CnC for high performance computing

Kath Knobe

Intel SSG

Thanks

- Frank Schlimbach – Intel
- Vivek Sarkar and his group – Rice University
- DARPA UHPC (Runnemede)
- DOE X-Stack (Trilacka Glacier)

Outline

- Intro
- Checkpoint/restart
- Tuning language
- Adaptive computing

Productivity
Analyzability
Protability
Adaptivity

The big idea

Don’t specify what operations run in parallel

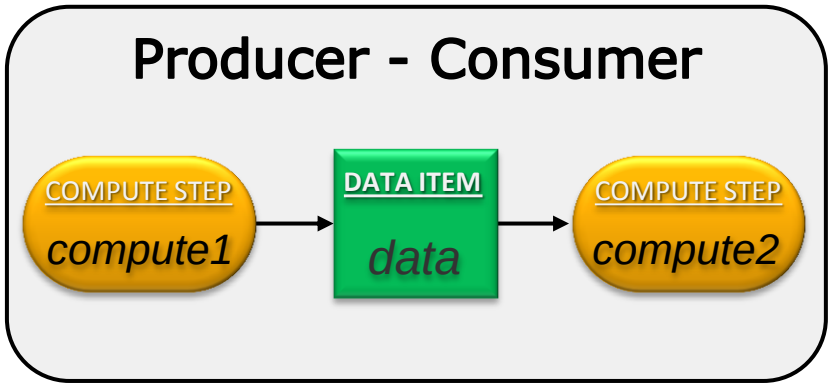
Difficult and depends on target

Specify only required orderings

Easy, known and depends only on application

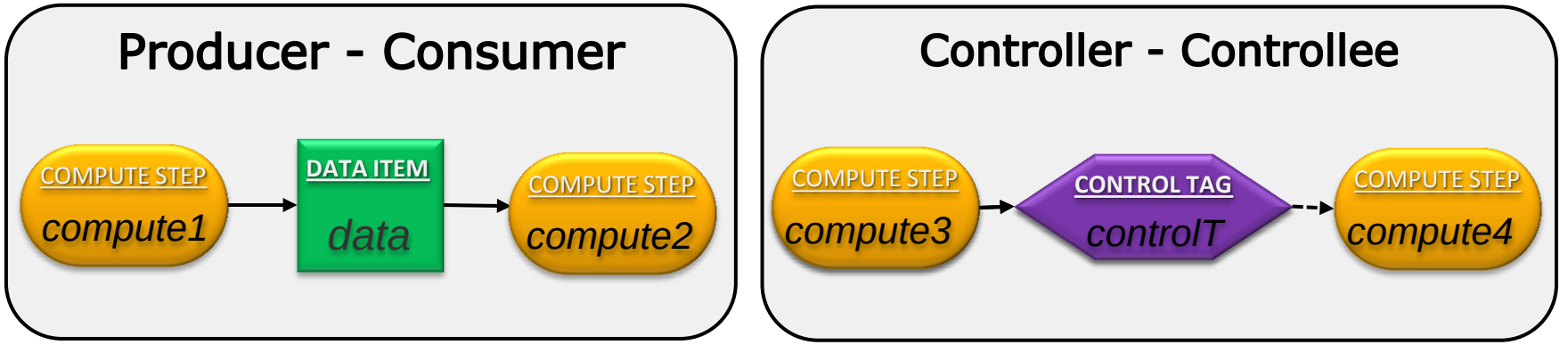
Exactly 2 constraints on parallelism

- Producer must execute before consumer

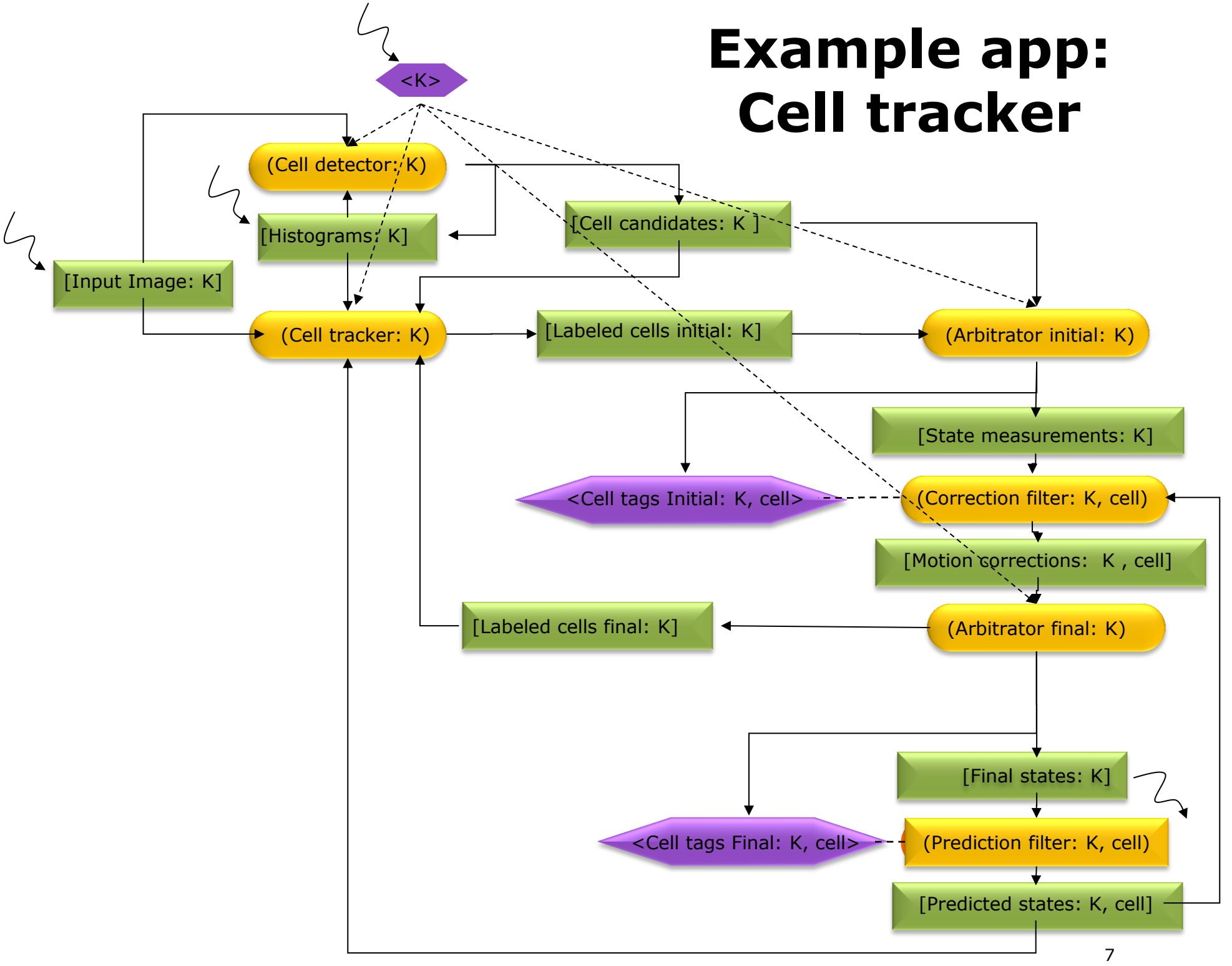


Exactly 2 constraints on parallelism

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



Example app: Cell tracker



Representations

Graphical

textual

compute1(j, k) -> data[j, k] -> compute2(j, k)

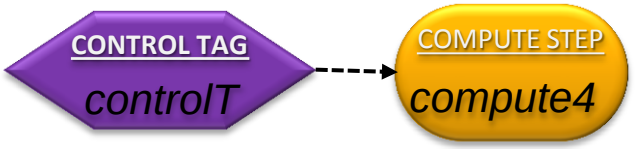
compute3(row, col) -> controlT<row, col> -> compute2(row, col)

API

Language dependent – one API call per graph edge

How to think about control tags

- Control tag: just an identifier in the form of a tuple
- Examples
  - For matrix computation
    - matrixTags<row, col, iter>
  - Set of images in video
    - imageTags<imageID>
  - Set of images in video that contain faces
    - faceTags<imageID>



- Very much like an iteration loop and its body
  - For each instance in the collection controlT the associated instance of compute 4 will execute with access to its tag value.

Pair CnC with computation language
Sample step code in C++

```
// Performs symmetric rank-k update of the submatrix.
// Input to this step is the given submatrix and the output of the previous step.
int update_step::execute( const triple & t, cholesky_context & c ) const
{
    const                                                    t int i = t[2];
    asser

    tile_type A    ck, L1_block, L2_block;
    c.tiles.get( triple( k, j, i ), A_block ); // Get the input tile.

    if( i==j ){   // Diagonal tile.
        c.tiles.get( triple( k+1, i, k ), L1_block ); // both the tiles are the same.
    } else{   // Non-diagonal tile.
        c.tiles.get( triple( k+1, i, k ), L2_block ); // Get the first tile.
        c.tiles.get( triple( k+1, j, k ), L1_block ); // Get the second tile.
    }

    // A_block is a copy, a local variable, so we can overwrite at will
    for( int j_b = 0; j_b < b; j_b++ ) {
        for( int k_b = 0; k_b < b; k_b++ ) {
        }
    }

    c.tiles.put( triple( k+1, j, i, A_block );  // Write the output at the next iteration.
    return CnC::CNC_Success;
}
```

get reads in an instance of a data item

put writes an instance of a data item

Status

Intel
- C++

Rice
- Java, C, Scala, Python, Babel, …

Indiana
- Haskell, …

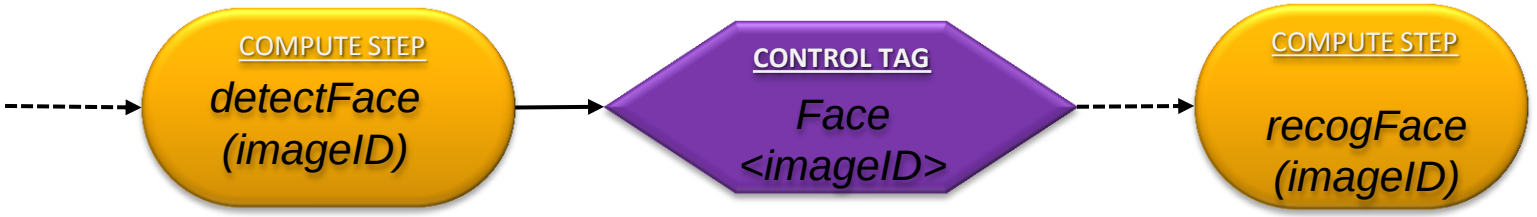
Participates in
- XStack Traleika Glacier
- UPHC Runnemede

As part of X-Stack
- Will have CnC on Open Community Runtime (OCR)

