ICPP 2016

# Declarative Tuning for Locality in Parallel Programs

**Sanjay Chatterjee**, Nick Vrvilo,
Zoran Budimlic, Kathleen Knobe,
Vivek Sarkar

Rice University
Habanero Extreme Scale Software
Research Project

RICE

http://habanero.rice.edu/

# Tuning Goal: A separation of concerns

Application problem → Domain Program → Optimized mapping to platform

The domain expert:
- Finance
- Gaming
- Chemistry

The tuning expert:
- Parallelism
- Locality
- Load balancing

Domain expert doesn't need to know a lot about parallelism

Tuning expert doesn't need to know a lot about the app

- Domain expert, but later
- Different person, different skills
- Automated: static analysis / autotuning

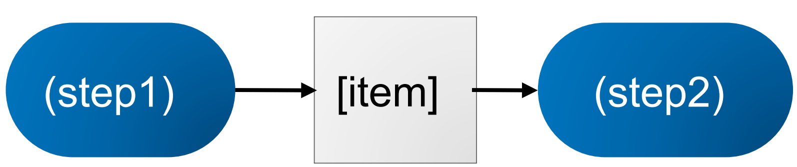
# The Concurrent Collections (CnC) Programming Model

- CnC is a *coordination* language, paired with a programming language
- Declarative *dependence*-based programming model
- CnC programming constructs:
  - Step collection
    - specifies computation tasks in the application
  - Item collection
    - specifies data sets for the application
  - Tag collection
    - specifies pure control flow in the application

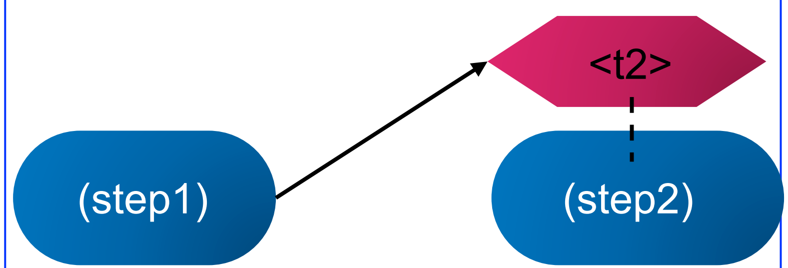
# CnC Dependence Relations

- Producer must execute before consumer
- Controller must execute before controllee

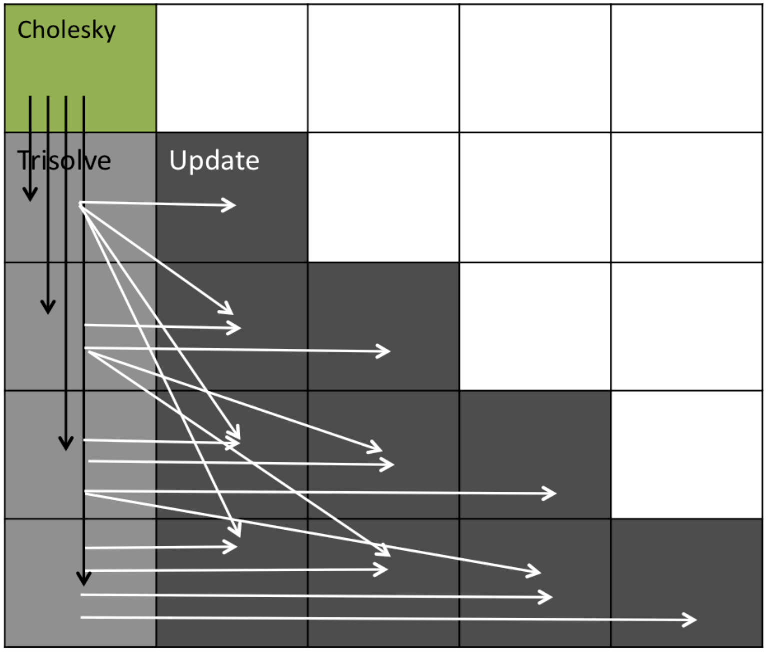
Producer - consumer




Controller - controllee



# CnC Domain Spec




Tiled Cholesky Factorization

(\$init:) → (cholesky:0…N)

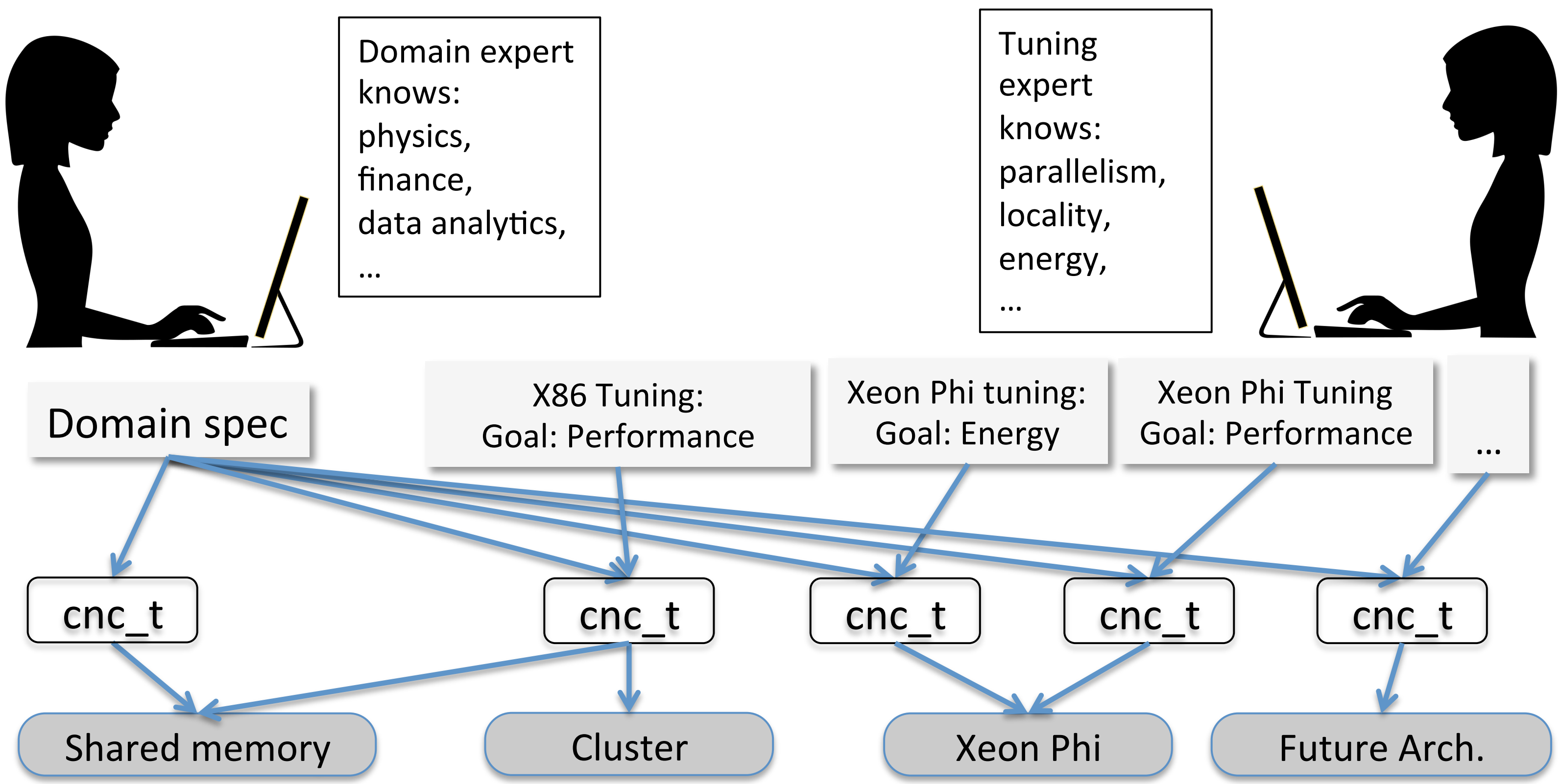
(cholesky:iter) → (trisolve:k+1…N,iter)
(cholesky:iter) ← [array:iter,iter,iter]
(cholesky:iter) → [array:iter,iter,iter+1]

