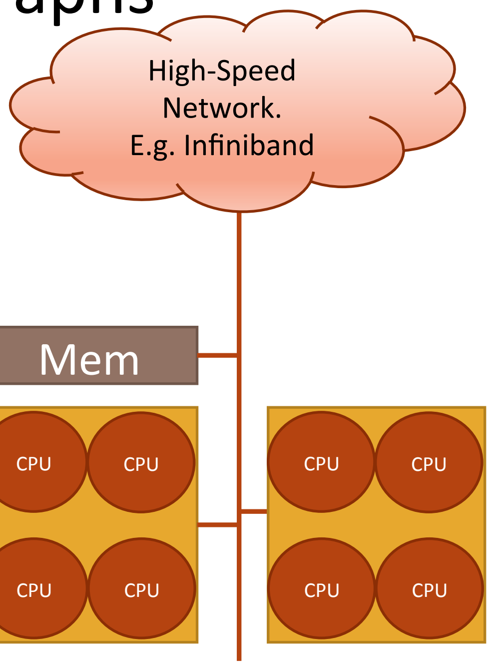
More information, see our HPDC'15 paper

Task based programming



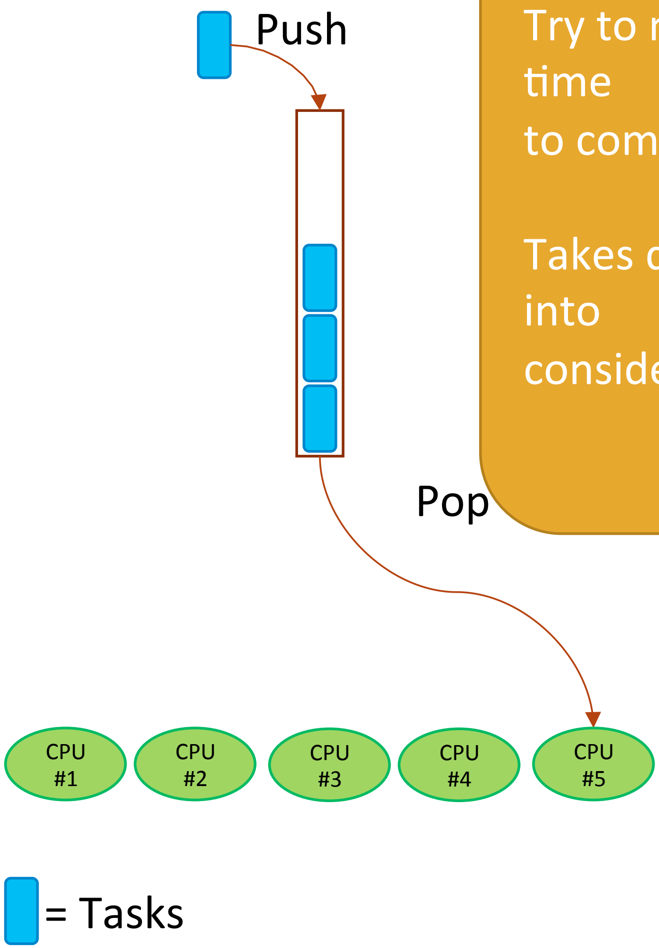
- Parallel programming model
  - **StarPU**, OmpSs, OpenMP, etc
- Express parallelism in terms of task-graphs
  - Can target heterogeneous systems
    - Our use case only runs on CPUs
  - High performance
    - Use kernels from BLAS libraries
- Challenge: How to do scheduling?
  - Eager
  - Data aware
  - Custom…

Couple with ArgoDSM for cluster awareness!