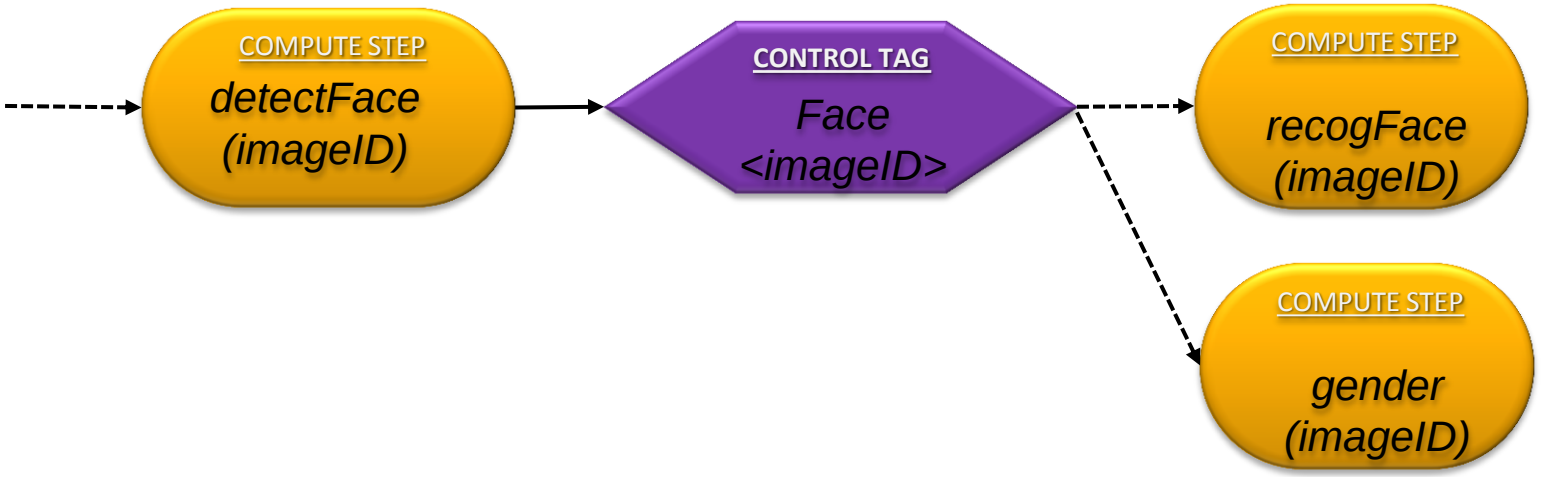
Existing support
- Distributed memory
- Heterogeneous platforms
  - CPU/GPU/FPGA
  - CPU/MIC

Some of the ideas here
- Exist
- Are in design
- Are in implementation

Isolation / mediation

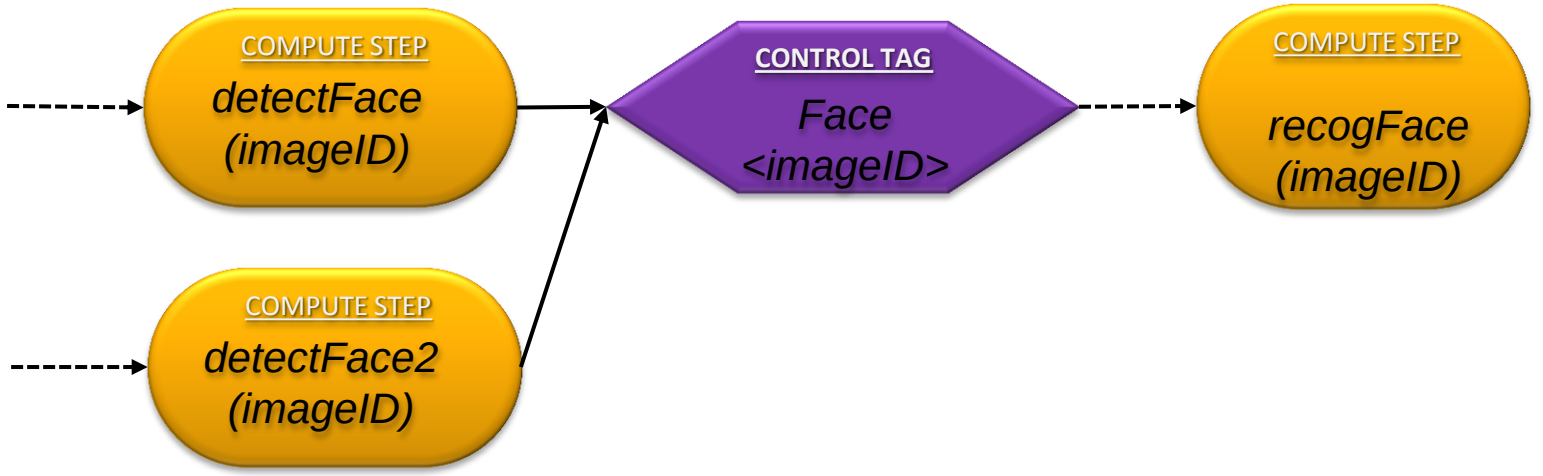


Isolation / mediation



Controller: detectFace<> doesn't need to know

Isolation / mediation



Controllee: recogFace<> doesn't need to know

The Contract

- If the programmer guarantees
  - Steps are atomic

    Get inputs, compute, put outputs
  - Steps have no side effects
  - Item obey dynamic single assignment (DSA) rule:

    Each given item name & tag is associated with a unique value (no overwriting)
- Then CnC guarantees
  - Determinism
- But

  Can do what you want. CnC will handle scheduling.

The Contract

- If the programmer guarantees
  - Steps are atomic

    Get inputs, compute, put outputs
  - Steps have no side effects
  - Item obey dynamic single assignment (DSA) rule:

    Each given item name & tag is associated with a unique value (no overwriting)
- Then CnC guarantees
  - Determinism
- But
  - Can do what you want. CnC will handle scheduling.
- **The rest of the talk assumes the contract**

A way to specify coordination of parts

Independent of:

- The computation language within the step
  - existing: C, C++, Java, Scala, Haskell, Python, Fortran (via Babel)…
- Parallelism within a computation
- Tuning
  - the distribution across the platform
  - the ordering in time (other than semantic ordering requirements)
- Type of runtime
  - static/dynamic choice of grain/distribution/schedule
- Underlying support
  - TBB, Qthreads, pthreads, Habenaro, MPI, Open Community Runtime (DOE), …
- The memory model
- The form in which the spec written
  - graphical interface
  - a textual representation of the graph
  - an API describing the graph.

A word about migration

- Can use your current computation is C, C++, Fortran, Java, …
  - Package it up a bit differently
- Bottom-up
  - start with a small piece of your app
  - create and execute a CnC graph
  - continue with the rest of your application
- Top-down
  - Break the whole app in to a small number of very large CnC chunks
  - As needed, break some of those into smaller pieces

High-level declarative specification
allows for a wide variety of runtime approaches

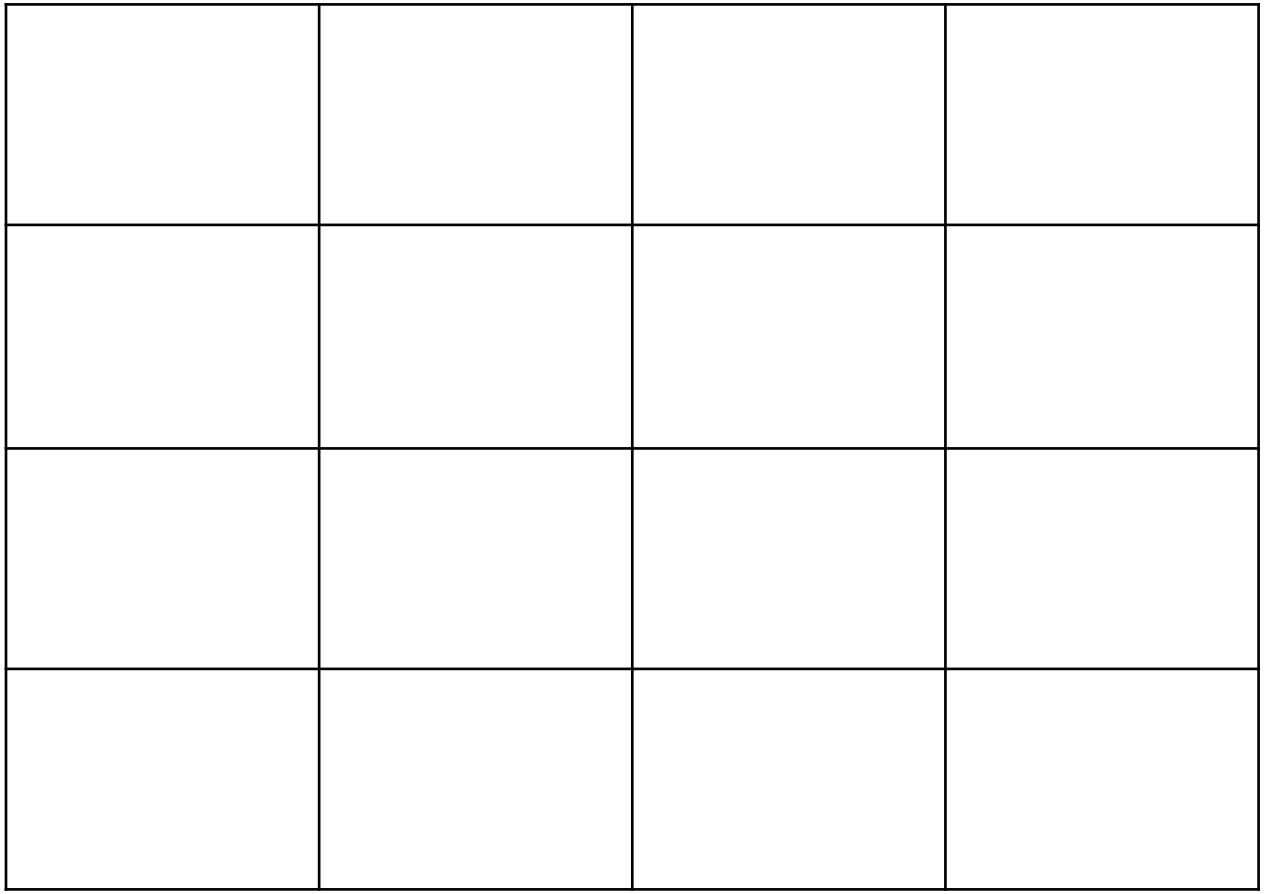
	grain	distribution	schedule
HP	static	static	static
HP	static	static	dynamic
Intel	static	dynamic	dynamic
Rice	static	dynamic	dynamic
Georgia Tech	dynamic	dynamic	dynamic