(trisolve:row,iter) → (update:iter+1…row+1,row,iter)
(trisolve:row,iter) ← [array:row,iter,iter]
(trisolve:row,iter) ← [array:iter,iter,iter+1]
(trisolve:row,iter) → [array:row,iter,iter+1]

(update:col,row,iter) ← [array: col, row, iter ]
(update:col,row,iter) ← [array:col,iter,iter+1]
(update:col,row,iter) ← [array:row,iter,iter+1]
(update:col,row,iter) → [array:col,row,iter+1]

(Domain Spec)

# CnC Tuning Spec



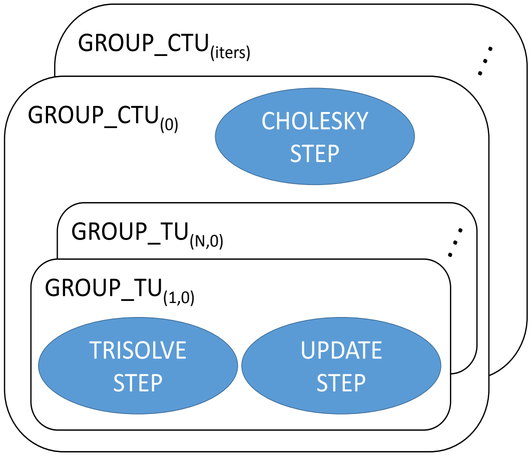
# CnC Tuning Spec for Locality

- Hierarchical Distribution Functions
  - Declarative step/item placement
  - Enables spatial locality

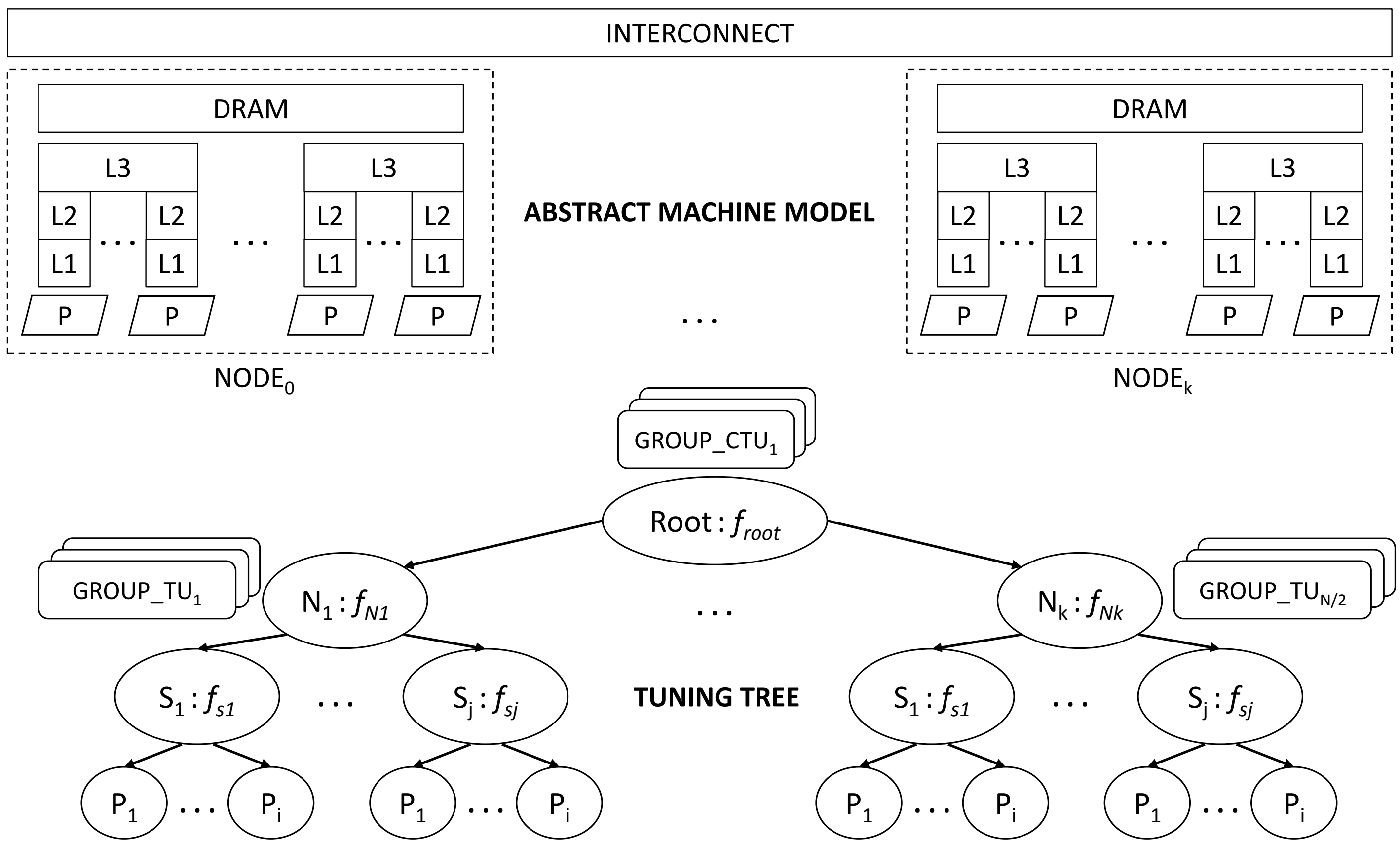
[array:col,row,iter]: { iter % $RANKS };