Task schedulers

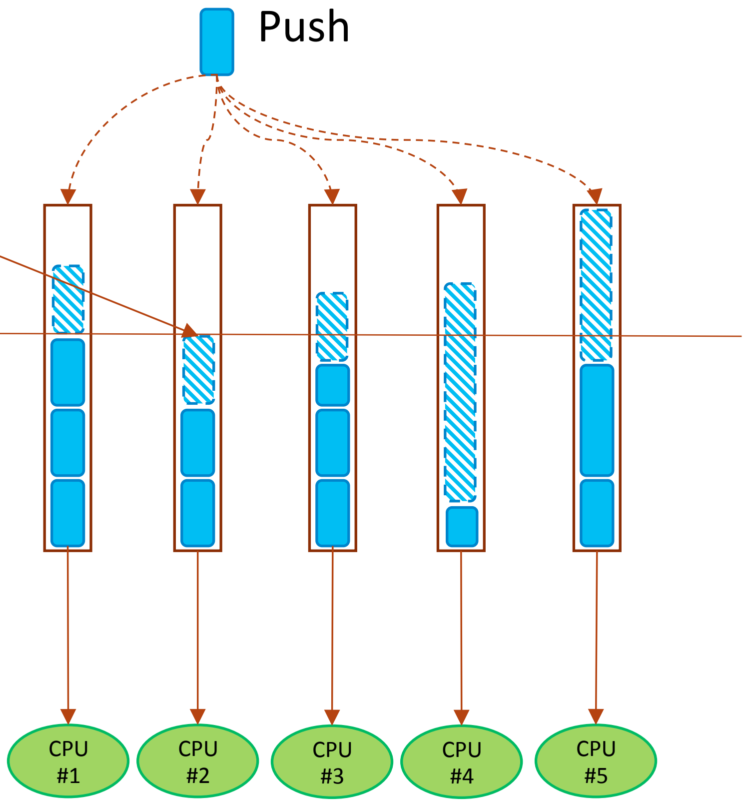
Eager Scheduling



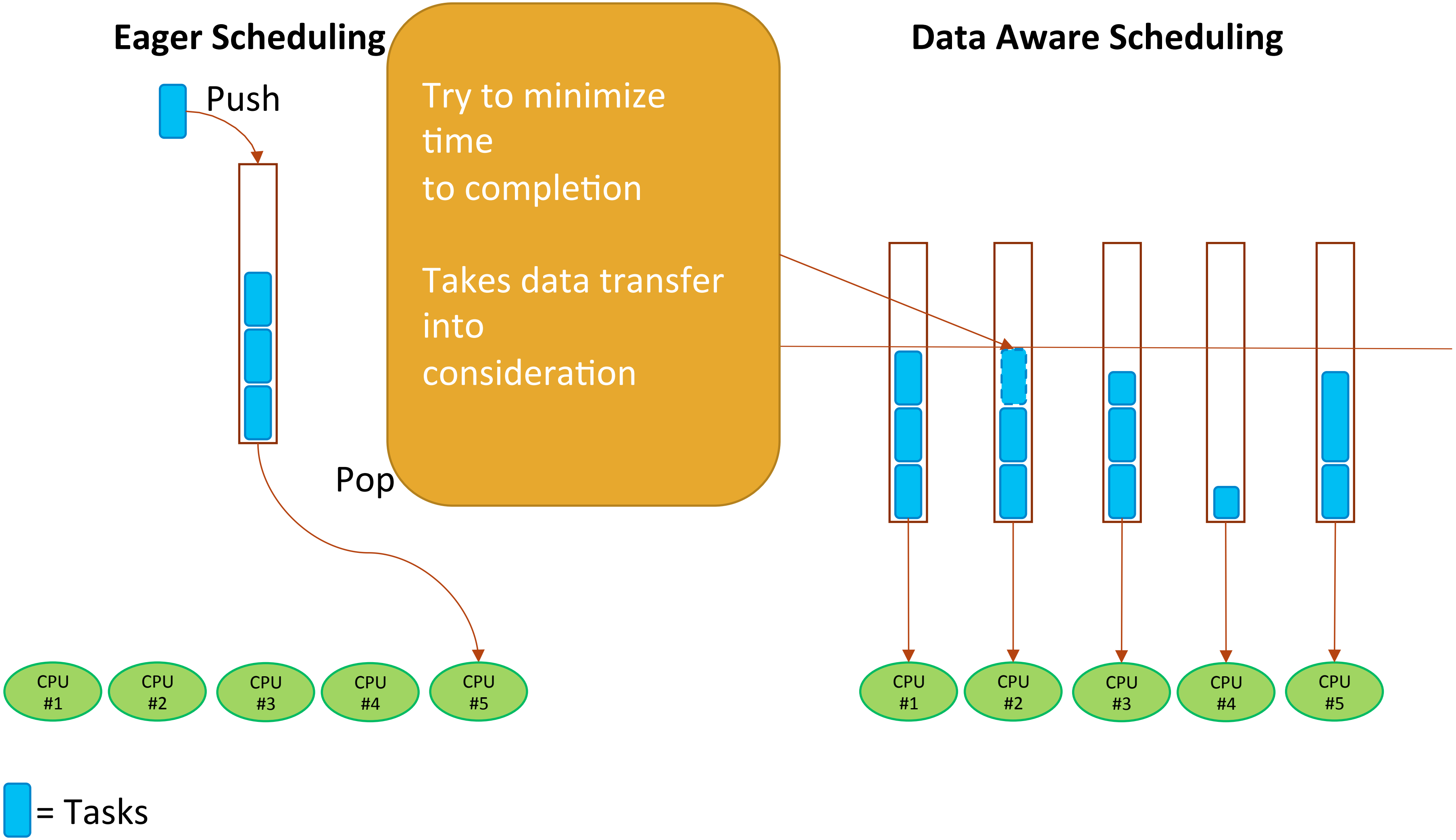
Try to minimize time to completion

Takes data transfer into consideration

Data Aware Scheduling



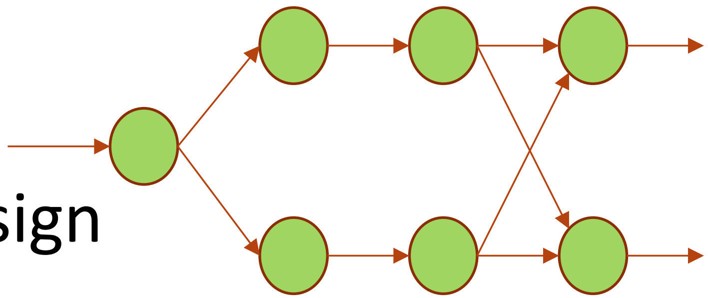
Task schedulers



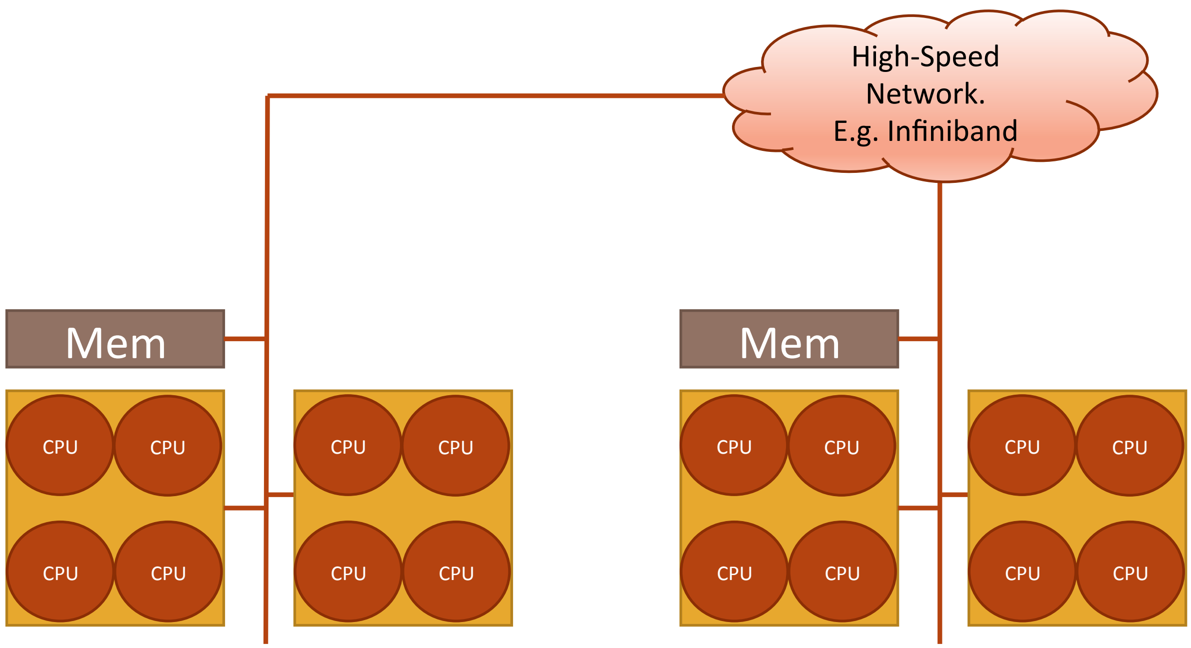
Try to minimize time
to completion

Takes data transfer into
consideration

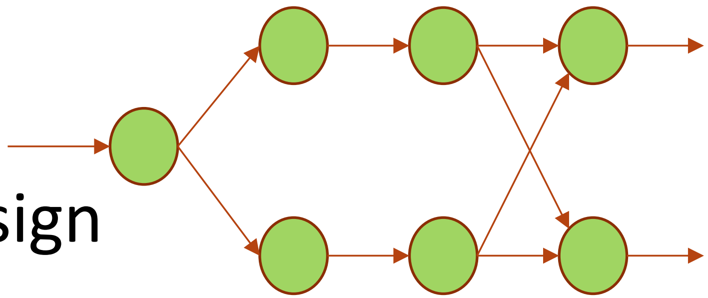
The perfect fit ArgoDSM+Tasks



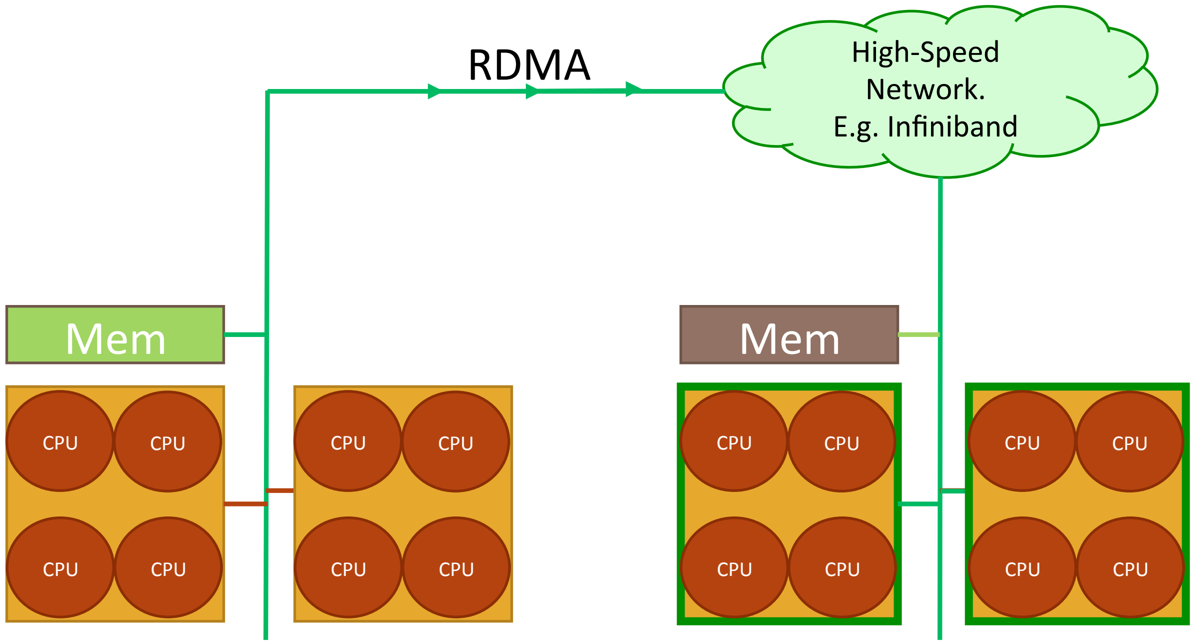
- Tasks over ArgoDSM Co-design
  - ArgoDSM can
    - Handle caching of remote data making it appear as local



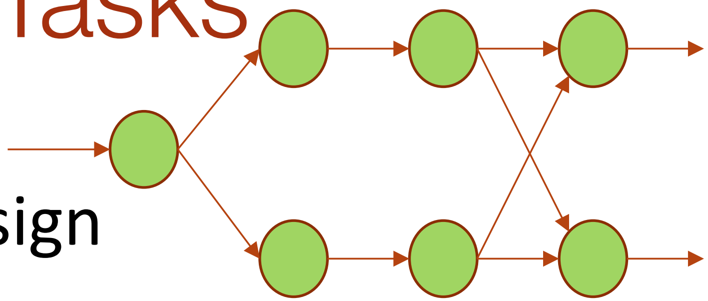
The perfect fit ArgoDSM+Tasks



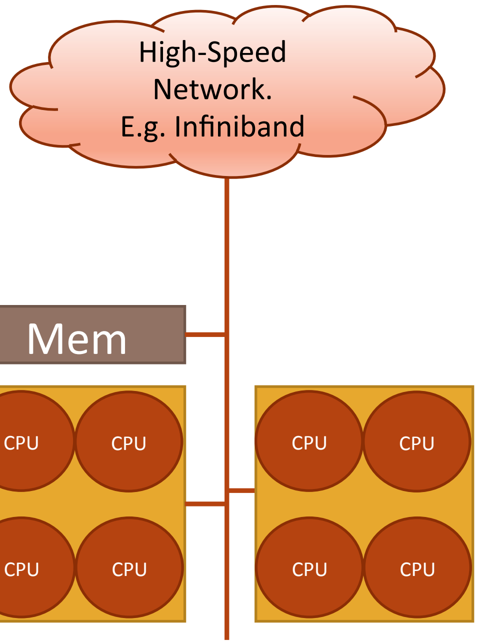
- Tasks over ArgoDSM Co-design
  - ArgoDSM can
    - Handle caching of remote data making it appear as local



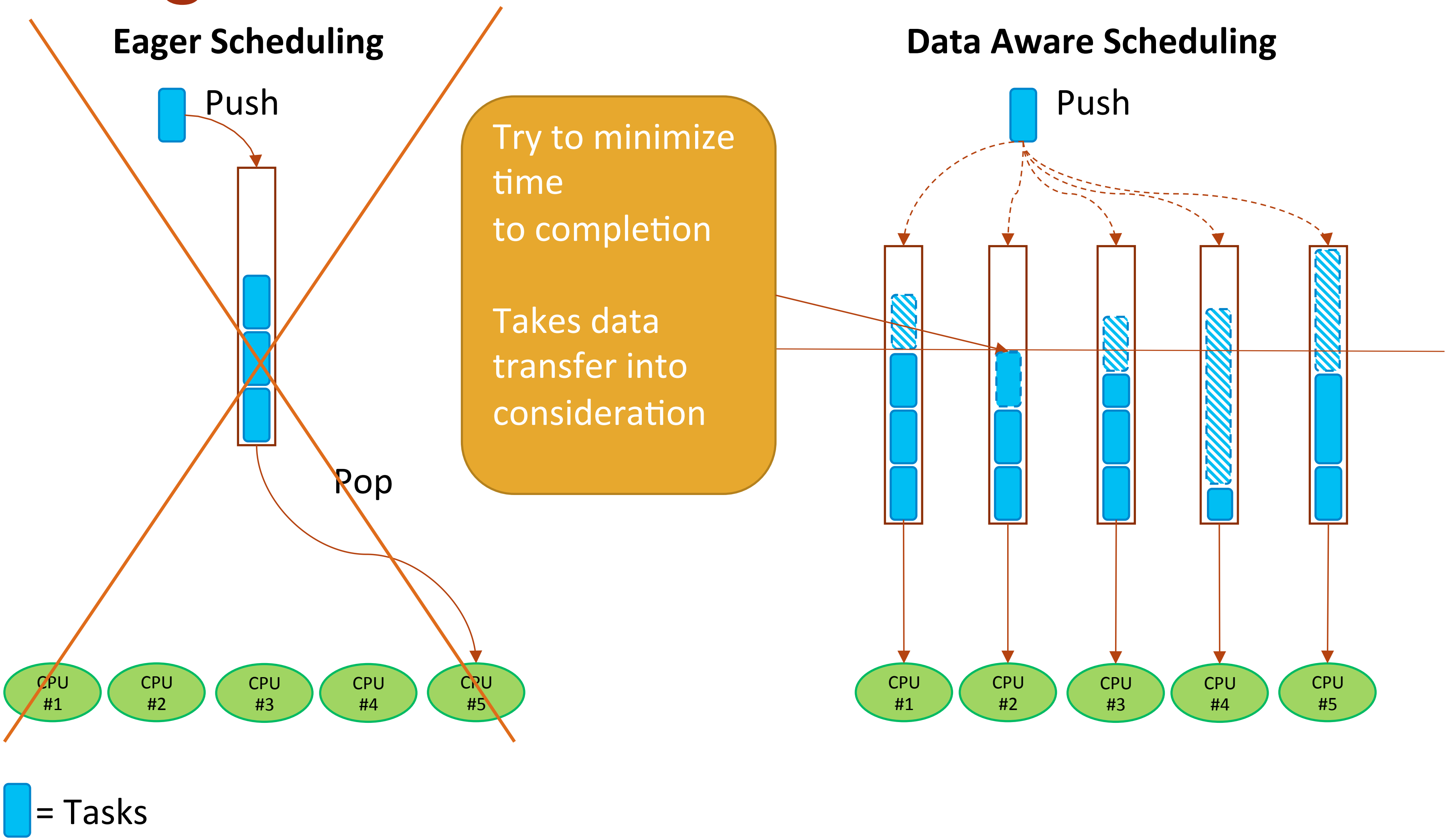
The perfect fit
Cluster level DSM+Tasks



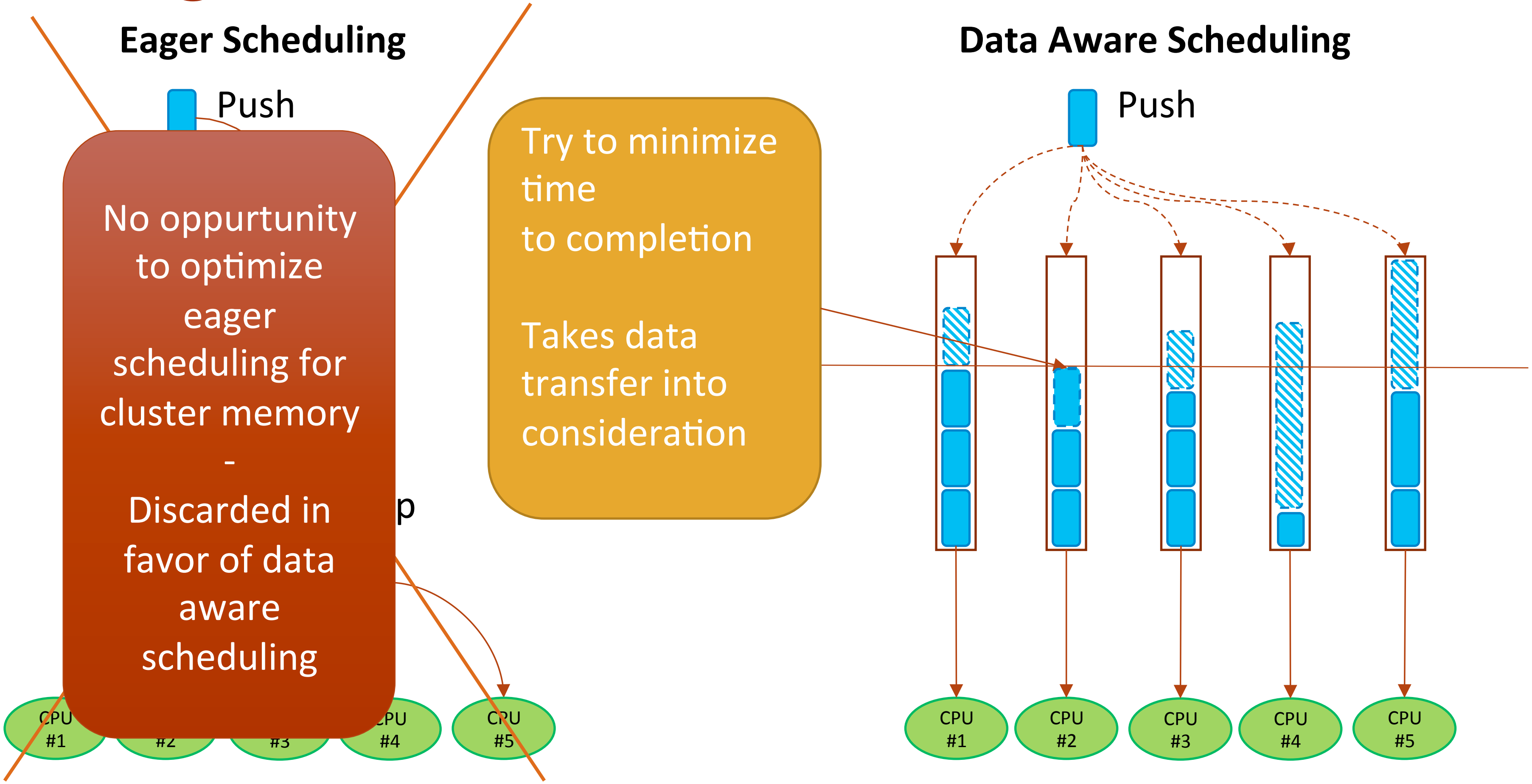
- Tasks over ArgoDSM Co-design
  - ArgoDSM can
    - Handle caching of remote data making it appear as local
    - Exploit task information
      - Prefetch data ahead of time
      - Prevent eviction of useful data
  - StarPU: performance models
    - ➔ ArgoDSM (cluster) locality!