CnC / TStreams

Three example specs

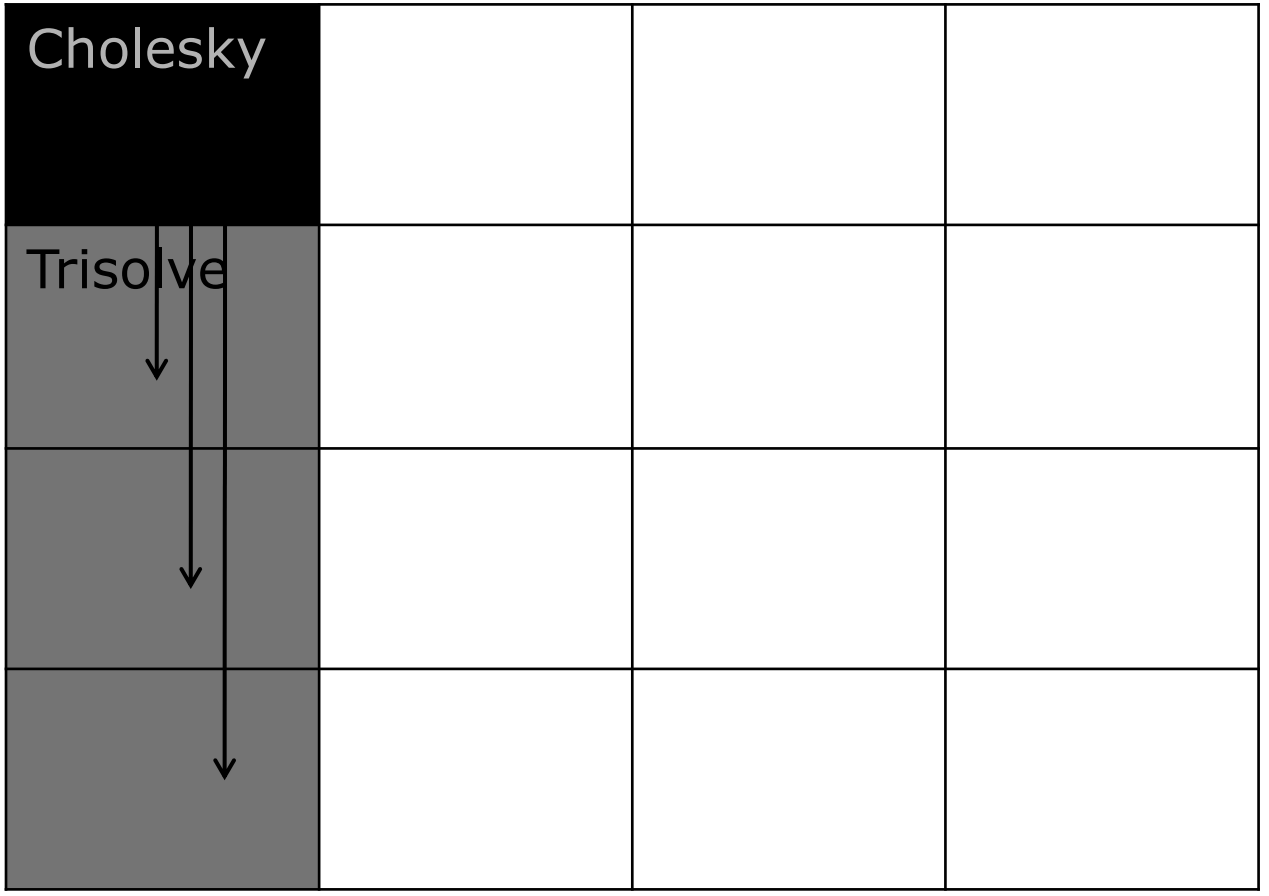
- Cholesky – all about data flow
- Face detection – all about control flow
- Iteration – about both control and data flow