- Hierarchical Affinity Groups
  - Declarative step/item grouping
  - Enables temporal locality

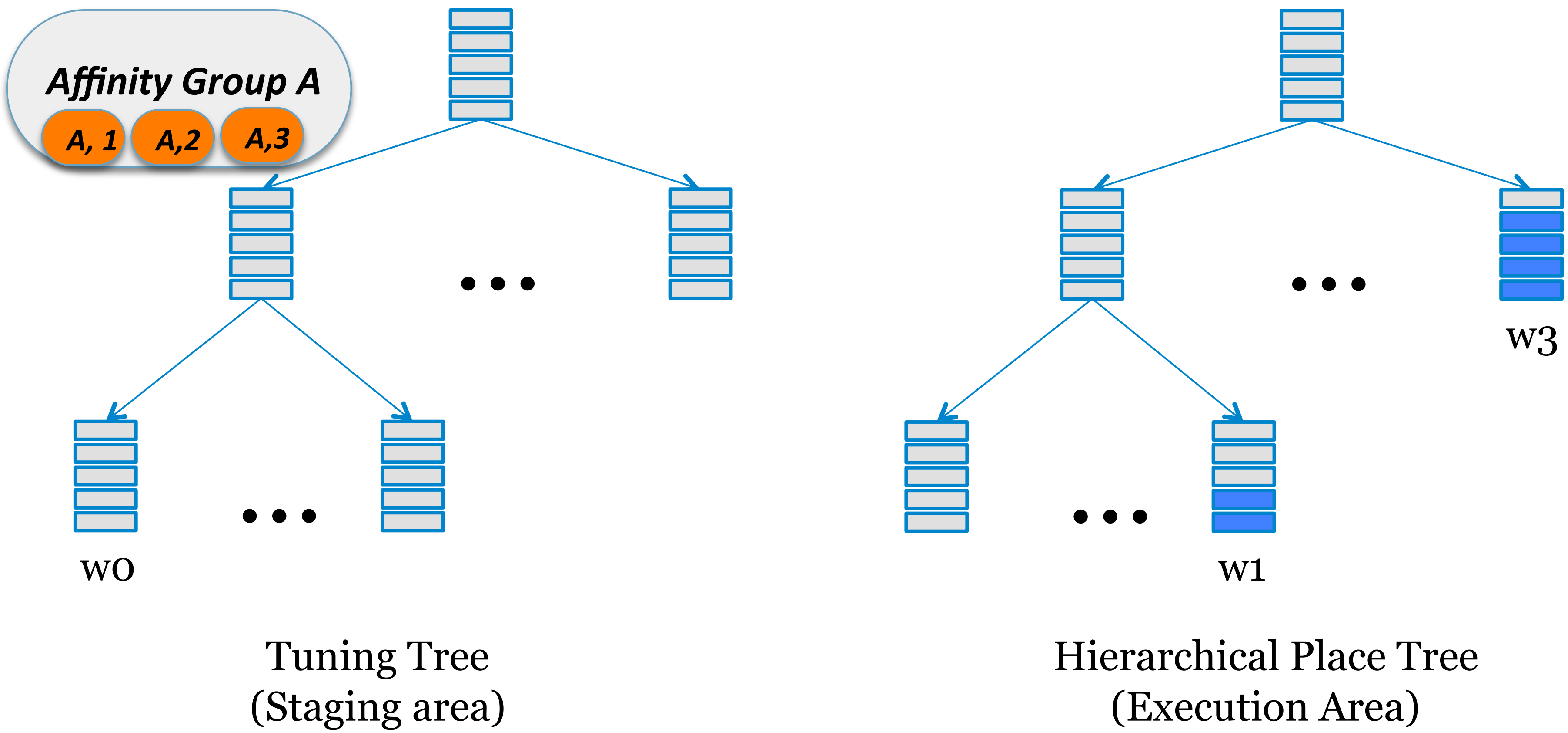
(CTU:iter) →
    (cholesky:iter), (TU:0...iter, iter);
(TU:row,iter) →
    (trisolve:row,iter), (update:0...iter, row, iter);