ArgoDSM aware scheduler

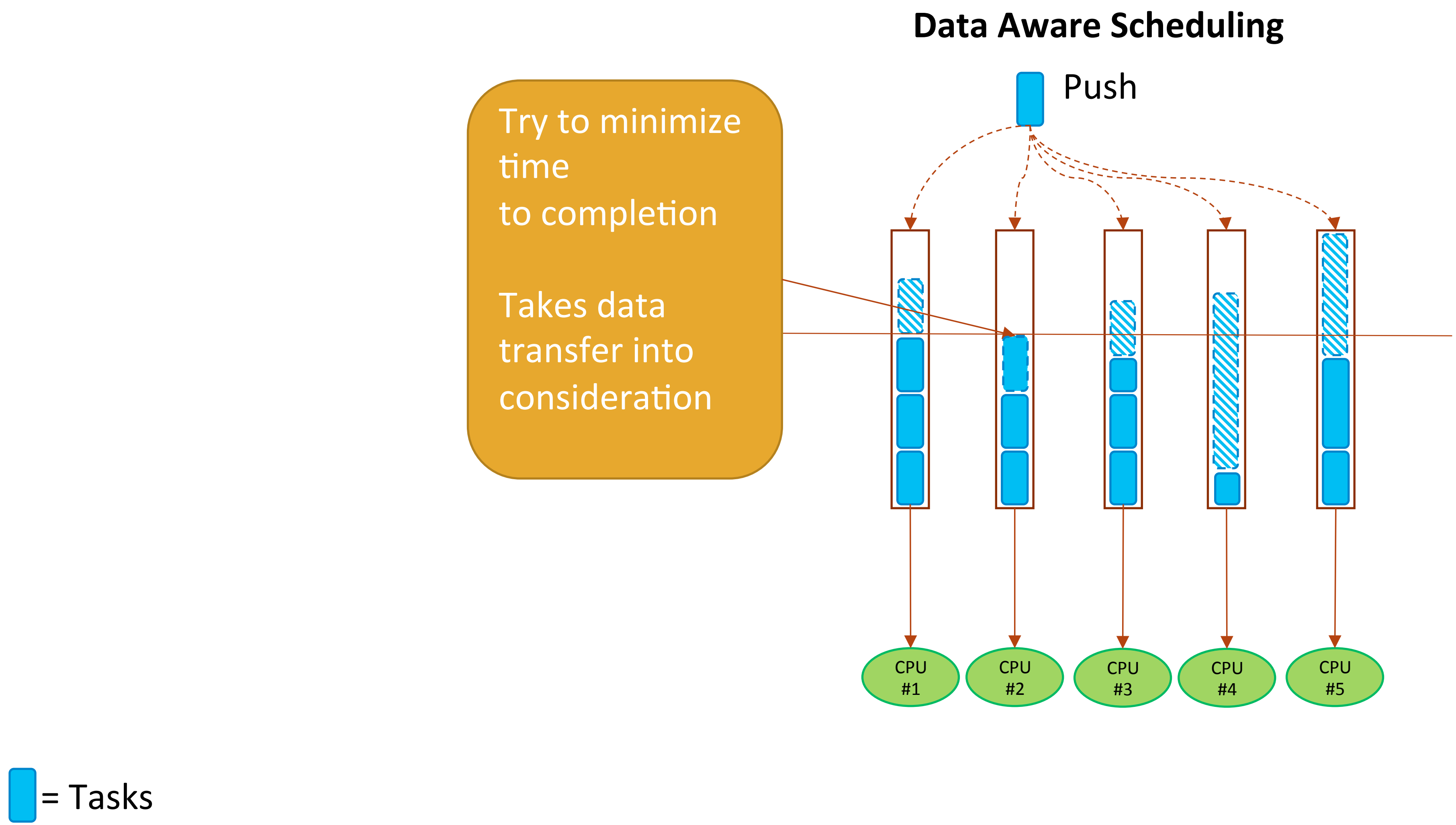


ArgoDSM aware scheduler



= Tasks

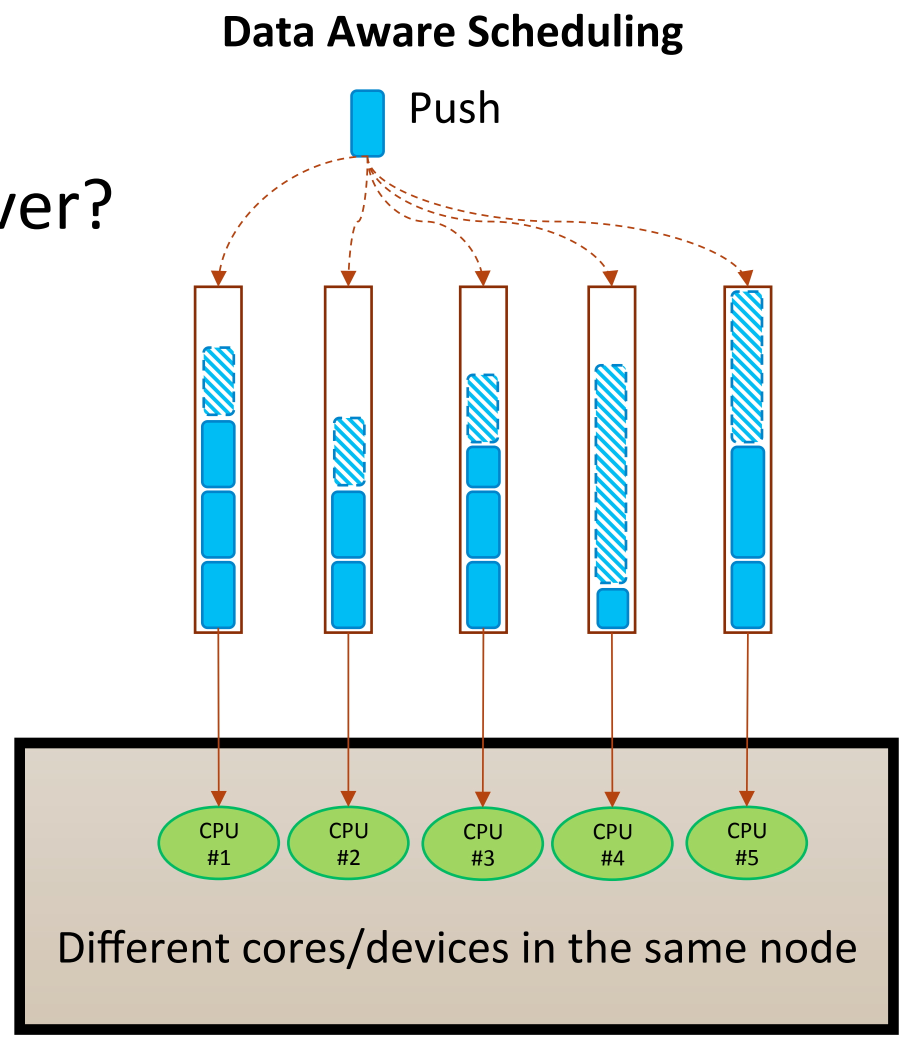
ArgoDSM aware scheduler



ArgoDSM aware scheduler
Data Aware

ArgoDSM aware scheduler
Data Aware

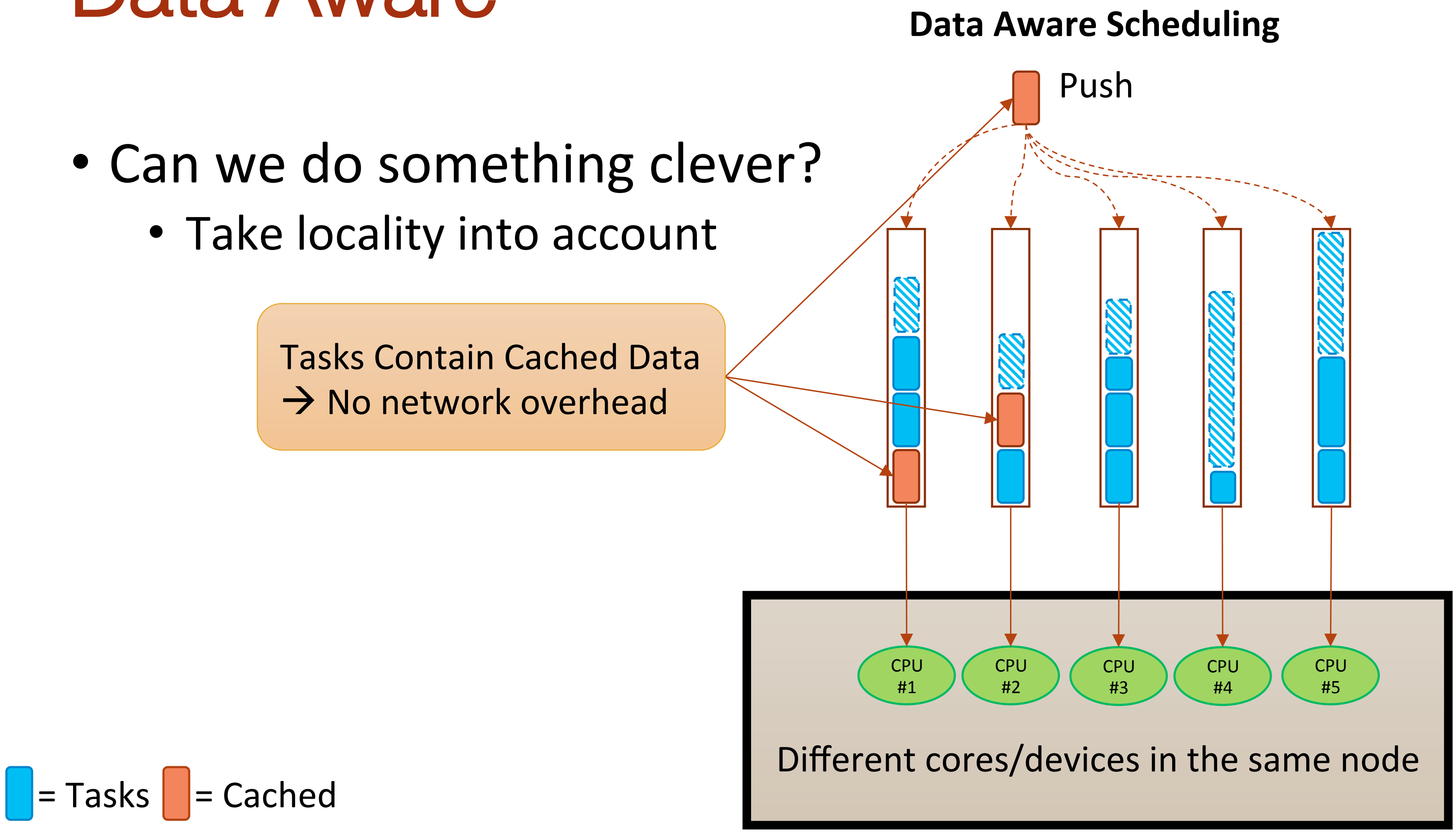
- Can we do something clever?