Cholesky factorization

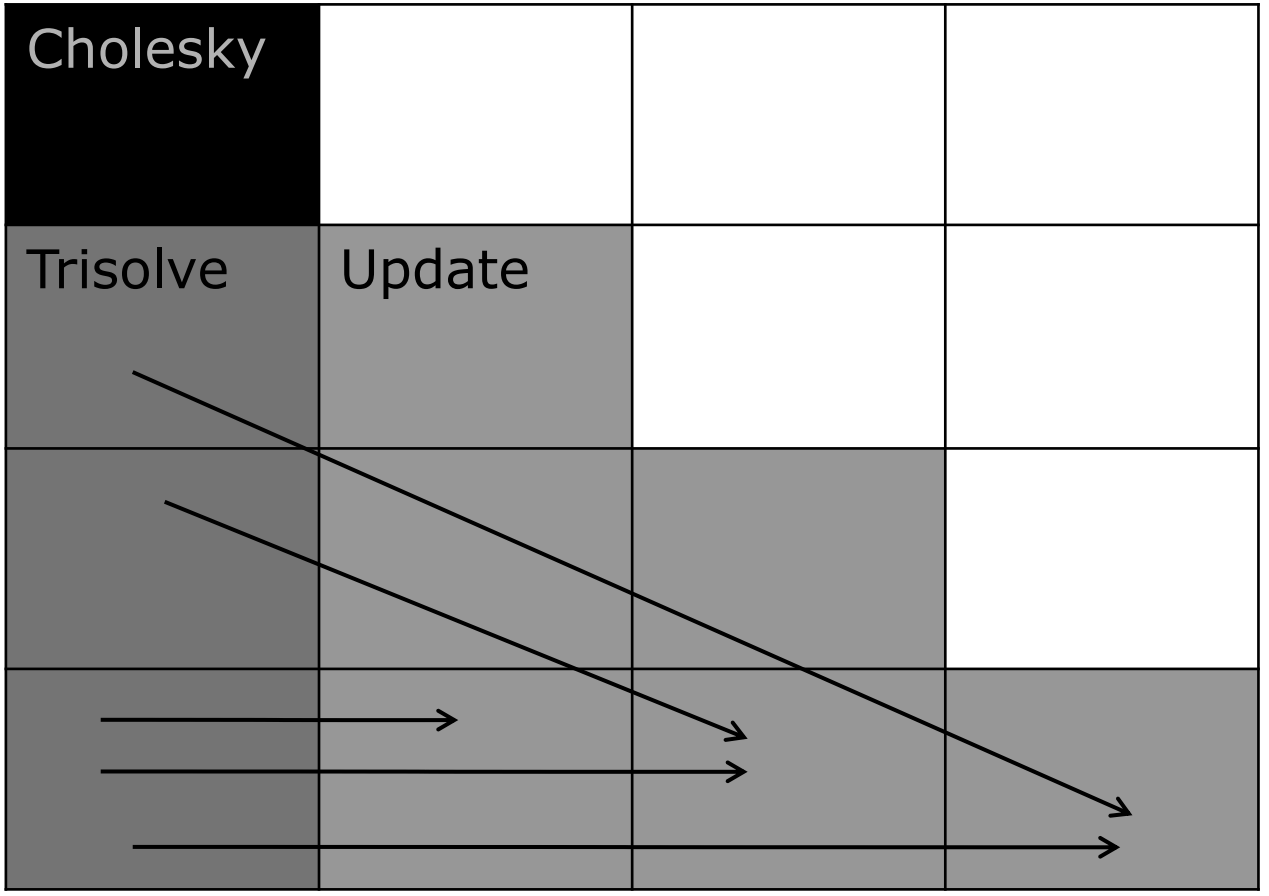


Cholesky factorization

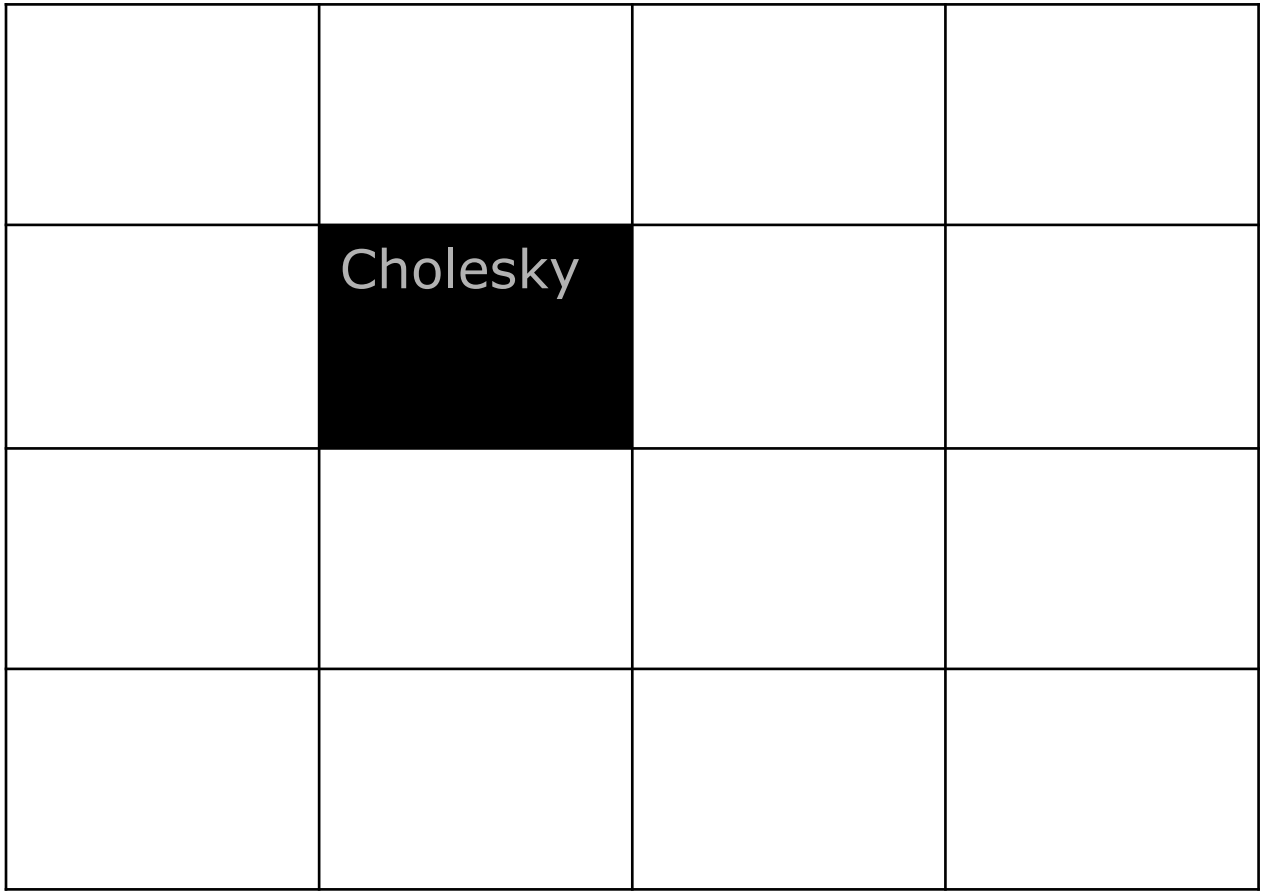
Cholesky factorization



Cholesky

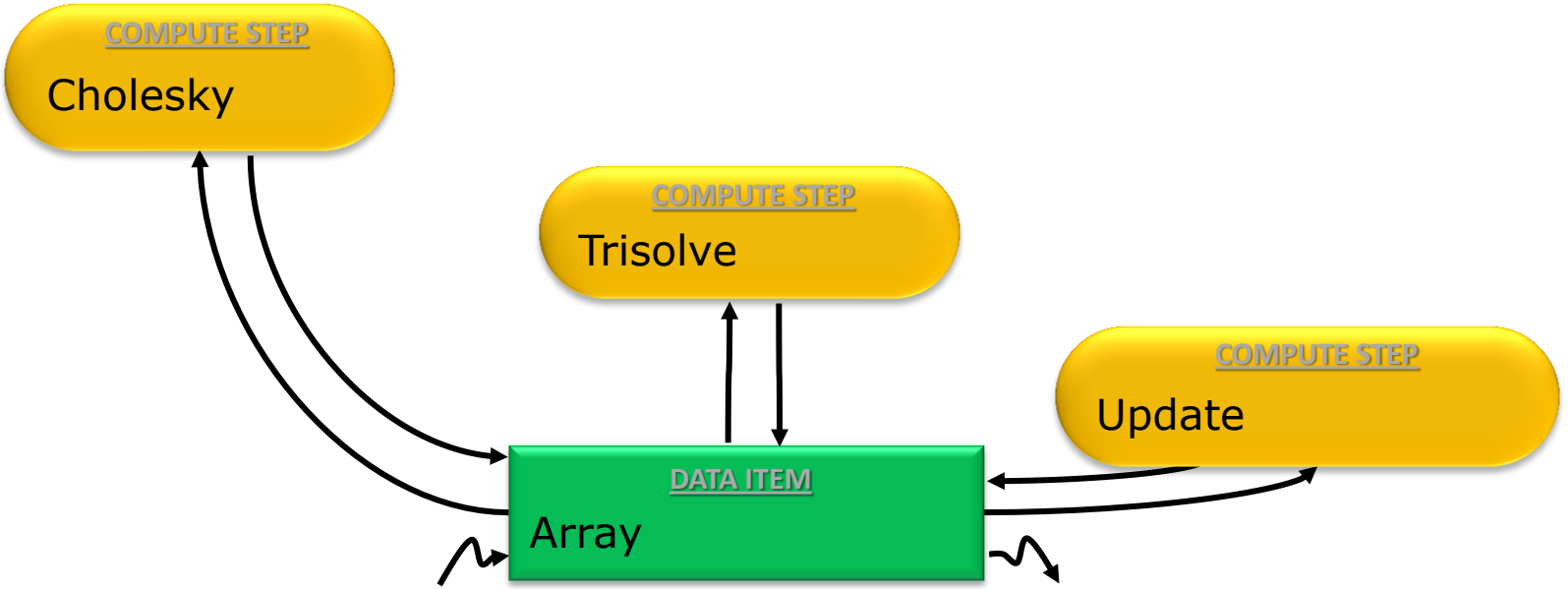


Cholesky factorization

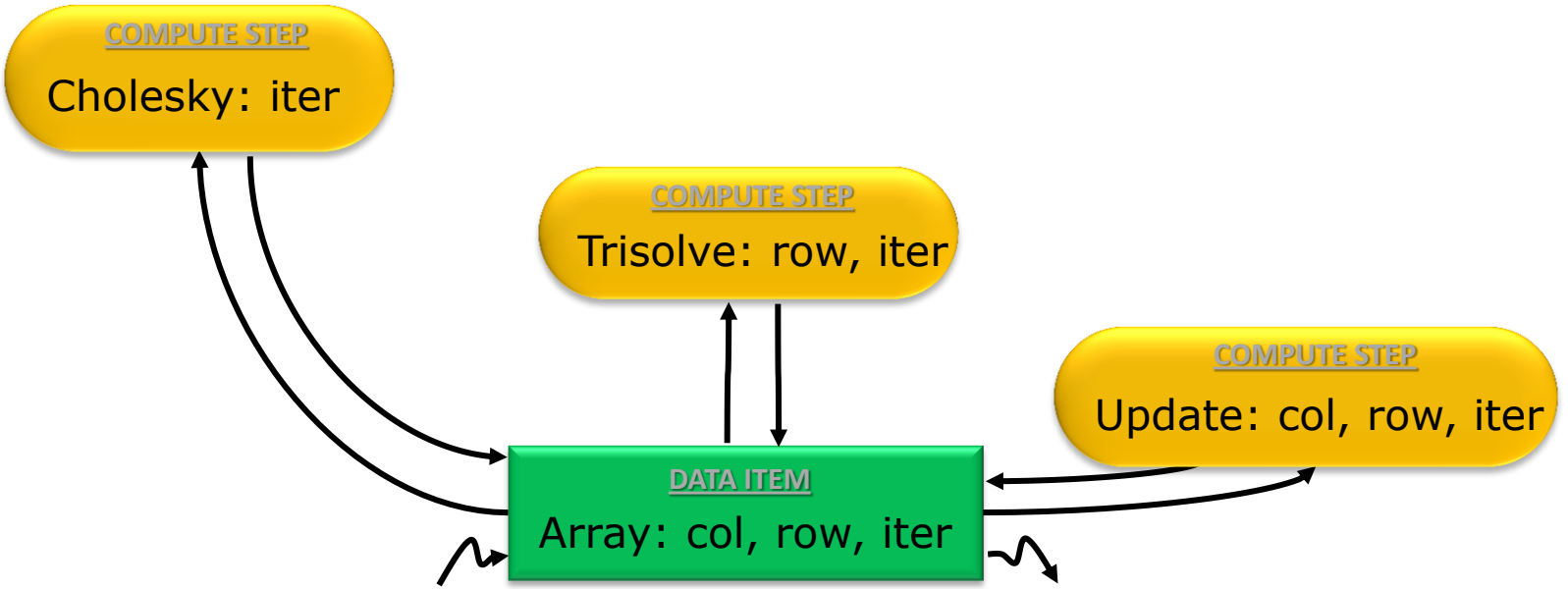


1. White board (how people think)

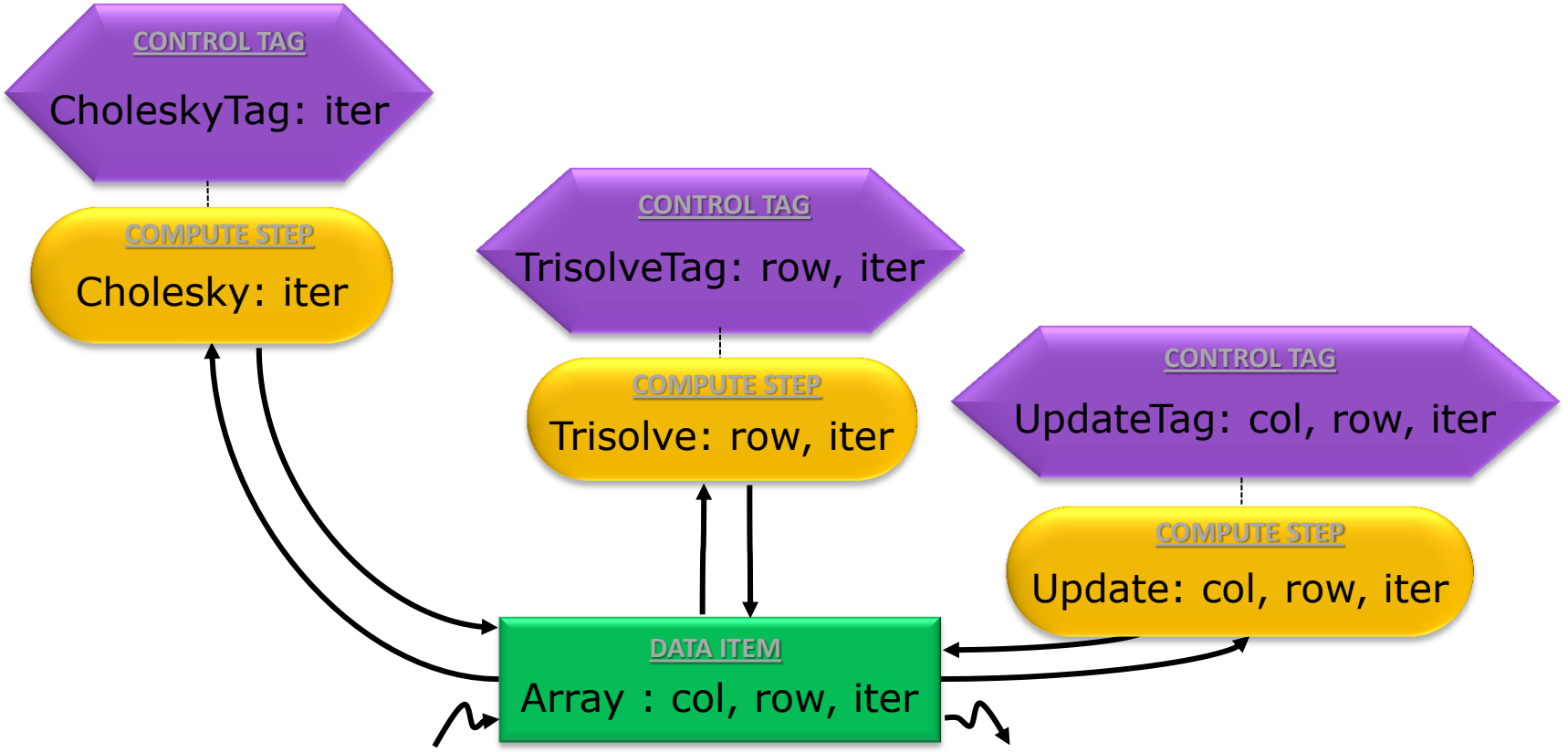
- computations
- data
- producer/consumer relations
- I/O