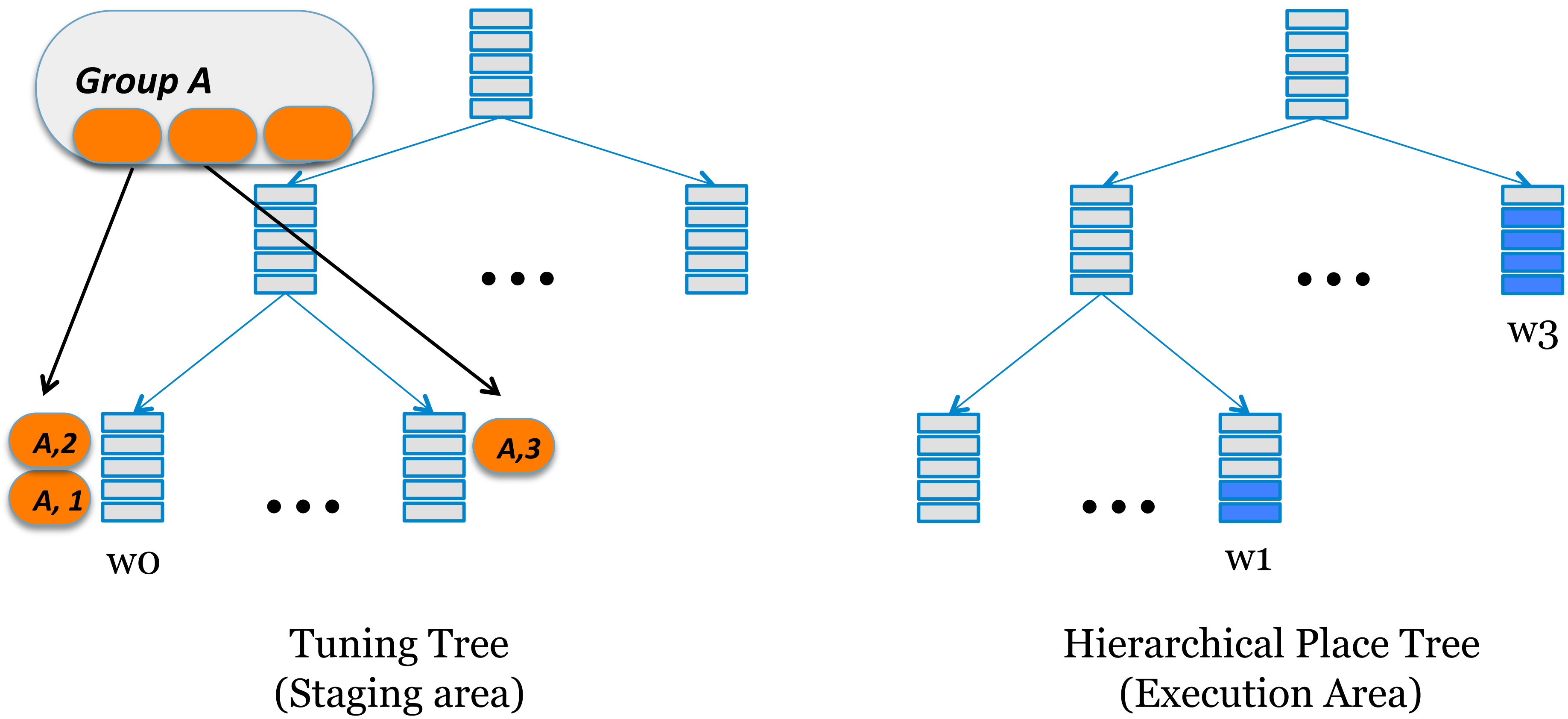
# CnC Tuning Abstract Machine Model



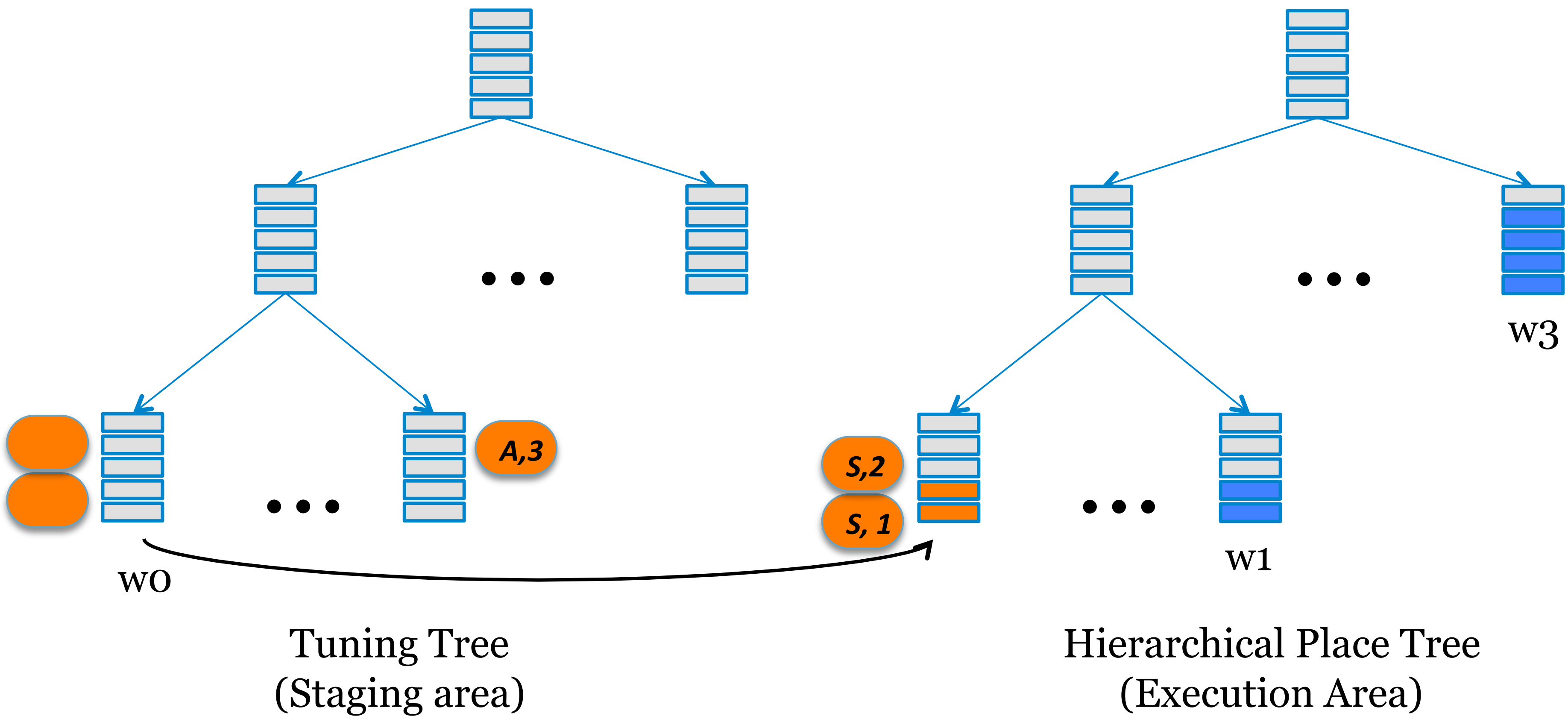
# CnC Tuning Execution Model: Affinity Grouping