= Tasks

ArgoDSM aware scheduler
Data Aware

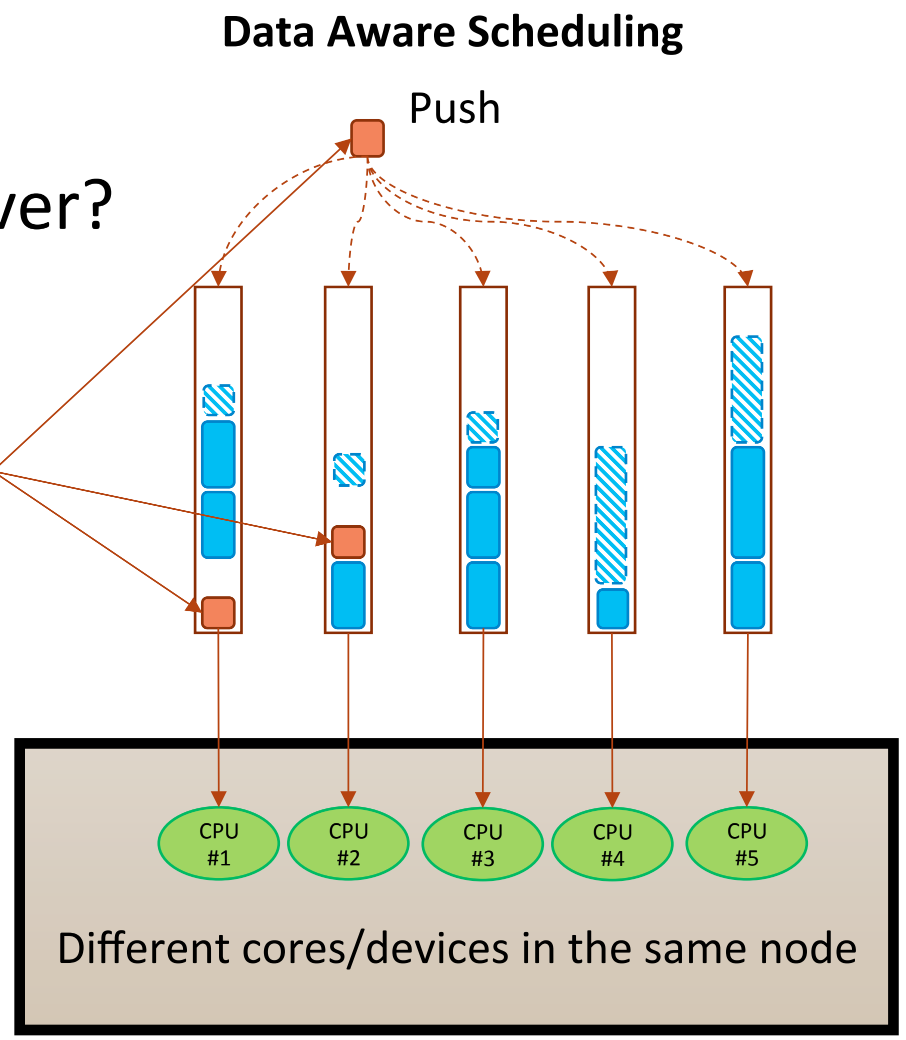
- Can we do something clever?
  - Take locality into account