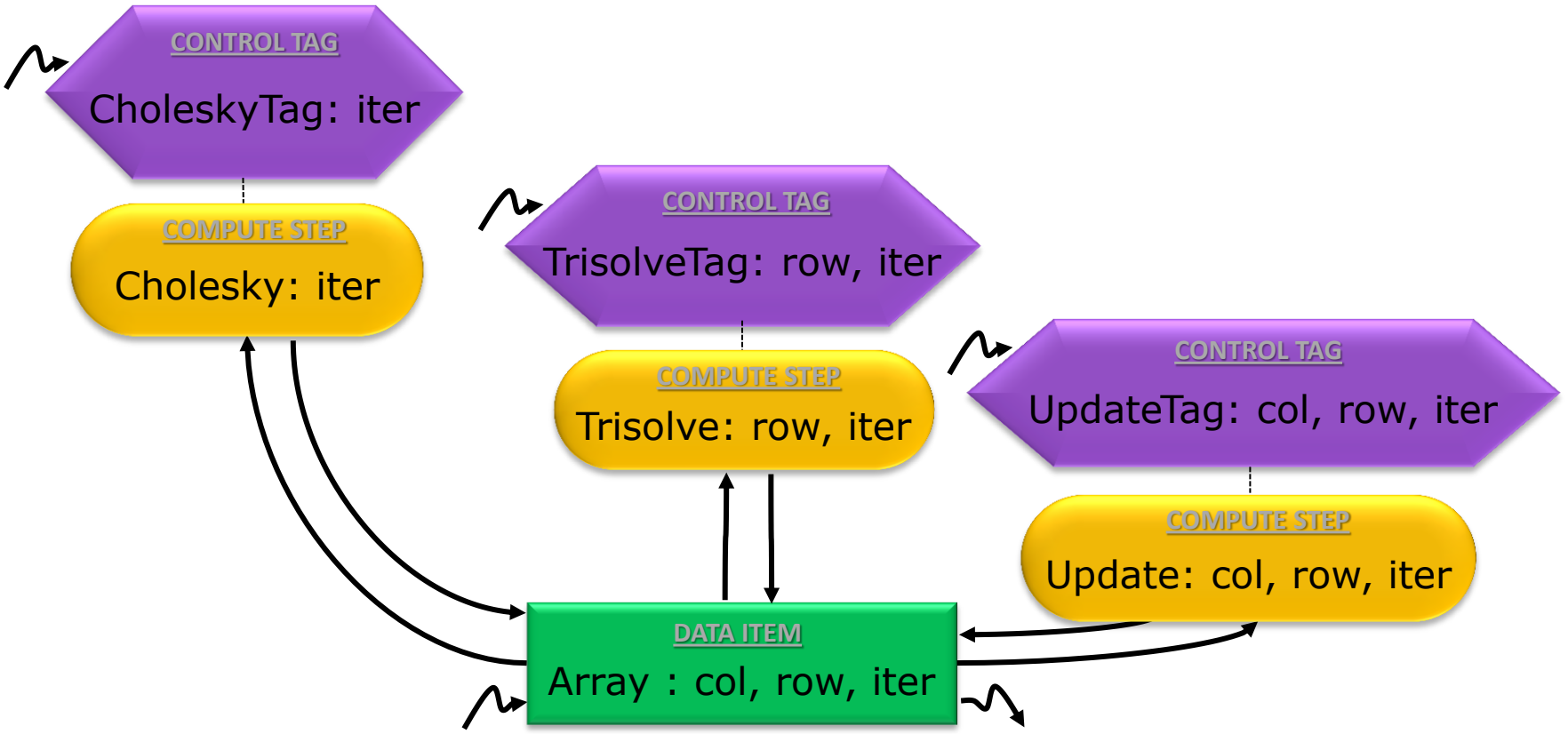
2: Distinguish among the instances



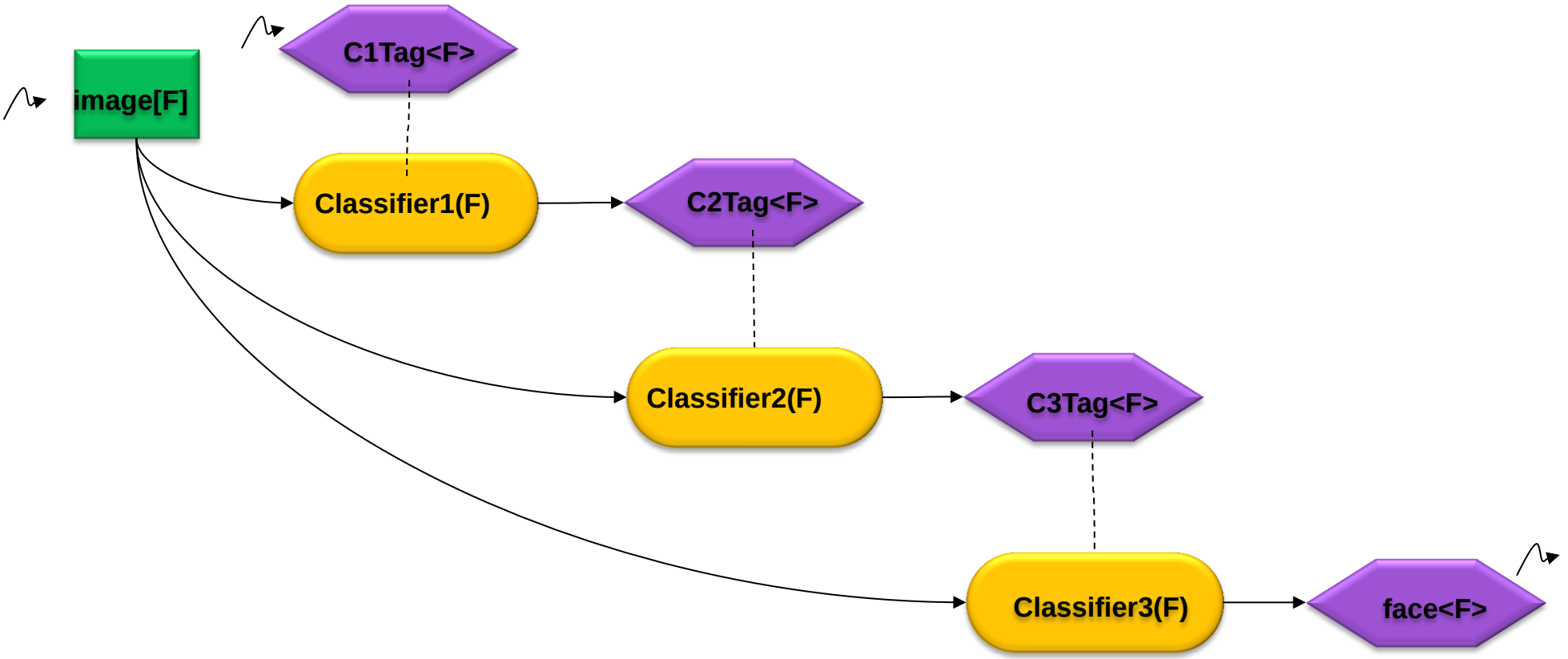
3: What are the control tag collections



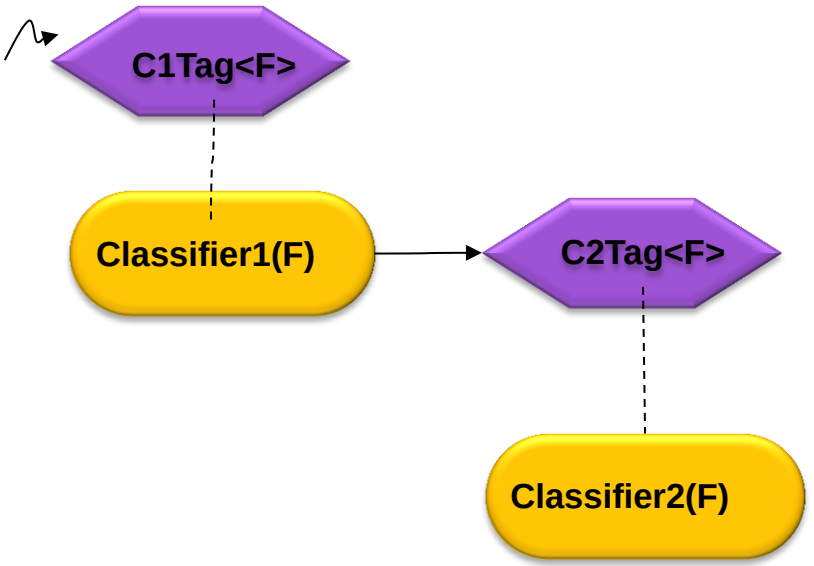
4: Who produces control



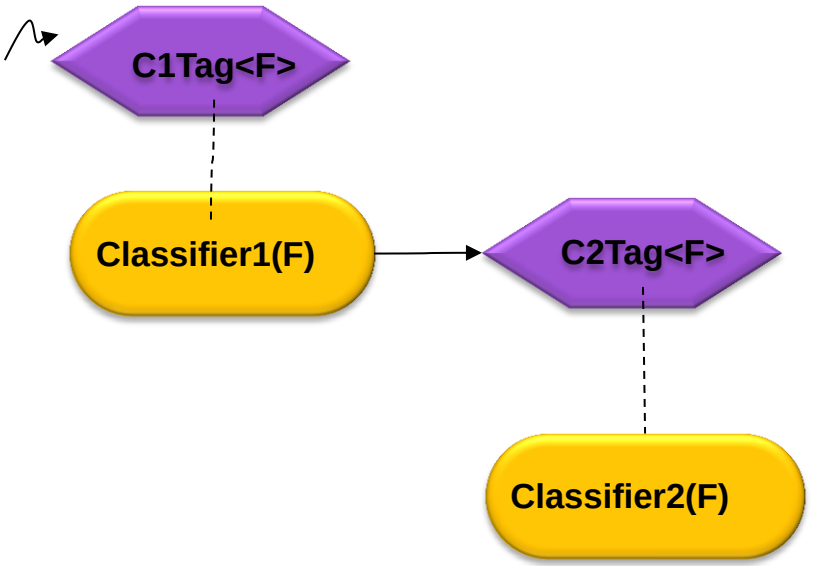
Sample user application face detection



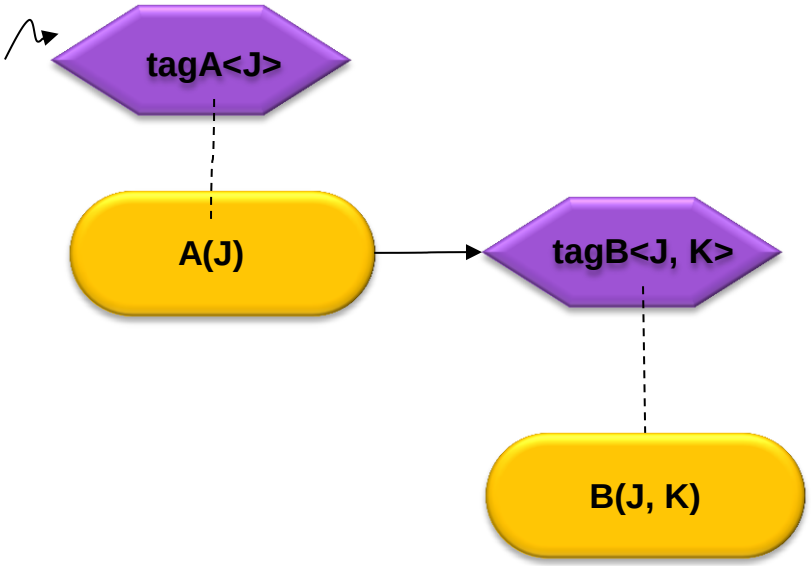
Conditional execution



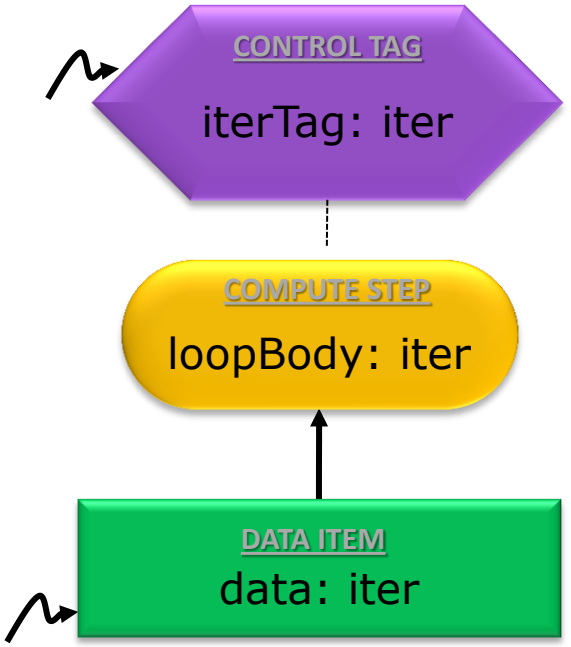
Conditional execution



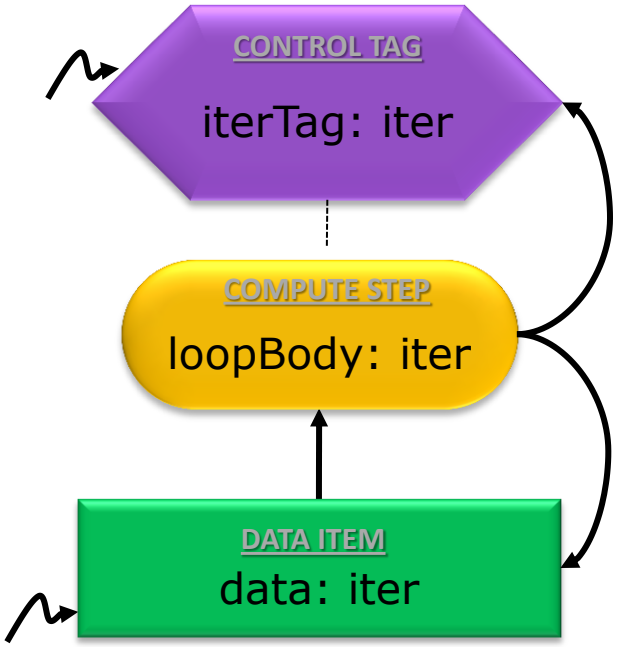
Increase iteration space



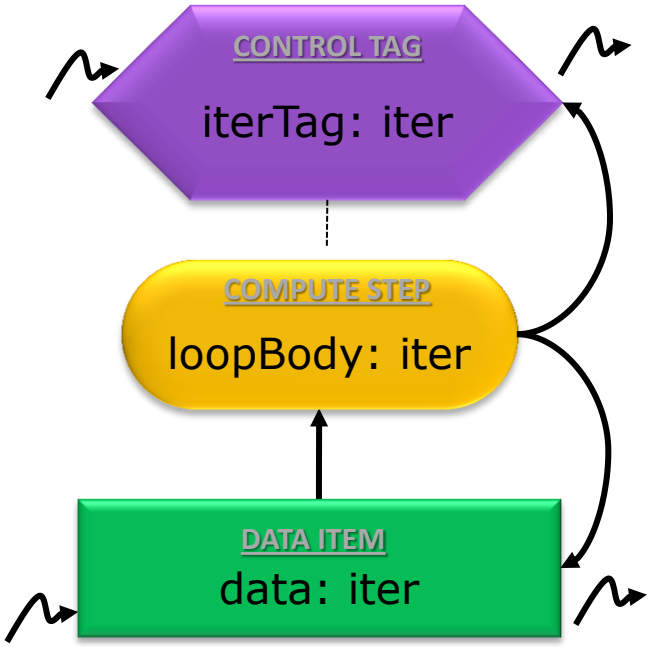
First iteration on initial data



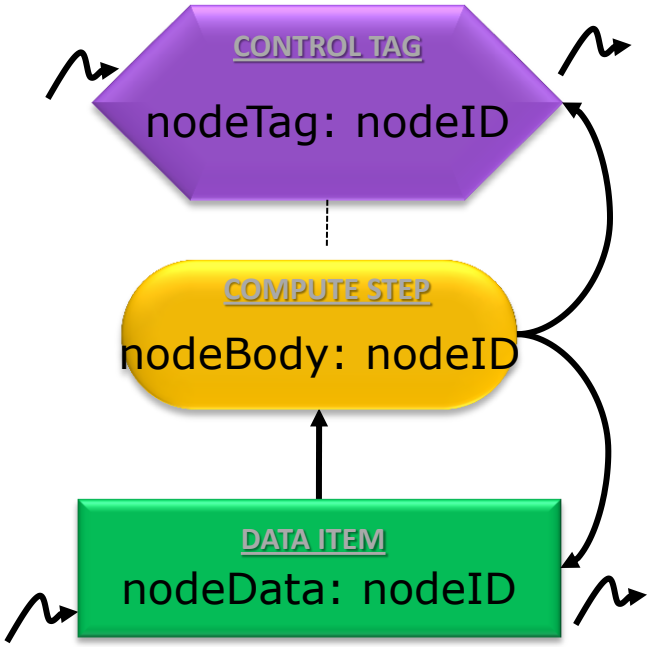
An iteration might produce the next control tag and the next data item



The data items and tags might be output



Exactly the same as building a tree