# CnC Tuning Execution Model: Affinity Group Distribution

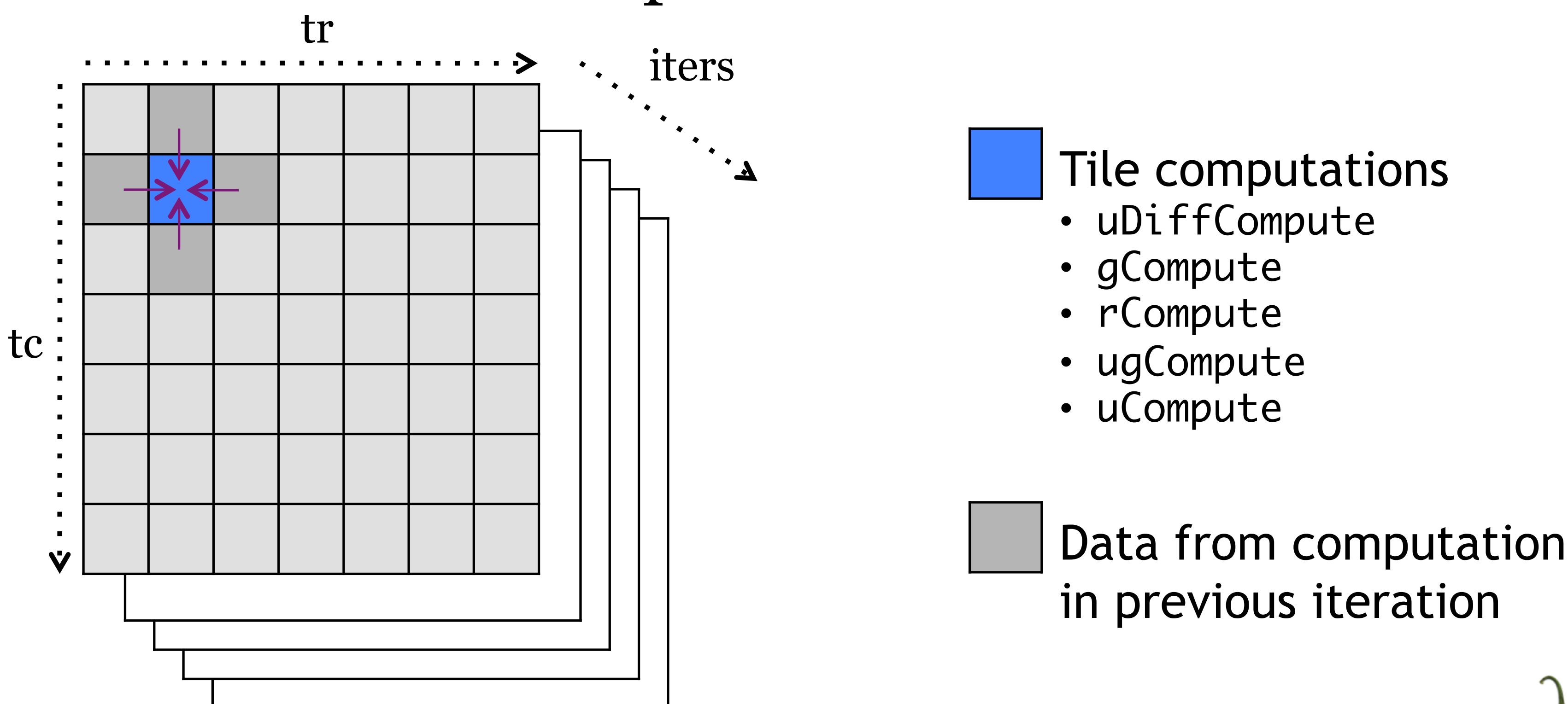


# CnC Tuning Execution Model: Runtime Step Execution

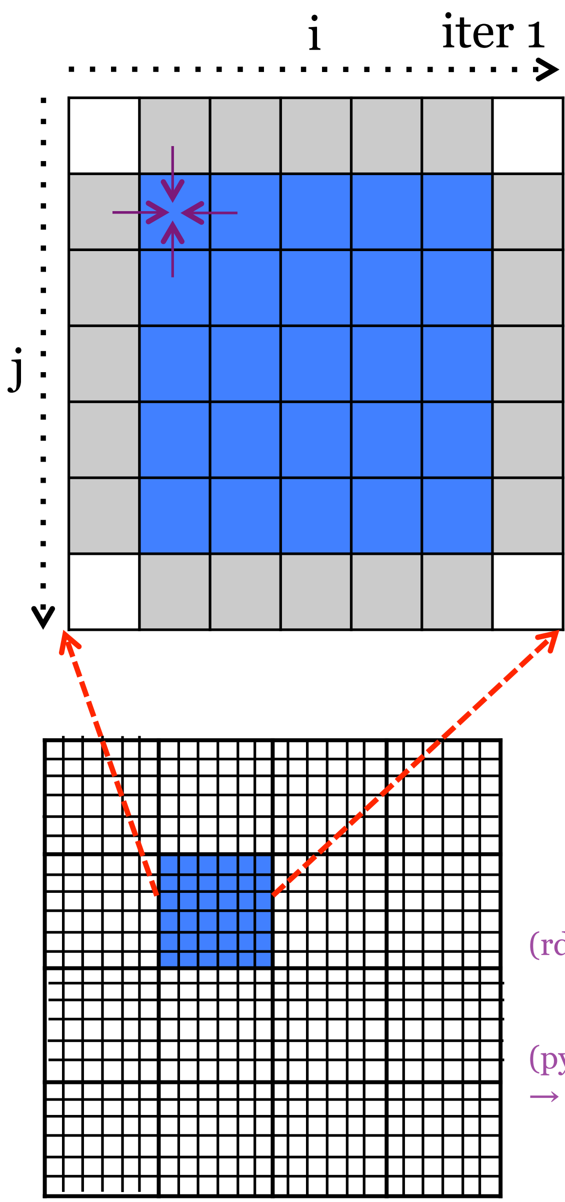


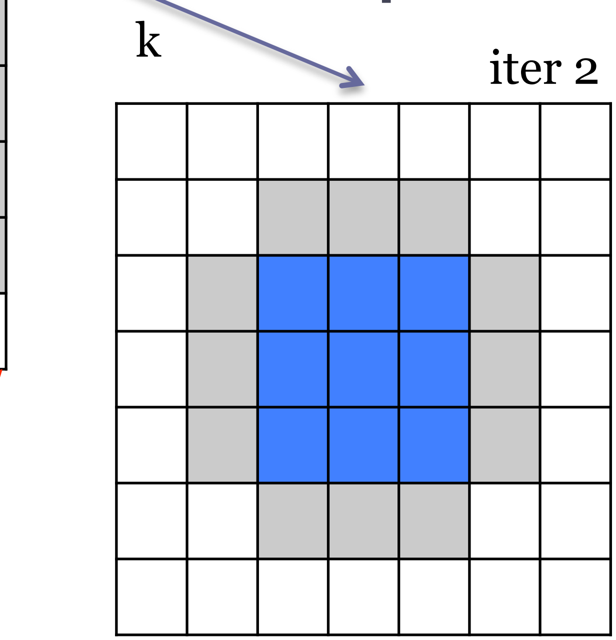
# CnC Tuning Example: Rician Denoising

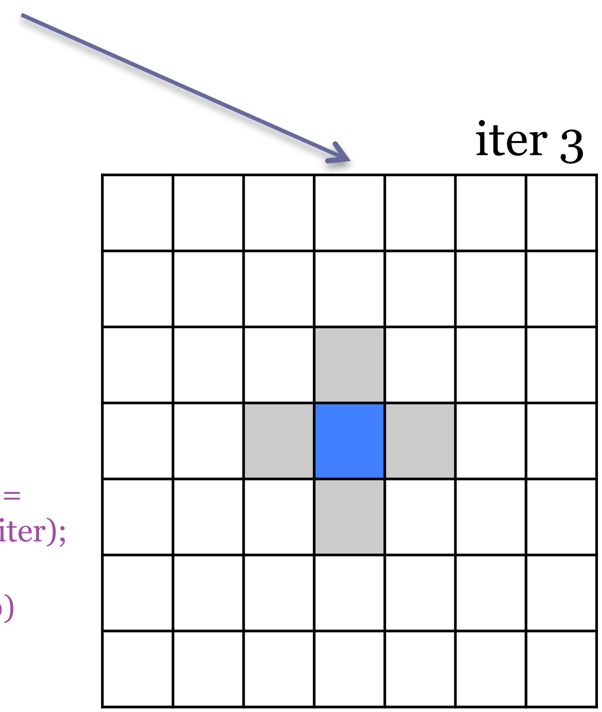
- Iterative stencil computation