ArgoDSM aware scheduler
Data Aware

- Can we do something clever?
  - Take locality into account

Tasks Contain Cached Data
→ No network overhead

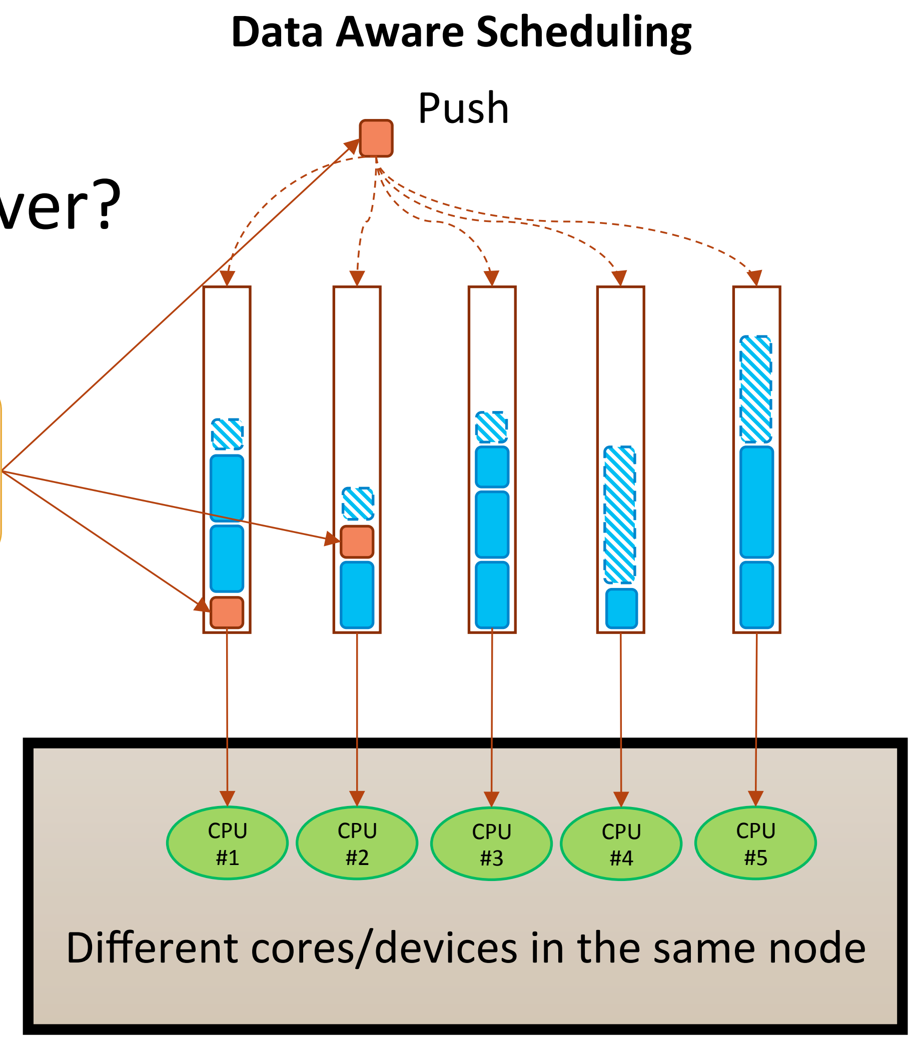


= Tasks = Cached

ArgoDSM aware scheduler
Data Aware

- Can we do something clever?
  - Take locality into account

Tasks Contain Cached Data
→ No network overhead

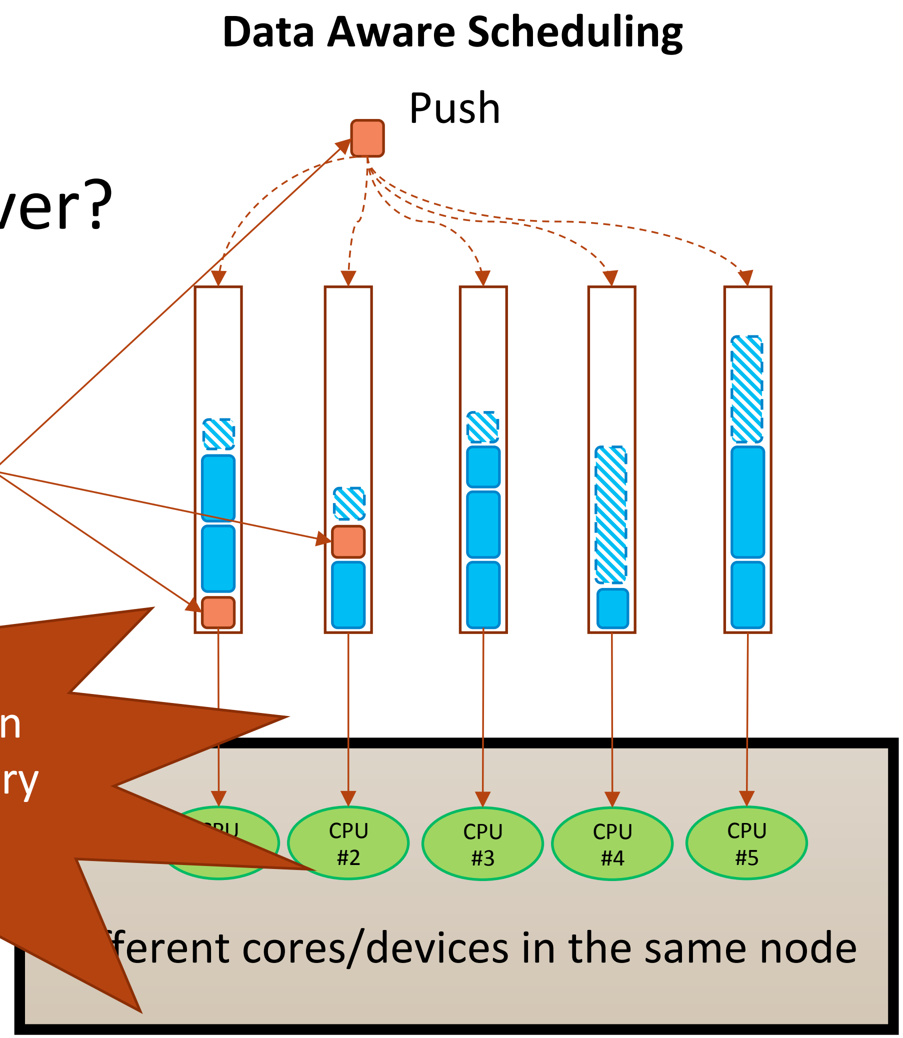


= Tasks = Cached

ArgoDSM aware scheduler
Data Aware

- Can we do something clever?
  - Take locality into account

Tasks Contain Cached Data
→ No network overhead



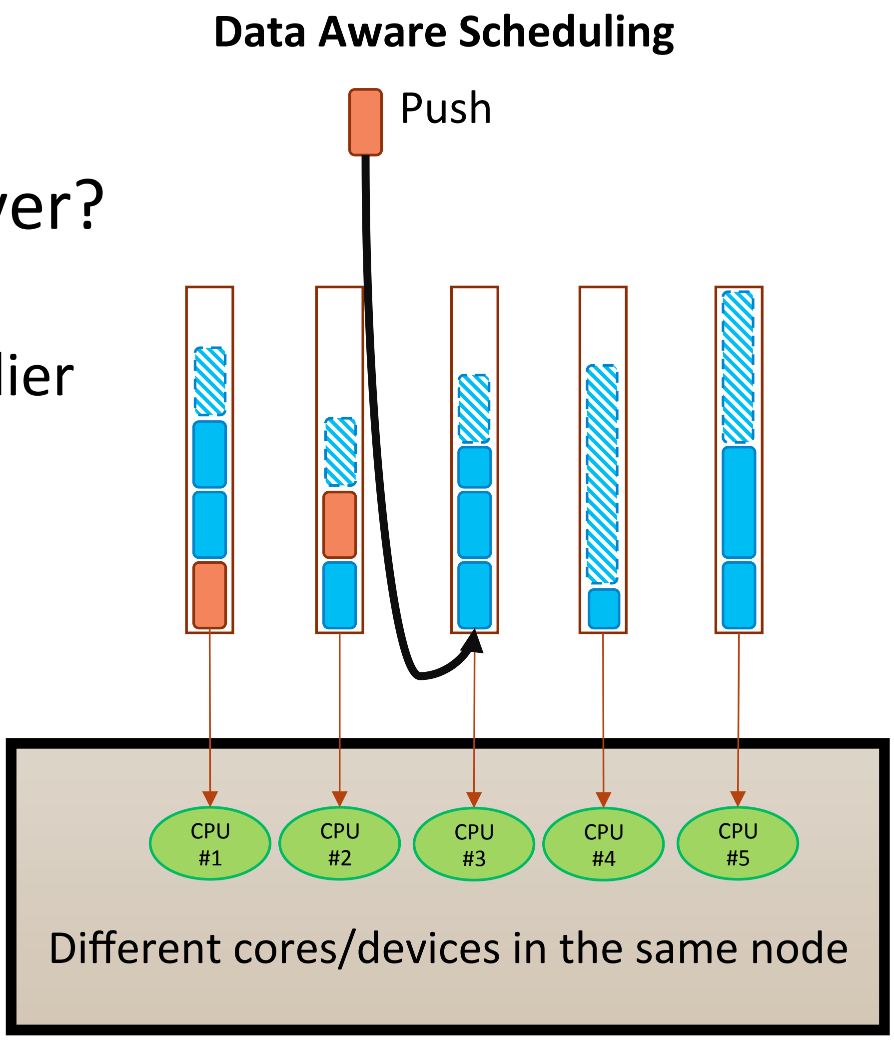
Better runtime prediction
adapted to cluster memory
→ Higher performance

= Tasks = Cached

ArgoDSM aware scheduler
Local First

ArgoDSM aware scheduler
Local First

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier



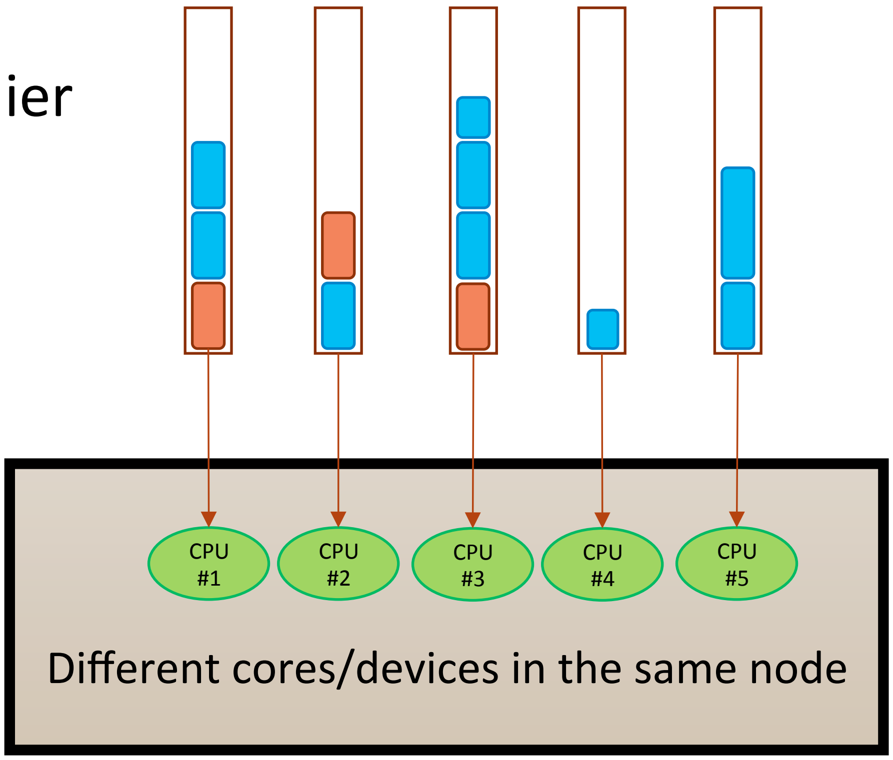
= Tasks = Cached

ArgoDSM aware scheduler
Local First

Data Aware Scheduling

Push

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier

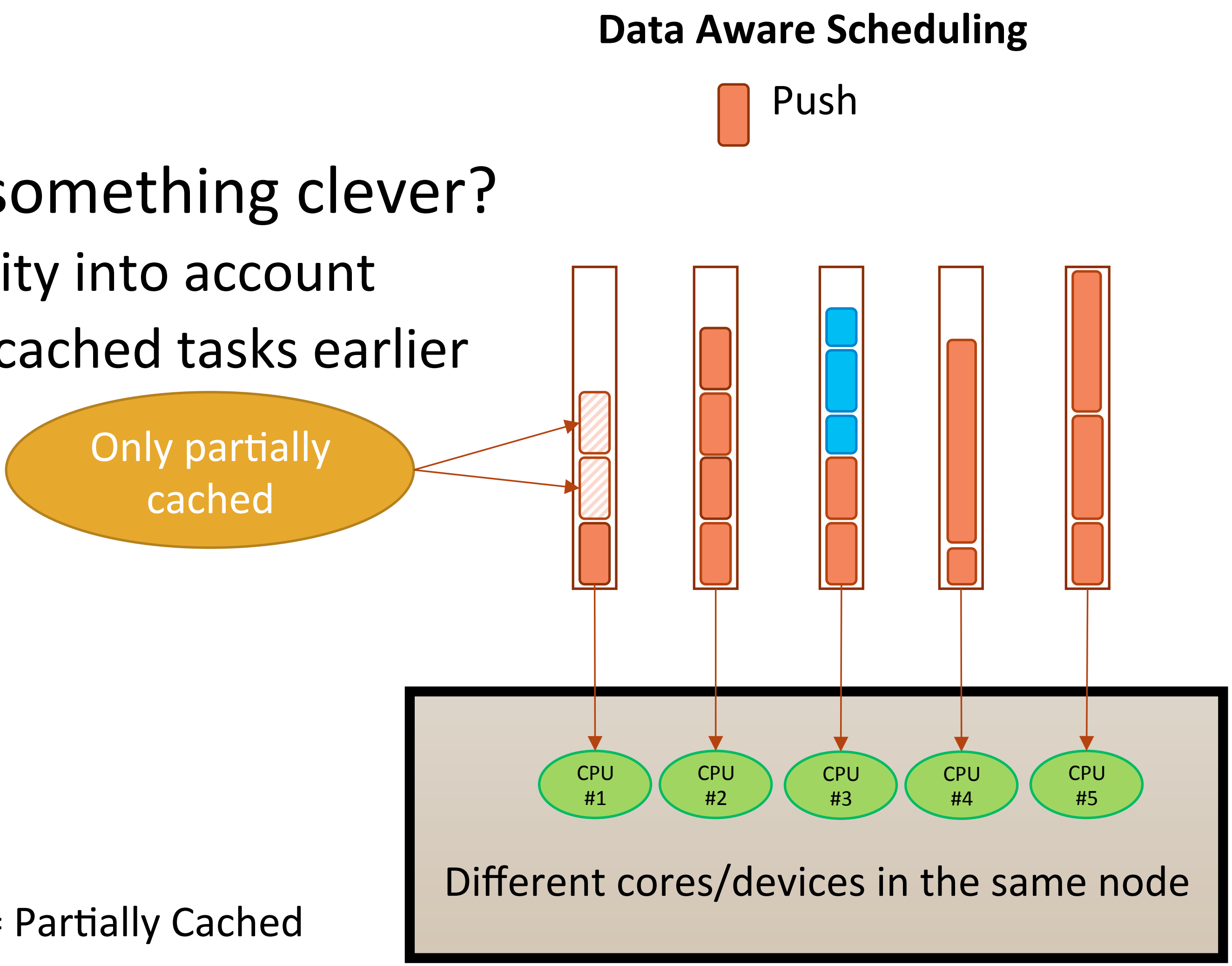


= Tasks = Cached

ArgoDSM aware scheduler
Sorted Insert

ArgoDSM aware scheduler
Sorted

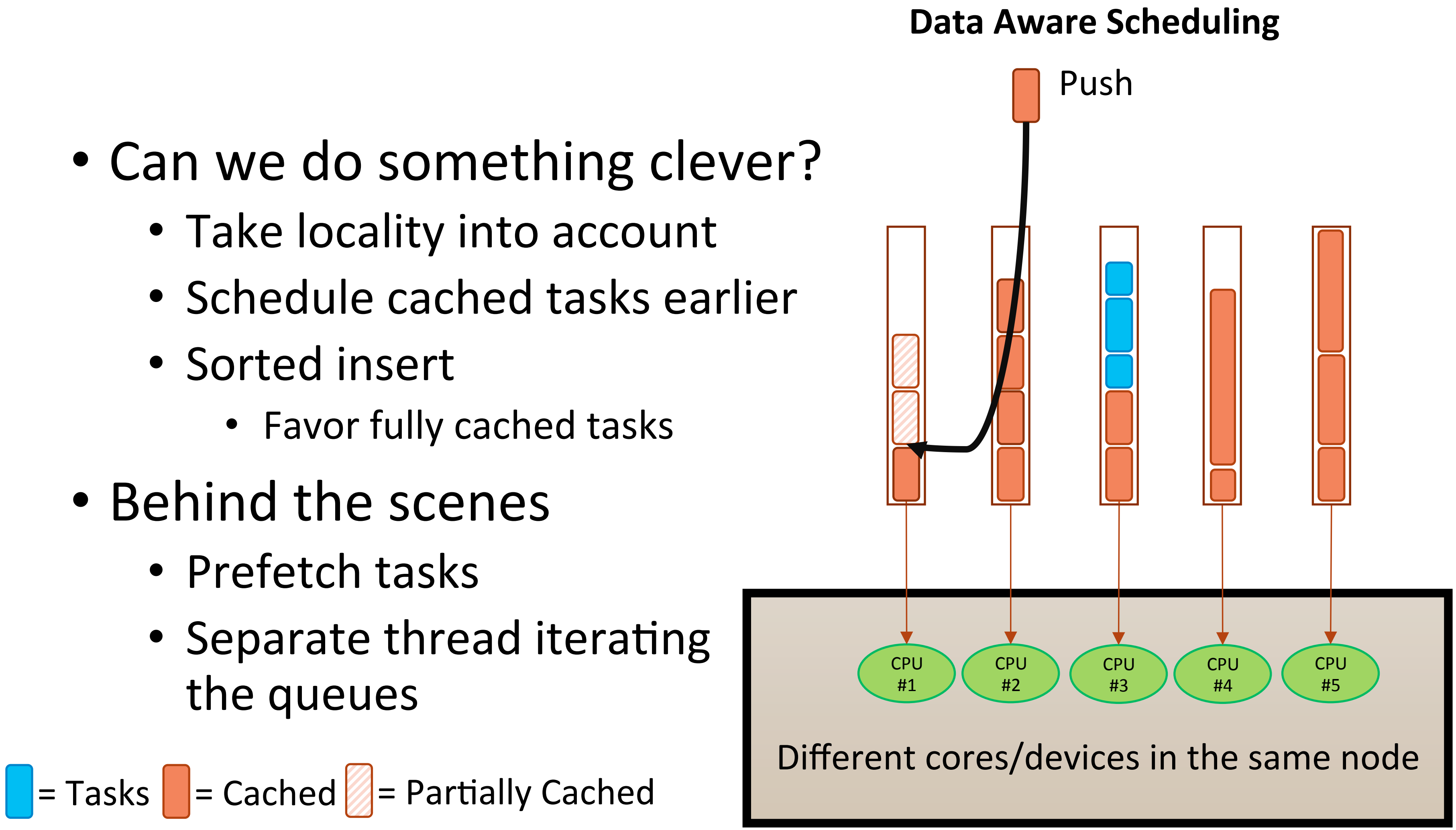
- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier



= Tasks = Cached = Partially Cached

ArgoDSM aware scheduler

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier
  - Sorted insert
    - Favor fully cached tasks
- Behind the scenes
  - Prefetch tasks
  - Separate thread iterating the queues

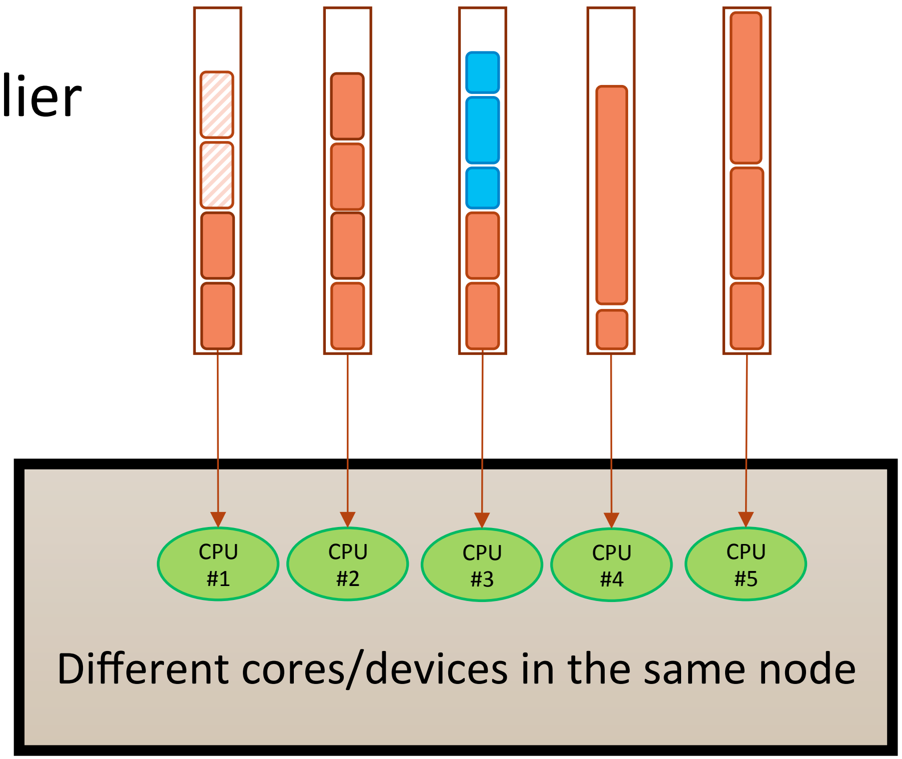


ArgoDSM aware scheduler

Data Aware Scheduling

Push

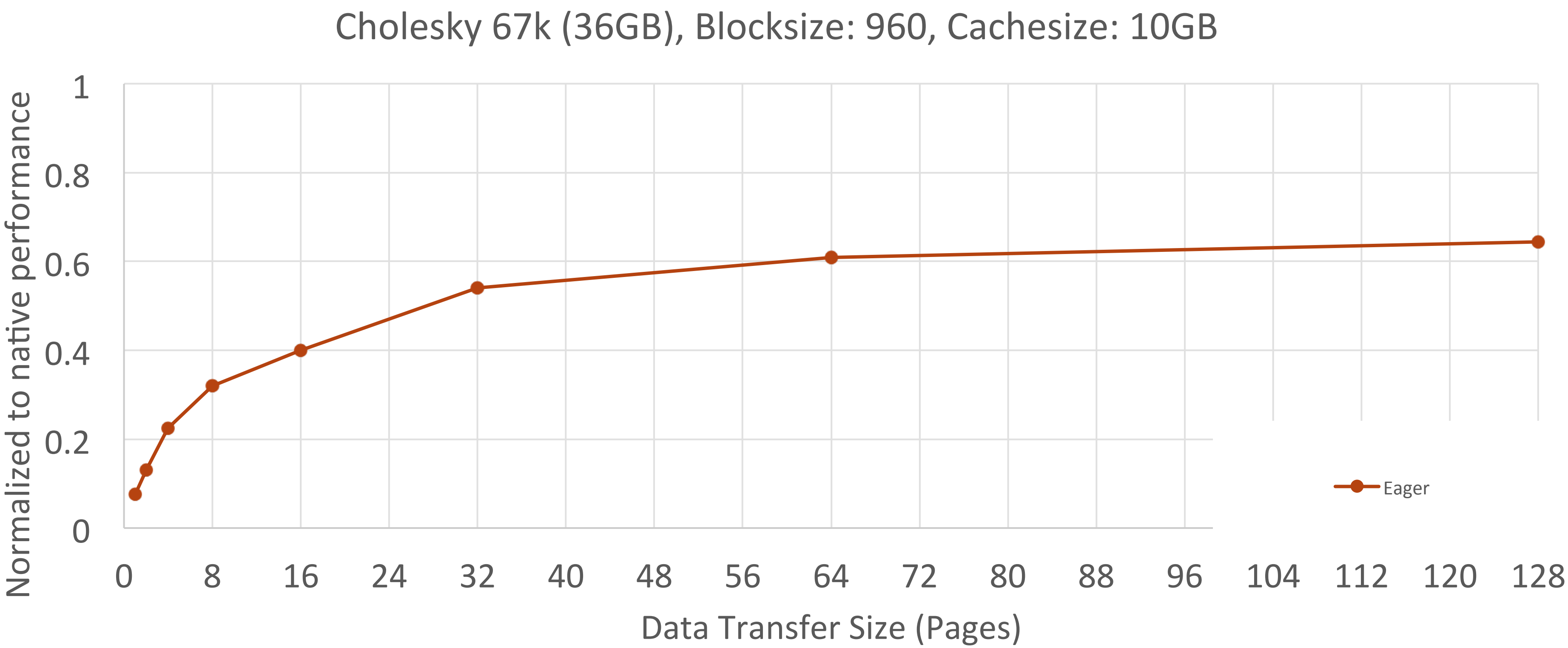
- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier
  - Sorted insert
    - Favor fully cached tasks
- Behind the scenes
  - Prefetch tasks
  - Separate thread iterating the queues



= Tasks = Cached = Partially Cached

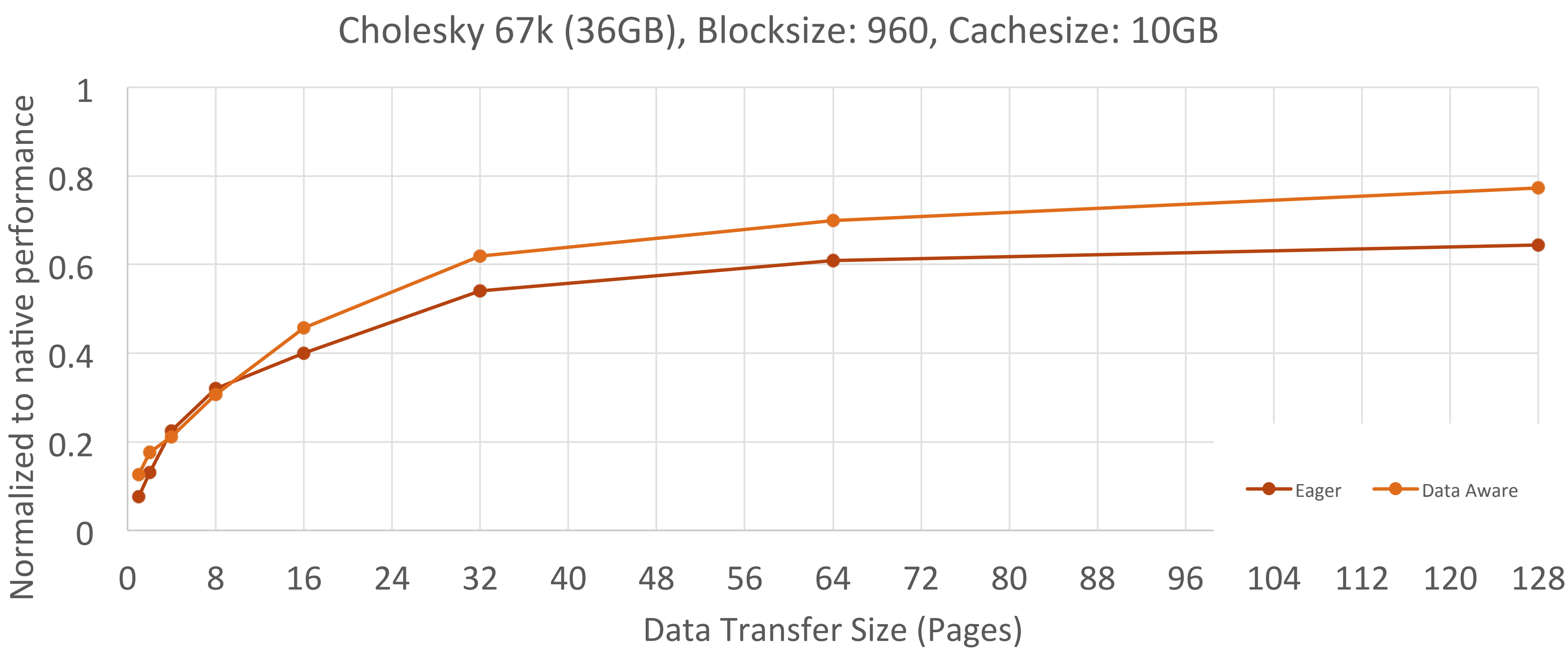
Cholesky

- Coarse grain data transfer make use of BW better
  - 1800 pages per task buffer
  - >32 pages data transfer unit to get performance