Output produces more than one tag and item



loop iteration, binary tree, quad tree, oct tree

Depth-first or breadth-first

What is it?

Influences

- Data flow but not just data flow
- Also control flow
- Influenced by tuple-spaces

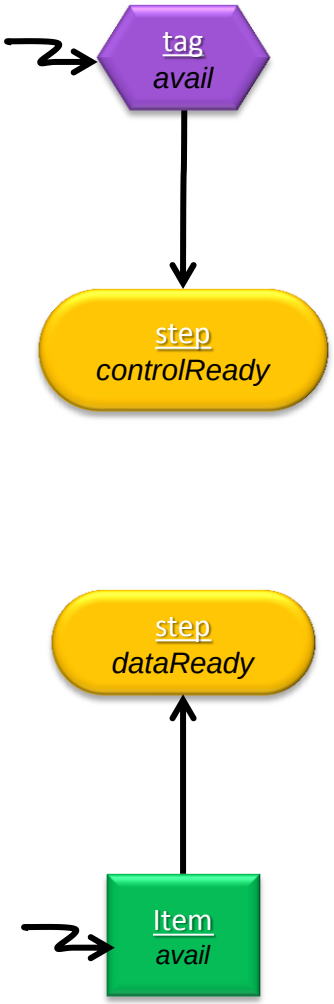
Alternate view

- Working in static analysis for parallel systems. Realize:
  - The user knows what the compiler is not able to uncover
  - The language gets in the way
- Design a language the allows the user to say exactly what we want to know

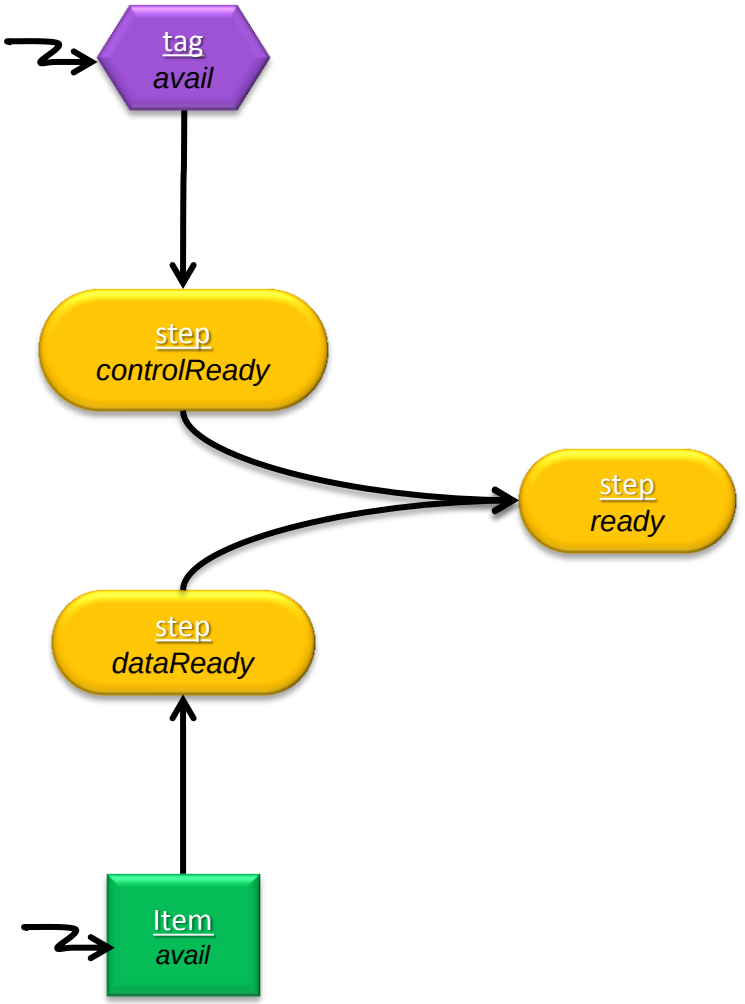
- User writes an ideal PDG
  - Data dependences
    - Only true dependences
    - No anti-dependences
    - No output dependences
  - Control dependences
    - PDG region nodes
- Dynamic single assignment