# Affinity Grouping for Temporal Locality



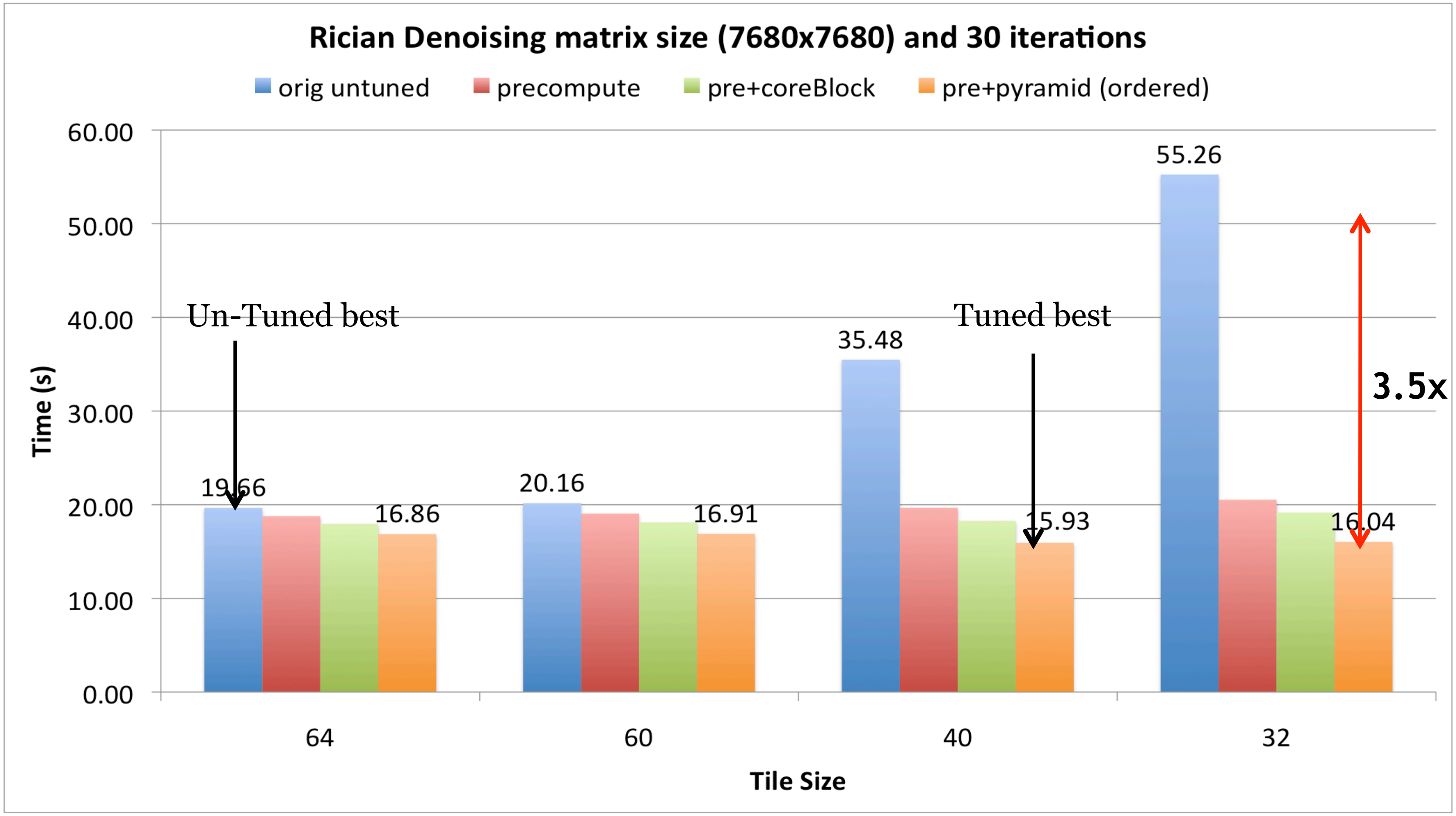




(rdTunedStep: iter, row, col, iter0, row0, col0 =
    rdStep: row0 + row, col0 + col, iter0 + iter);

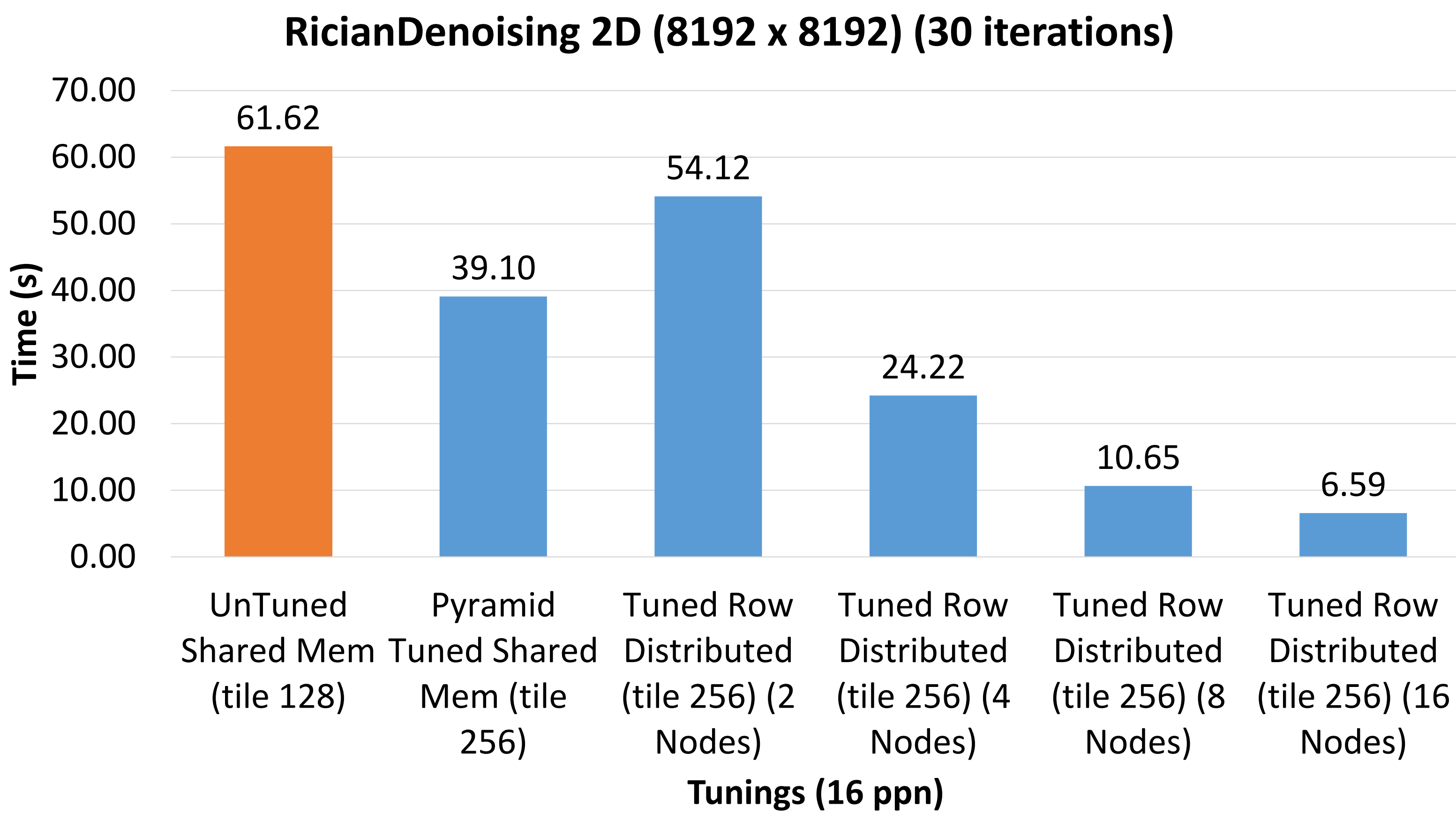
(pyramid: row0, nRow, col0, nCol, nIter, iter0)
→ (rdTunedStep:
    iter @ 0...nIter, iter...(nRow - iter),
    iter ... (nCol - iter),
    iter0, row0, col0)