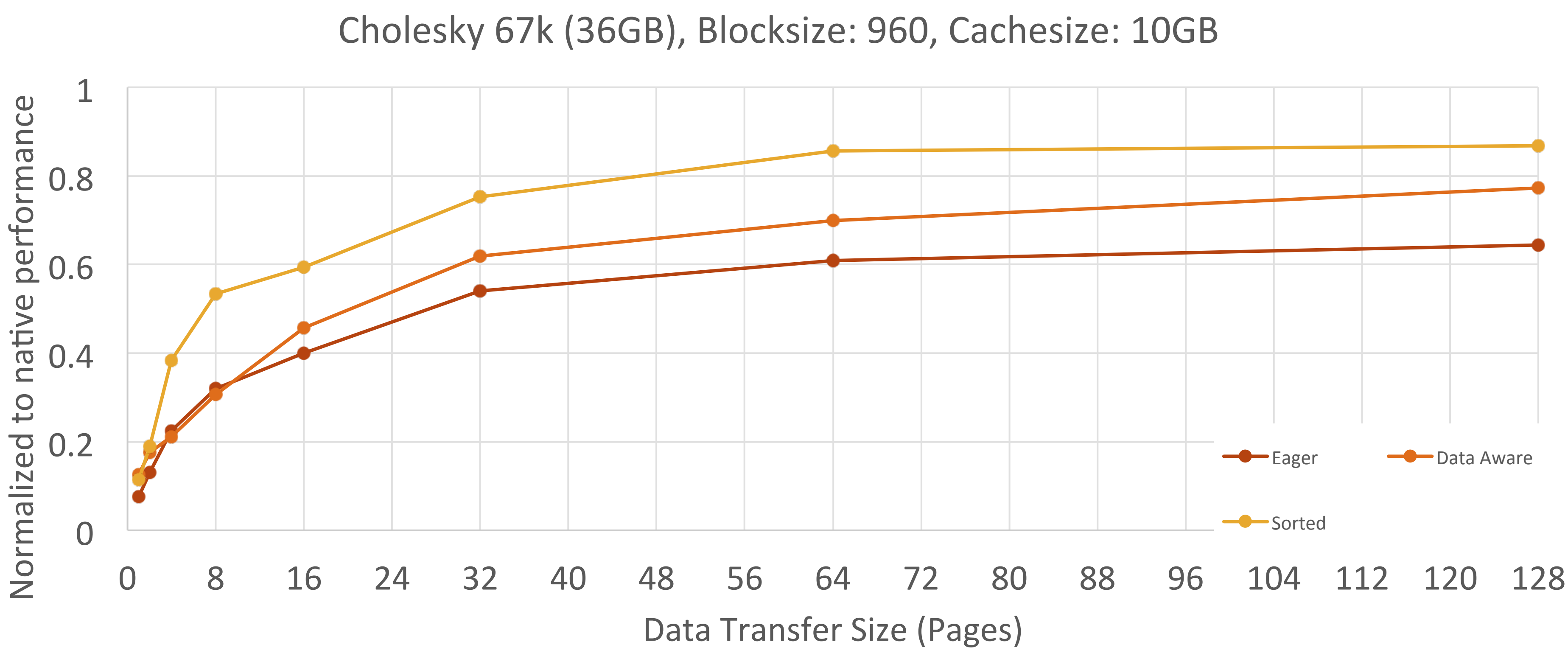
Cholesky

- Data aware – increases performance by 10%



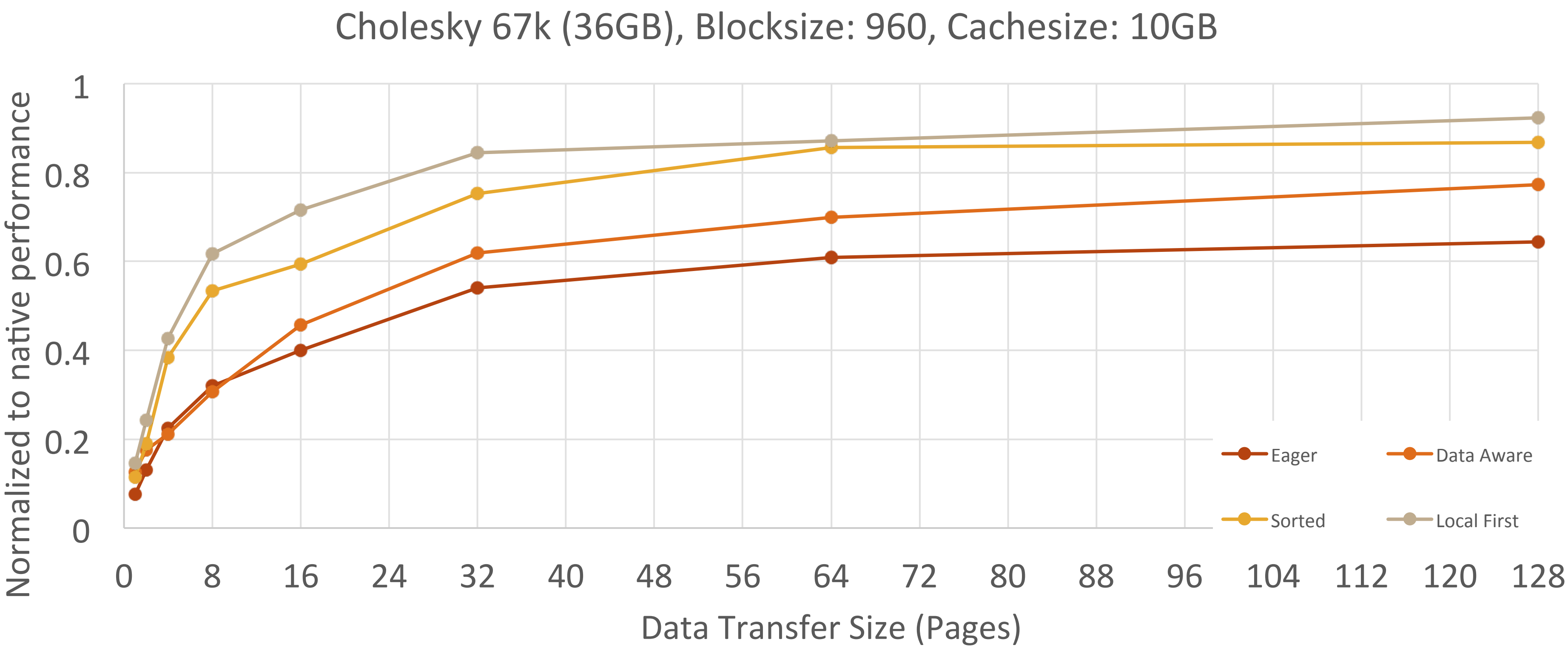
Cholesky

- *Sorted* - Inserts tasks based on how much is cached
  - Increase performance another ~10%



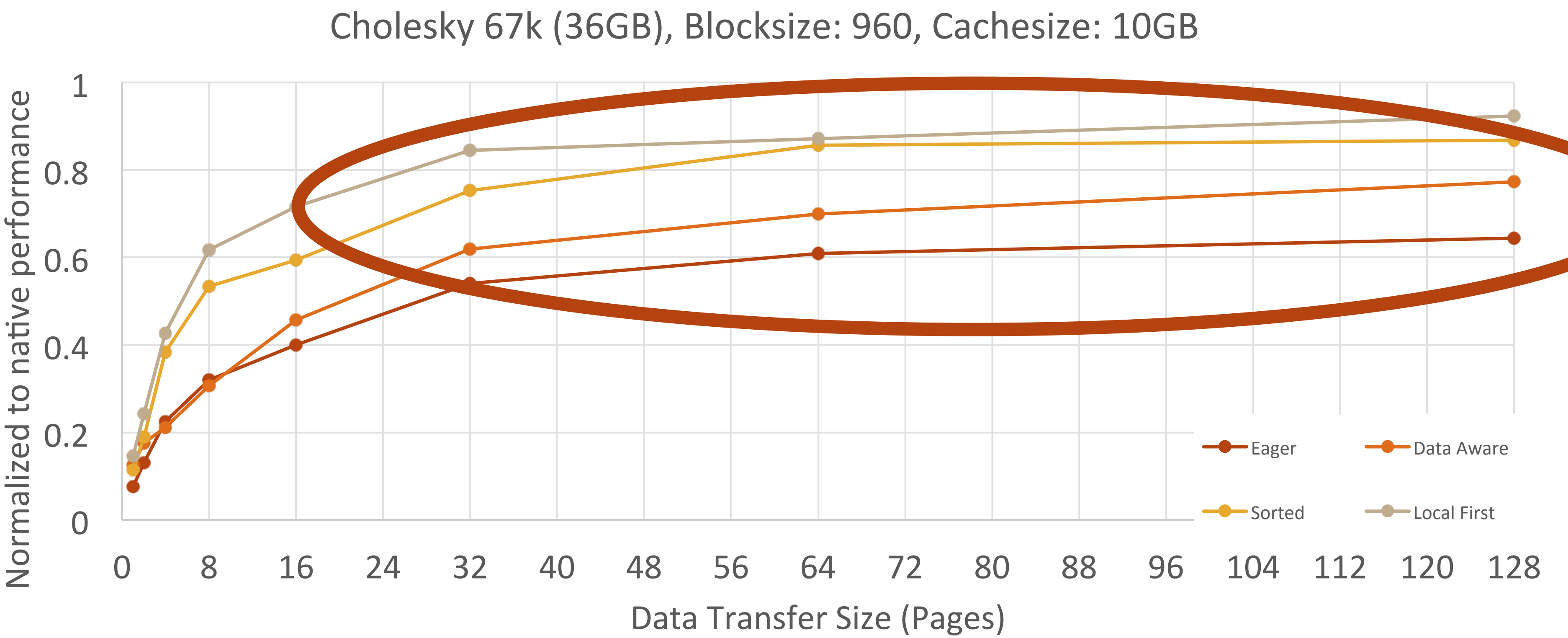
Cholesky

- *Local First* - Any cached task is always placed first
  - Small chance of data evicted when executed
  - Increases performance up 90%+ of performance