How to think about CnC execution

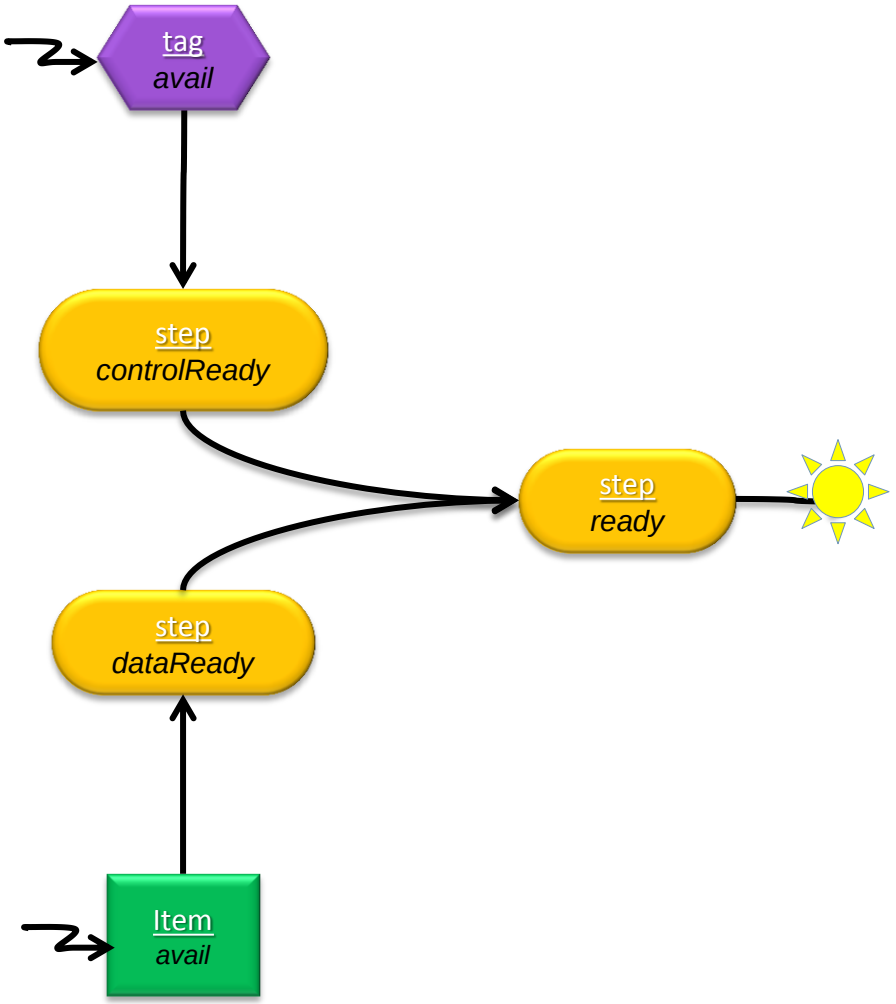
How to think about CnC execution



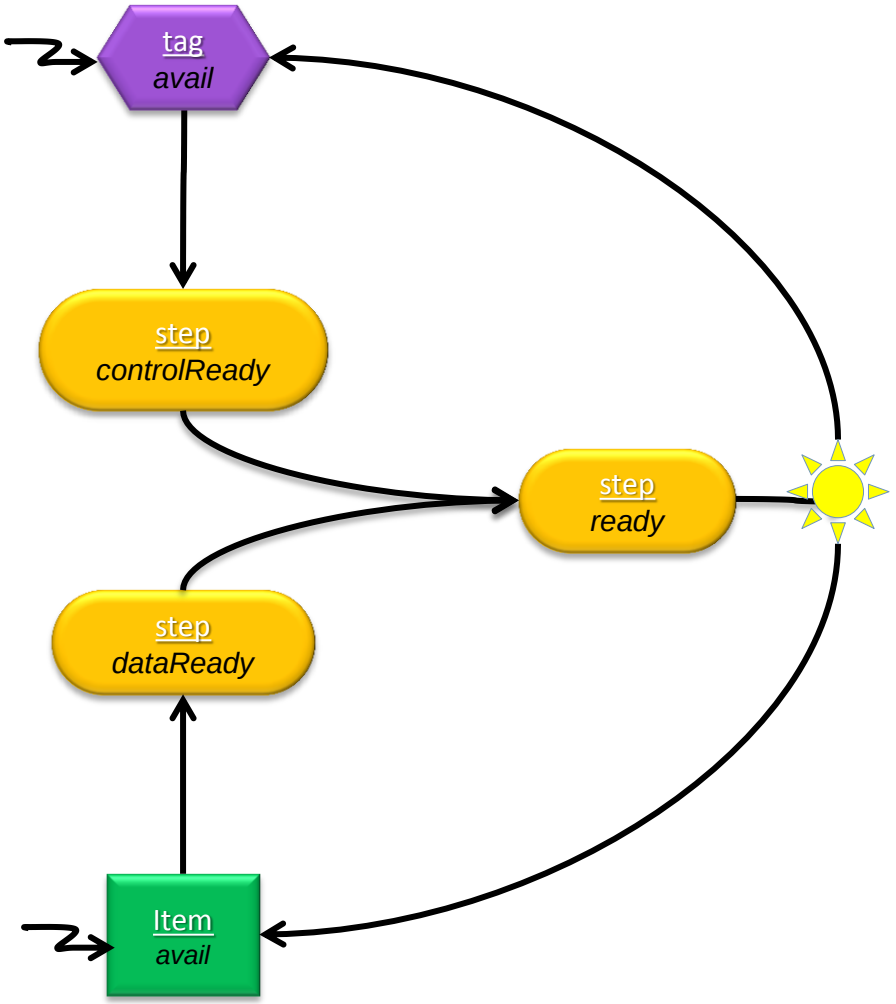
How to think about CnC execution



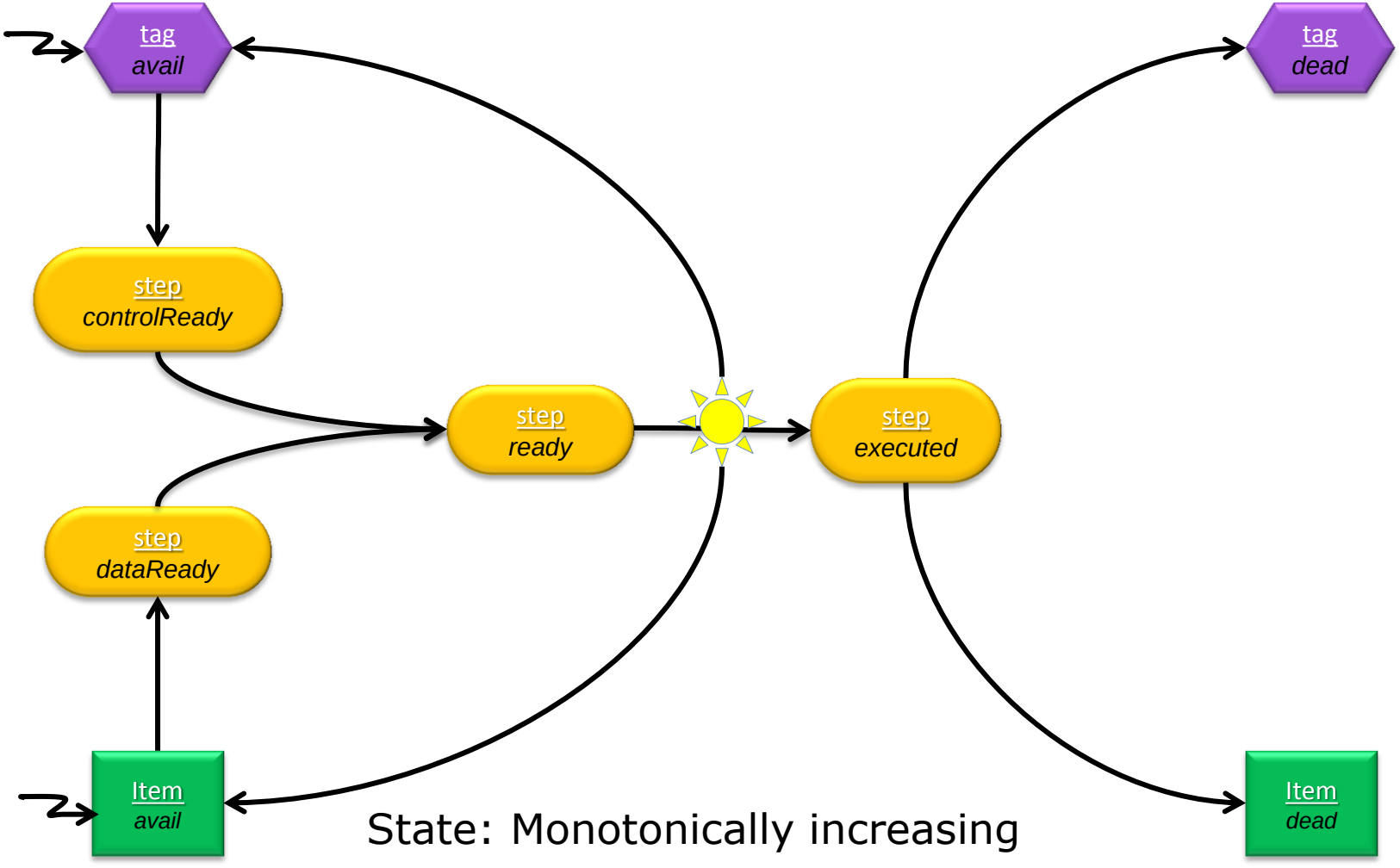
How to think about CnC execution



How to think about CnC execution



How to think about CnC execution



State: Monotonically increasing

- Set of instances with any attribute
- Set of attributes of those instances
- Contents of available items

Analyzability

Characteristics that support analyzability

- Dynamic single assignment
- No side-effects
- Deterministic
- Optional tag functions and edge annotations
- Both control and data dependences are explicit
- Serializable

What might the analysis support

- Determine when data item is dead [DAMP 2009: Zoran Budimlic, et. al.]
- Optimize generic runtime [CPC 2013: Kath Knobe, Zoran Budimlic]
- Mapping computation steps to time/platform [PDP 2013: Frank Schlimbach, et. Al.]
- Mapping data items to memory (or to each other) [Europar2012: Dragos Sbirlea, et. al.]

Productivity

- Deterministic
- Potential for tools at the level the domain expert understands
  - Computation steps, data item in the domain spec (not processors, threads, …)

Distributed memory

- Unified language (not MPI + x)
  - Still just graph of steps, items and tags
- User can provide functions for mapping
  - Functions are isolated from code
  - Either focus on distribution of data or of computation

Portability

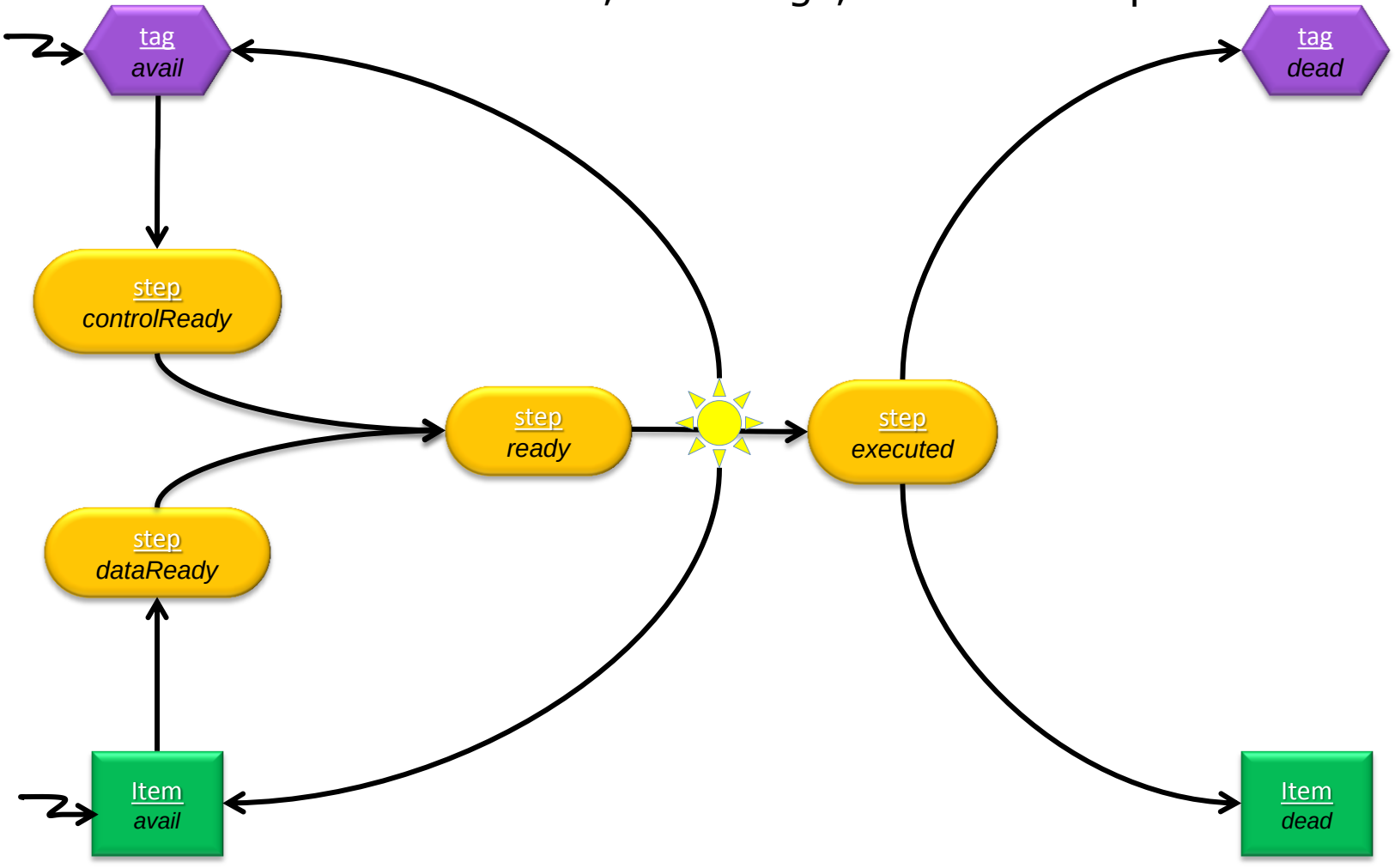
- Memory
  - Single processor
  - Shared memory
  - Distributed memory (Intel)
- Heterogeneous platforms
  - CPU/MIC (Intel – not released)
  - CPU/GPU/FPGA (Rice)

Outline

- Intro
- Checkpoint/restart
- Tuning language
- Adaptive computing

Execution frontier:

leading edge: monotonically increasing
trailing edge: remove irrelevant instances
dead items, dead tags, executed steps



leading edge: monotonically increasing
trailing edge: remove irrelevant instances
dead items, dead tags, executed steps

Execution frontier

- An *execution frontier* is a CnC program state:
  - The set of attributes of instances of steps, tags and items
  - The contents of items (available but not dead)
  - No state of the runtime data structures

Checkpoint/restart

Nick Vrvilo (Rice)

- Continuously, asynchronously save state changes
- No user involvement required
  - User can optimize
- No barriers
- No synchronization
- Can restart from any saved state
  - On a different machine
  - On a different configuration
  - On a different CnC runtime
  - …

Outline

- Intro
- Checkpoint/restart
- Tuning language
- Adaptive computing

Domain spec / tuning spec

- Separate specs
  - One domain spec / multiple tuning specs
- Domain spec/ tuning spec
  - Same person / different time
  - Different people with different expertise
  - Person / automatically generated tuning spec
- Distribution functions for distributed CnC
  - Currently restricted to where (not when)
  - Currently restricted to where at the node level

Tuning

Tuning – Sanjay Chatterjee, Zoran Budimlic, Mike Burke (Rice)

- Parallelism is there from CnC
- Load balance is there from work stealing runtimes
- Issue is locality
  - Locality => time and space (across the platform)
  - No benefit if too far away in either time or space