# Tuning Rician Denoising for Multi-Core




- Dual-socket 12-core Intel Westmere
- Up to 3.5x improvement on specific tile size
- Overall improvement of 19% over best optimal tiled version

# Tuning Rician Denoising for Distributed Systems



# Conclusions

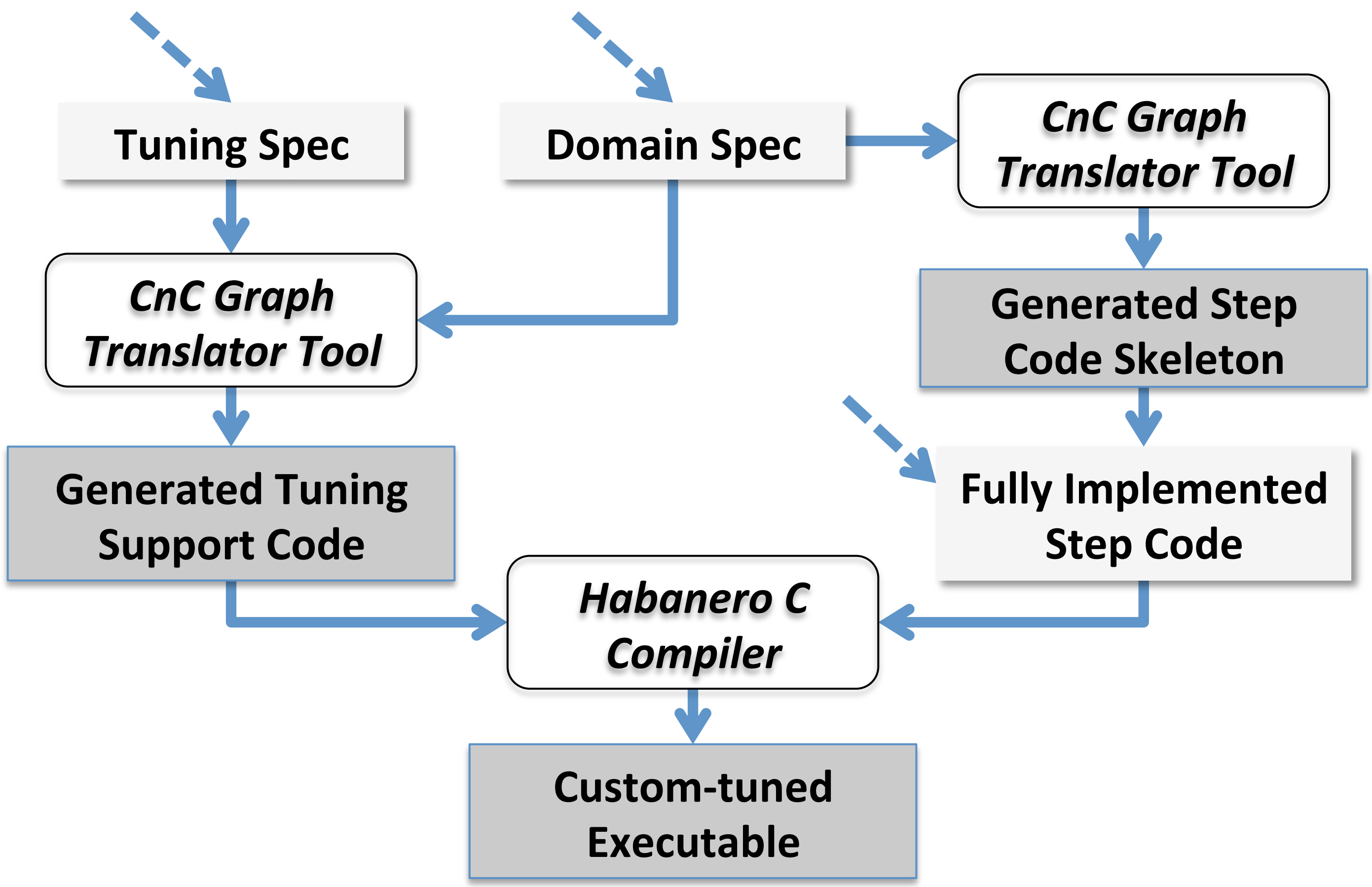
- The CnC tuning framework helps allows locality tuning of applications separately from its domain specification
- Declarative tuning specification simplifies exploration of optimal mapping of application for different targets
- Habanero task-based dynamic runtime system enables balanced execution for optimized locality and parallelism