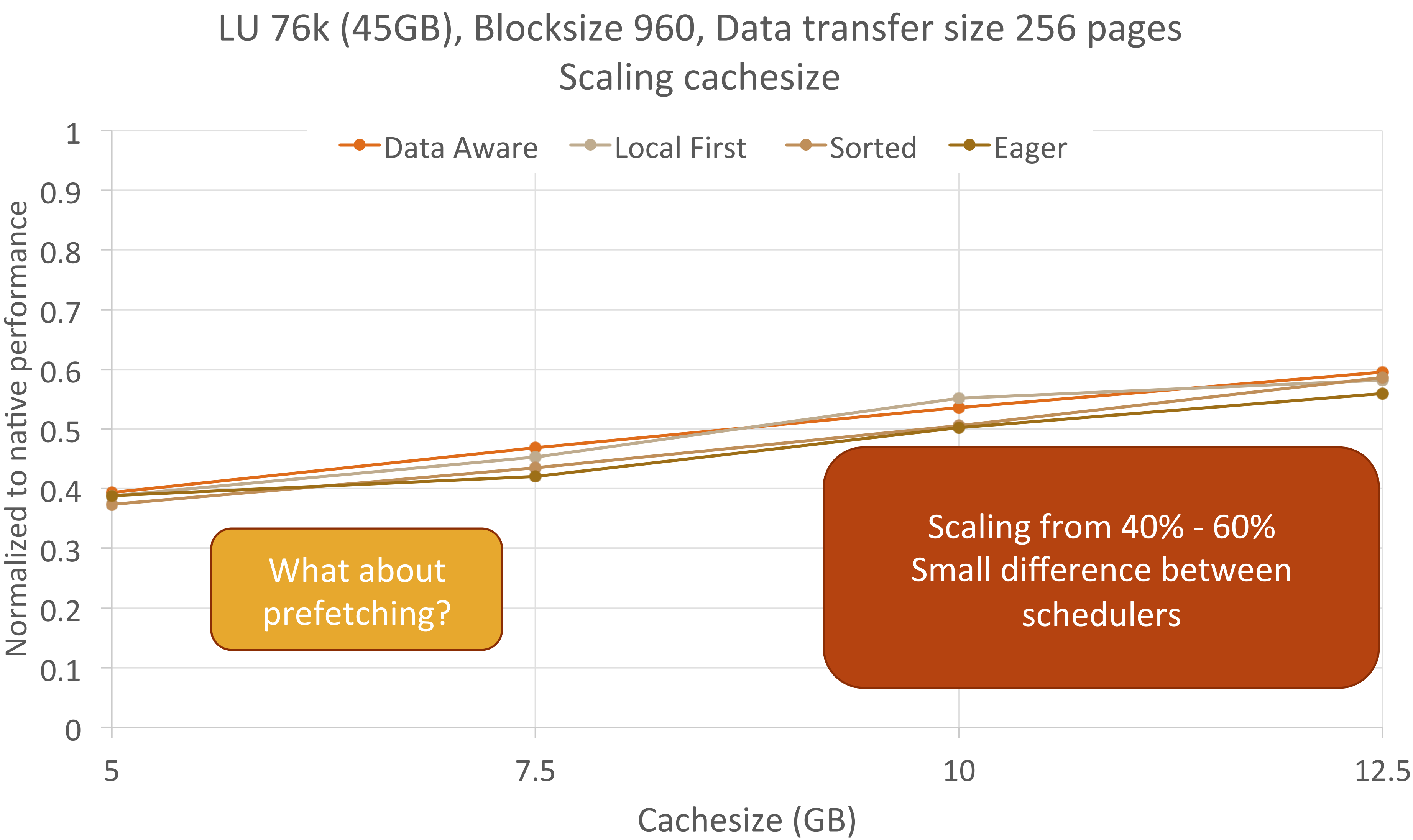


Cholesky

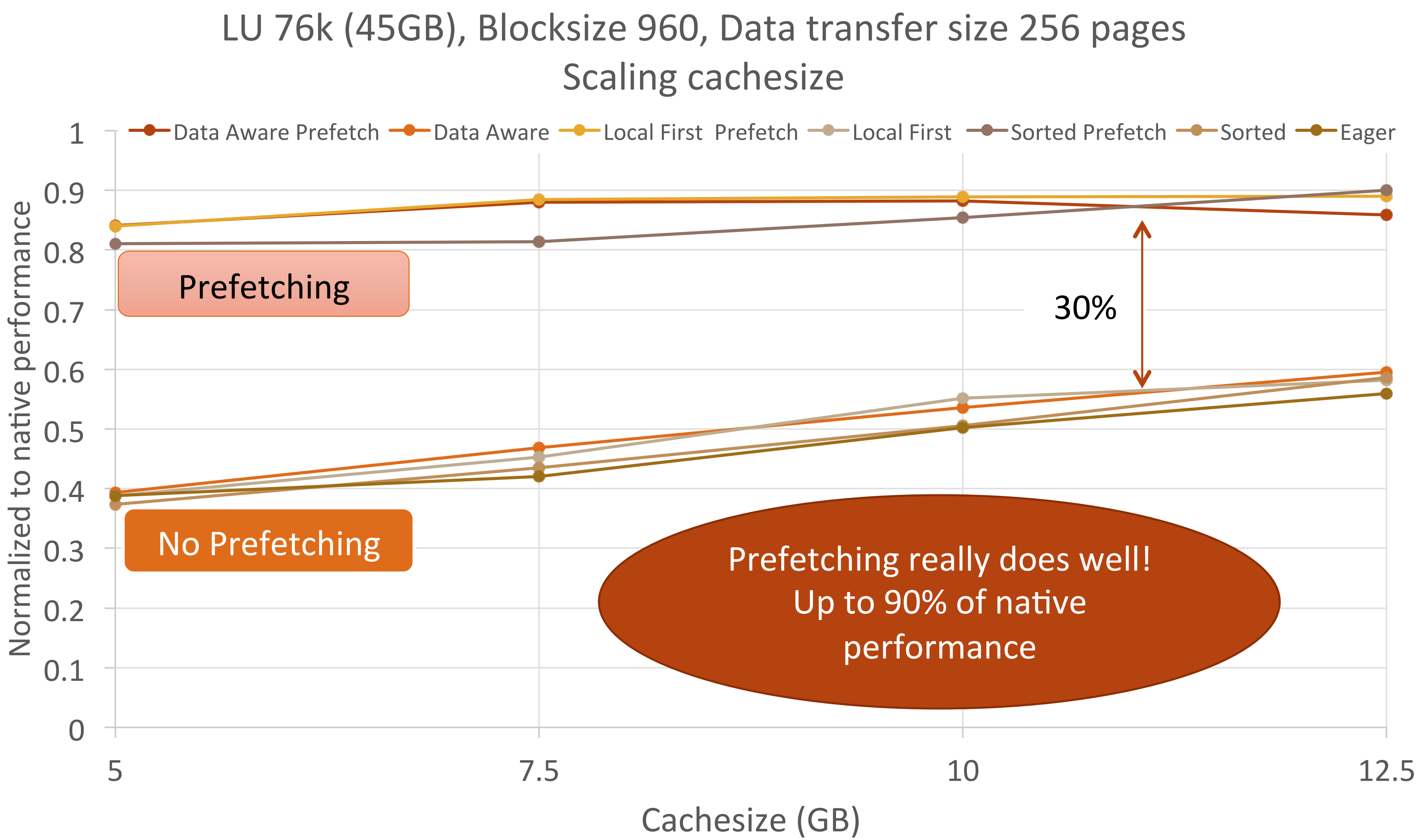
- Coarse grain data transfer make use of BW better
  - 1800 pages per task buffer
  - 32 pages data transfer unit get performance



LU


LU 76k (45GB), Blocksize 960, Data transfer size 256 pages
Scaling cachesize
Data Aware
Local First
Sorted
Eager
Normalized to native performance
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
7.5
10
12.5
Cachesize (GB)
What about prefetching?
Scaling from 40% - 60%
Small difference between schedulers

LU


LU 76k (45GB), Blocksize 960, Data transfer size 256 pages
Scaling cachesize
Data Aware Prefetch
Data Aware
Local First Prefetch
Local First
Sorted Prefetch
Sorted
Eager
Normalized to native performance
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
7.5
10
12.5
Cachesize (GB)
Prefetching
No Prefetching
30%
Prefetching really does well!
Up to 90% of native
performance

The next step – Scale computation

- Using our method we have 2 degrees of freedom
  - Move data or tasks (EPEEC – Horizon 2020)
- Optimize at cluster level
  - How to divide up work?
    - Cluster global work queues?
    - Open question: When do we synchronize?
  - Give the scheduler locality information to improve scheduling
  - Give the DSM task information
    - Similar to this talk, memory is the important part

Conclusions

- Task-based systems are a great fit for DSMs

Conclusions

- DSMs are a great fit for a Task based systems
  - ArgoDSM can make use of efficient caching and prefetching based on scheduler information
  - Scheduling policies can make use of DSM information to schedule for locality
- As advertised:
  - Up to 90%+ of native performance operating on remote memory using a co-design of scheduler and DSM caching techniques

Thank you!
Questions?

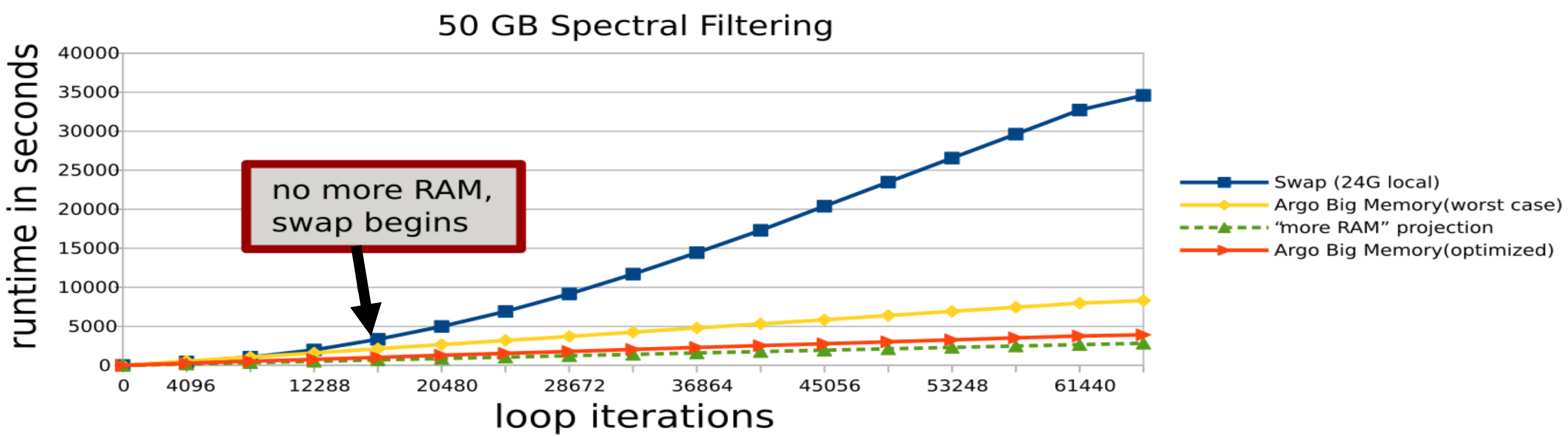
Magnus.Norgren@it.uu.se
www.argodsm.com

Resources and References

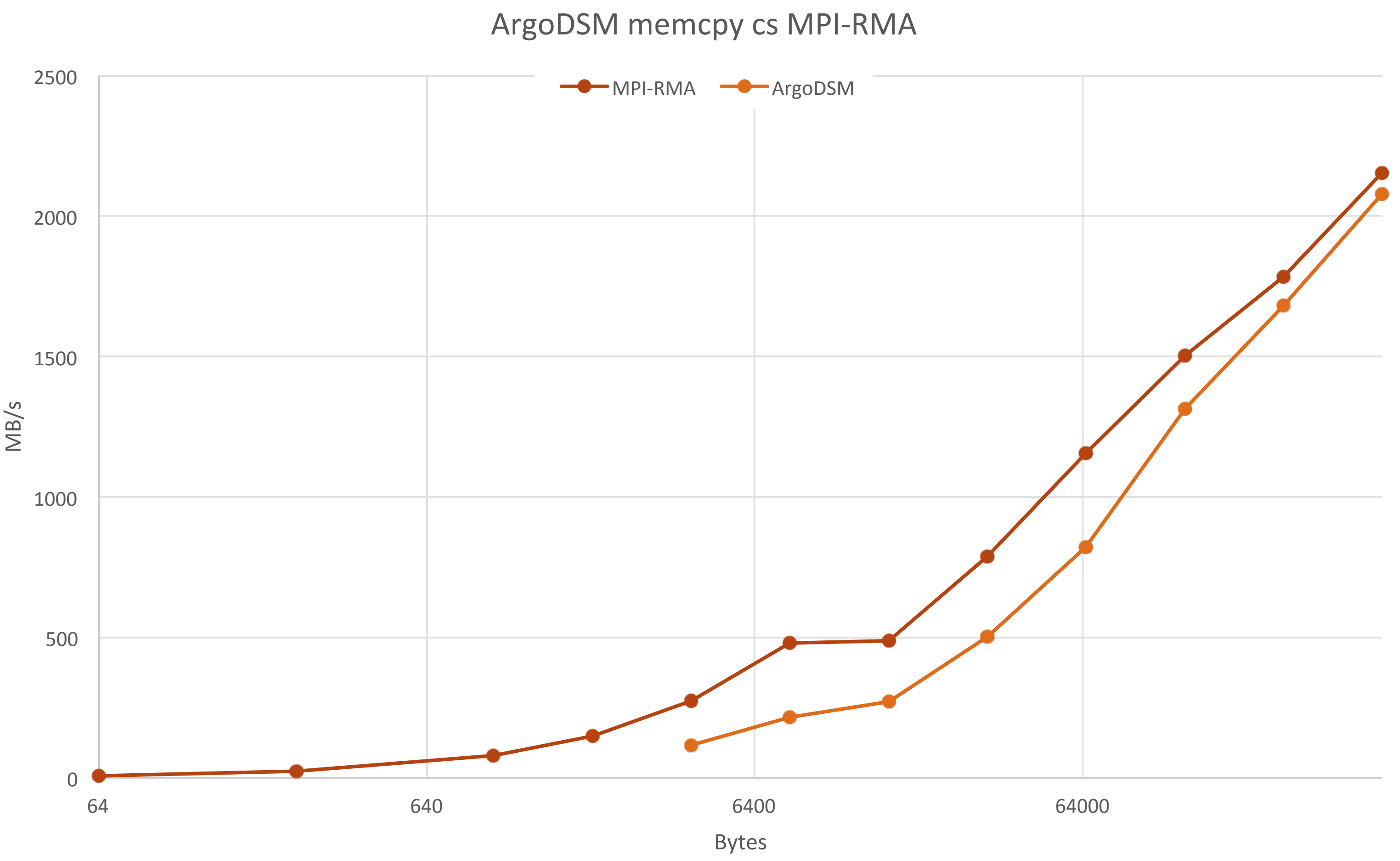
- ArgoDSM
    - Project site (and the HPDC'15 paper)
        - http://www.it.uu.se/research/project/argo
    - *http://www.argodsm.com*
    - https://dl.acm.org/citation.cfm?id=2749250
- VIPS Coherence
    - https://dl.acm.org/citation.cfm?doid=2370816.2370853
    - http://ieeexplore.ieee.org/document/6398353/?reload=true
- Queue-Delegation Locking
    - http://www.it.uu.se/research/group/languages/software/qd_lock_lib
- StarPU
    - http://starpu.gforge.inria.fr

Extra Slides – Scaling ONLY MEMORY Python example

- Python-based spectral filter application from meteorology (very large data set using SciPy)
  - 8 Nodes for memory, one node for compute
  - Outperforms swap



Extra Slides - Bandwidth



Extra Slides – HW Specs

- Compute per node
  - Two Intel Xeon E5 2630 v4, 2.20Ghz (10 cores, 20 threads)
- Network
  - Infiniband FDR
    - Latency 0.7us
    - Bandwidth 56GB/s

Extra Slide - Eviction Policies

- Eviction Policy - Direct Mapped, Random, ’least recently mapped’=FIFO