# Source Code:

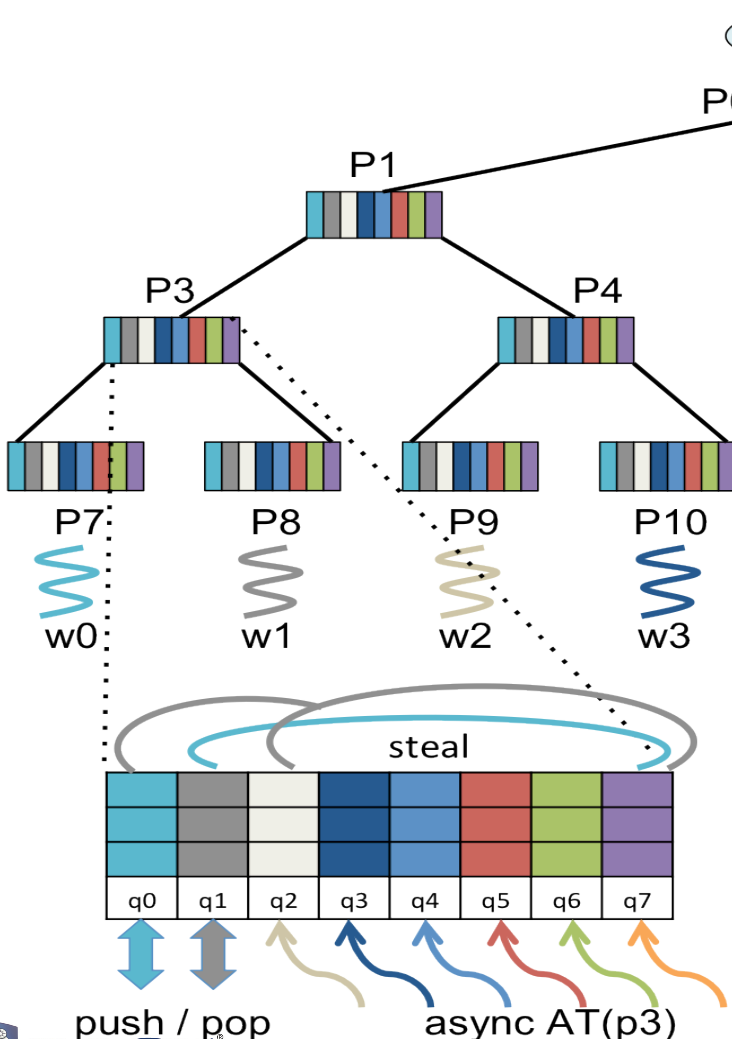
- CnC Tuning:
  - https://github.com/habanero-rice/cnc-ocr/tree/icpp2016-tuned-cnc
- Habanero Project:
  - https://github.com/habanero-rice

# Backup

# CnC Tuning Workflow



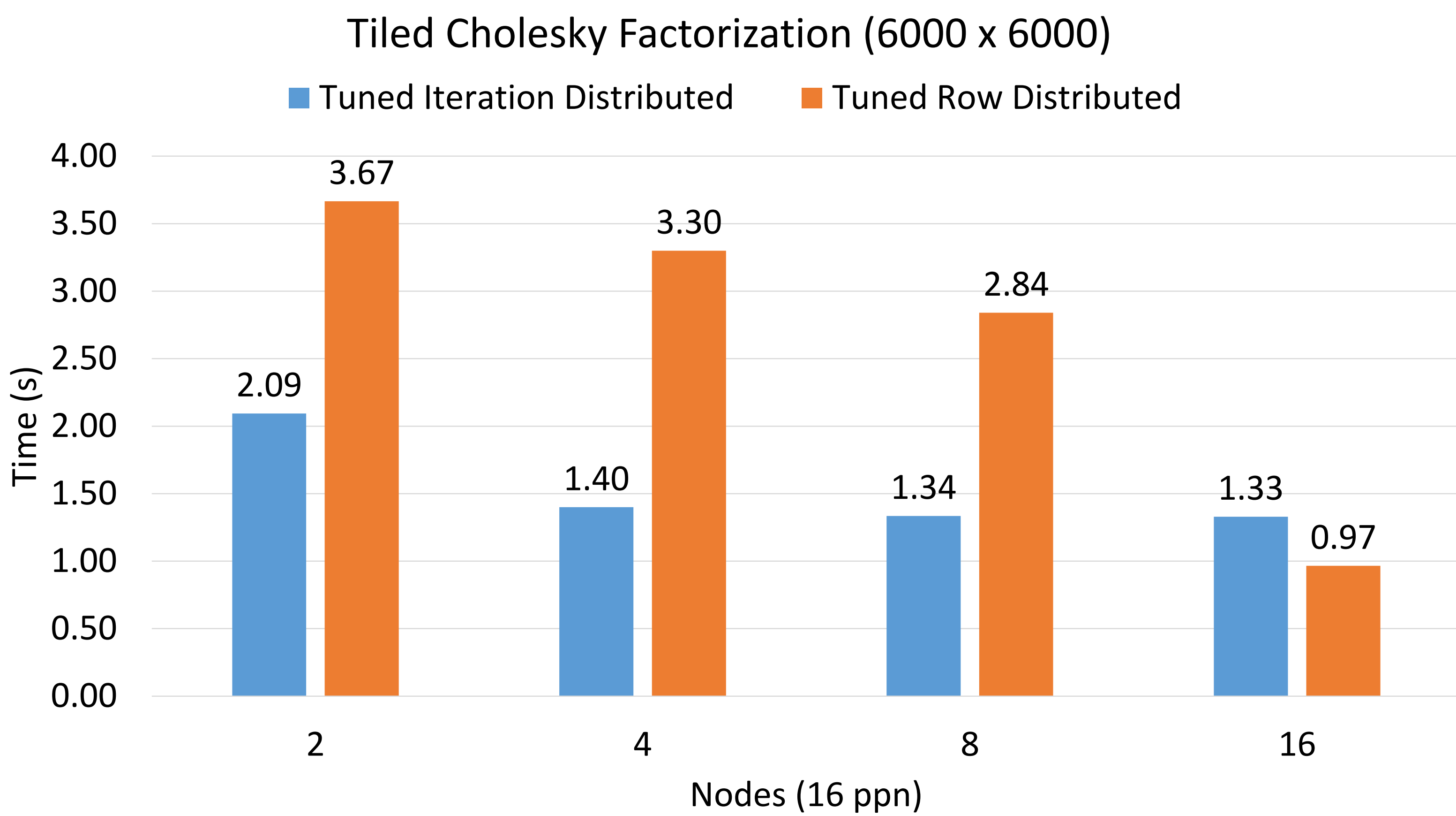
# Runtime Locality Control: Hierarchical Place Trees (HPT)




Example: Intel Xeon Dual Quad Core
-2 sockets with shared L3
-2 shared L2 per socket

- Each place has one queue per worker
- Ensures non-synchronized push and pop
- Workers attach to (own) leaf places
- Any worker can push a task at any place
- Pop / steal access permitted to subtree workers
- Workers traverse path from leaf to root
- Tries to pop, then steal, at every place
- After successful pop / steal worker returns to leaf
- Worker threads are bound to cores

# Tuning Tiled Cholesky Factorization for Distributed Systems



# Tuning Smith-Waterman for Distributed Systems