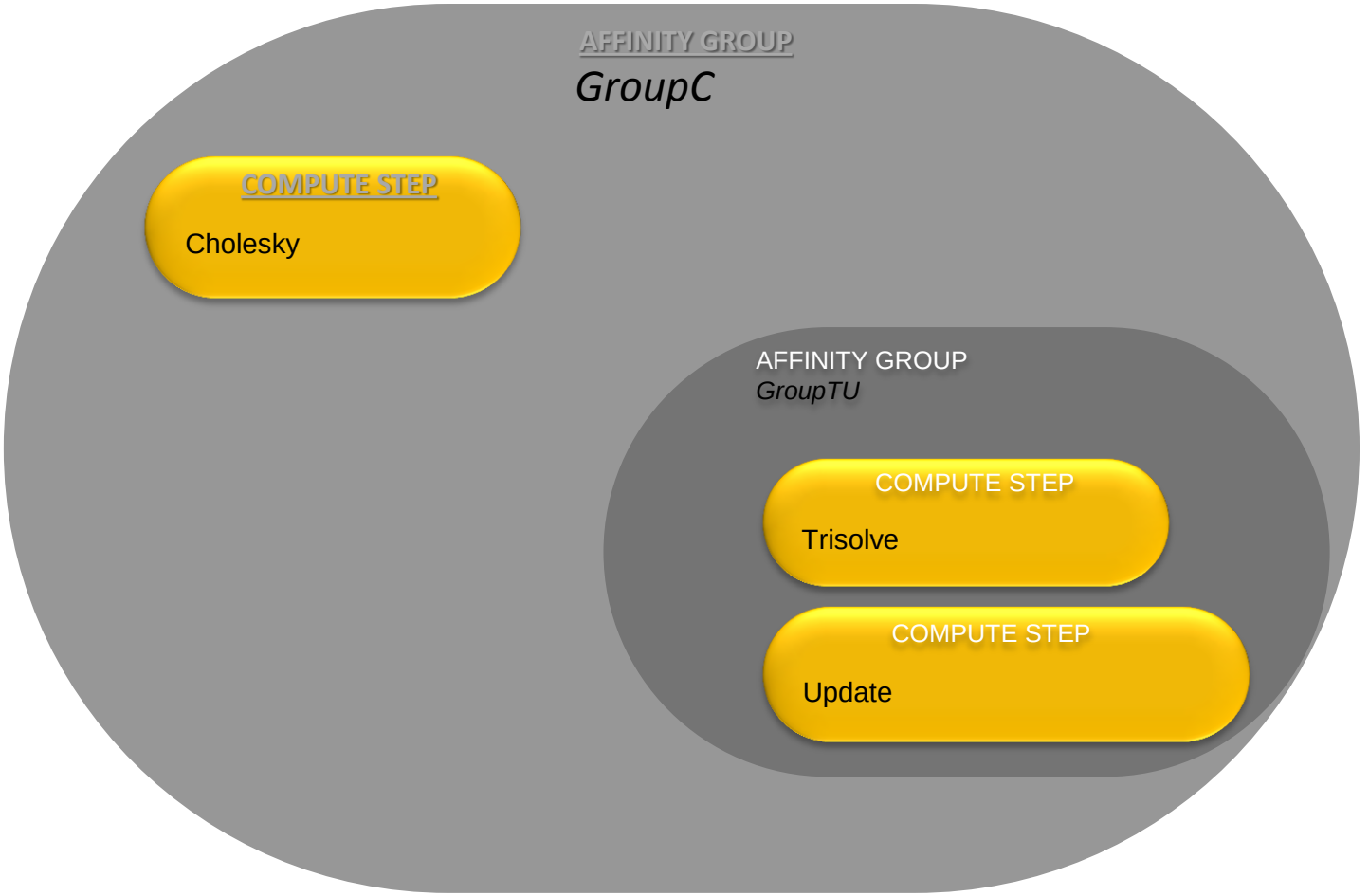
Hierarchical affinity groups

- Affinity groups – For locality
  - Foundation:
    - Doesn't distinguishing between spatial and temporal locality

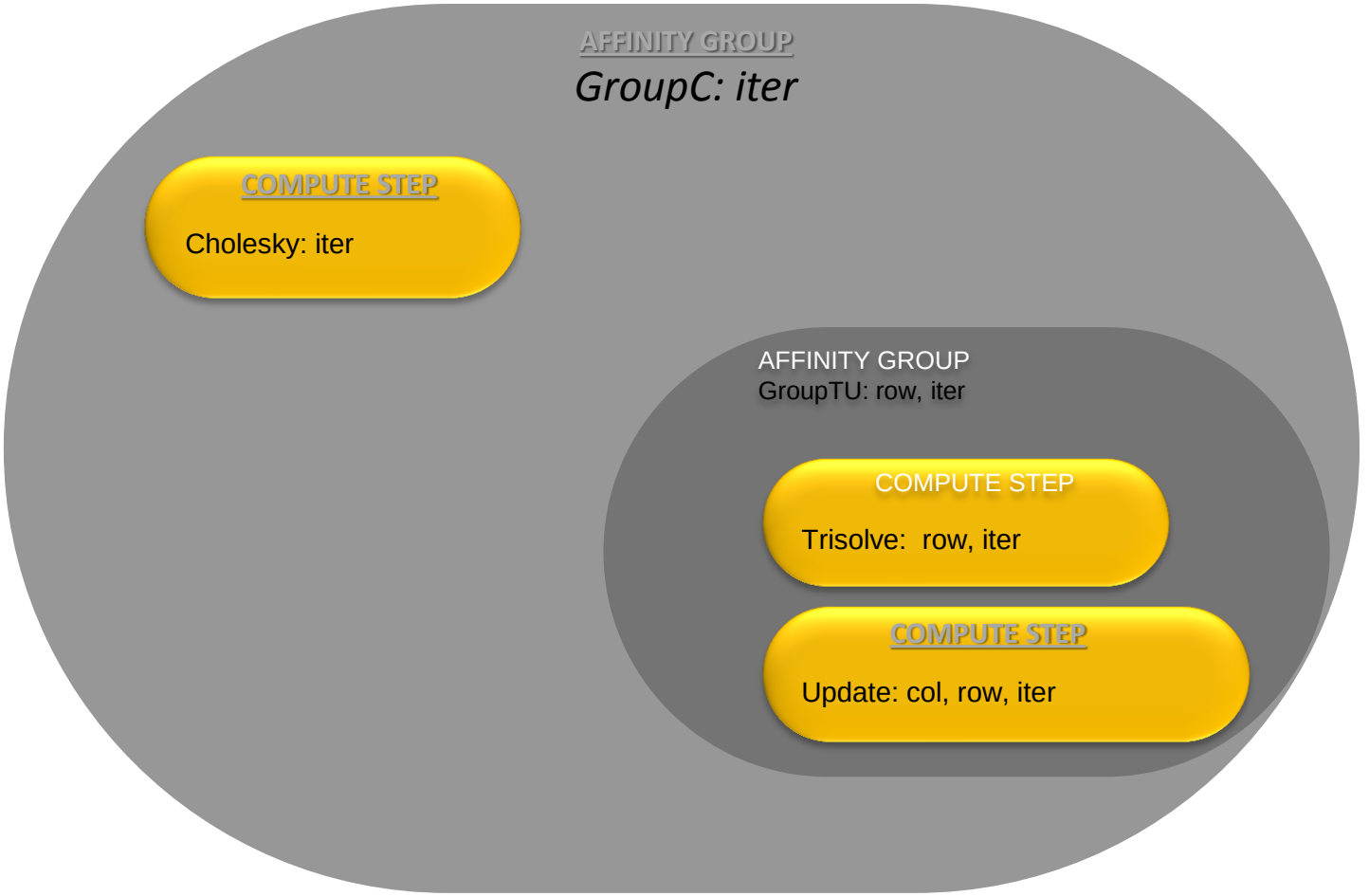
      (supports space-time locality)
  - On top of this foundation:
    - we allow (but don't require) time-specific and space-specific control

- Hierarchical affinity groups
  - Computations in the same low-level group have tight affinity
  - Computations in the same higher-level group have a weaker affinity

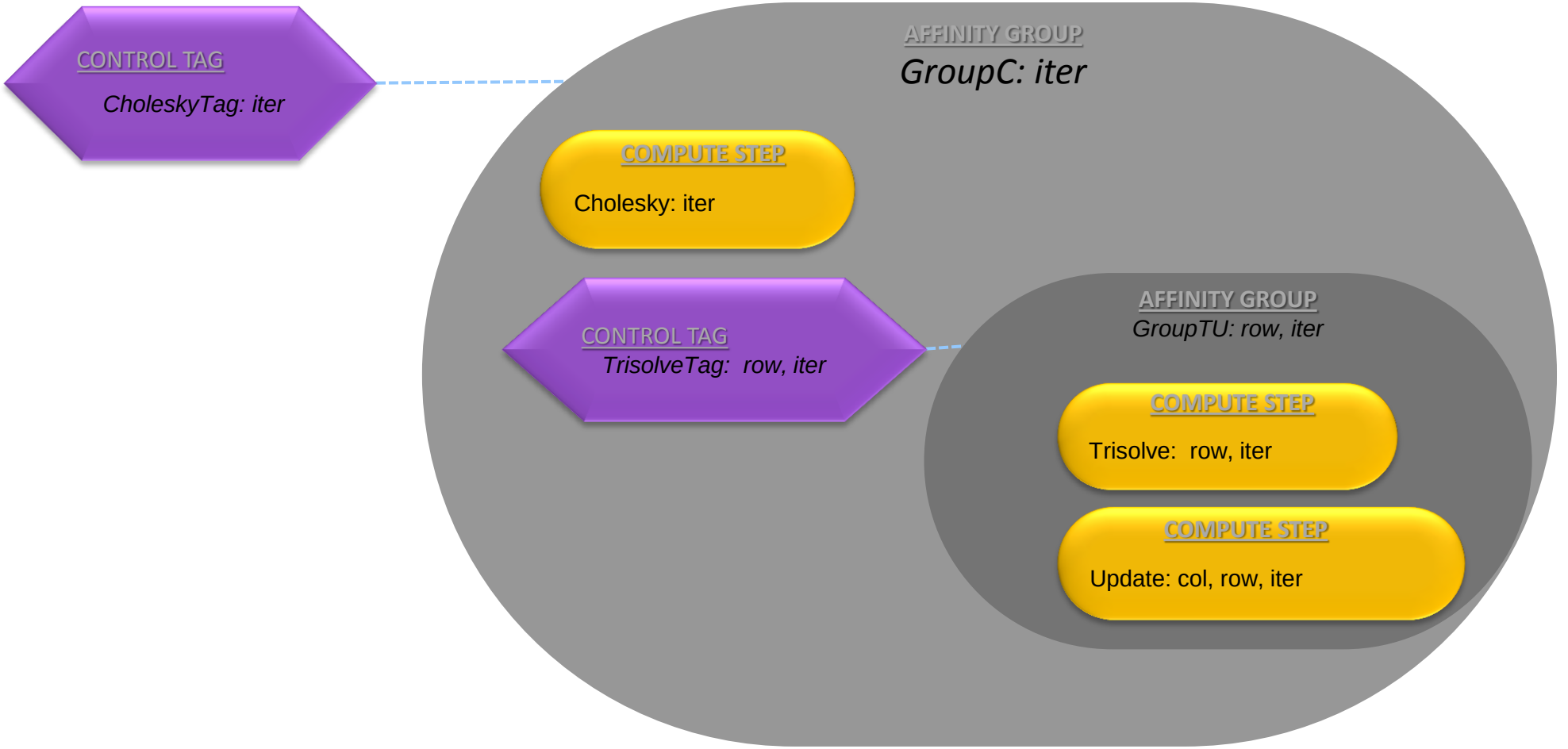
Three phase methodology
1: Hierarchical affinity structure

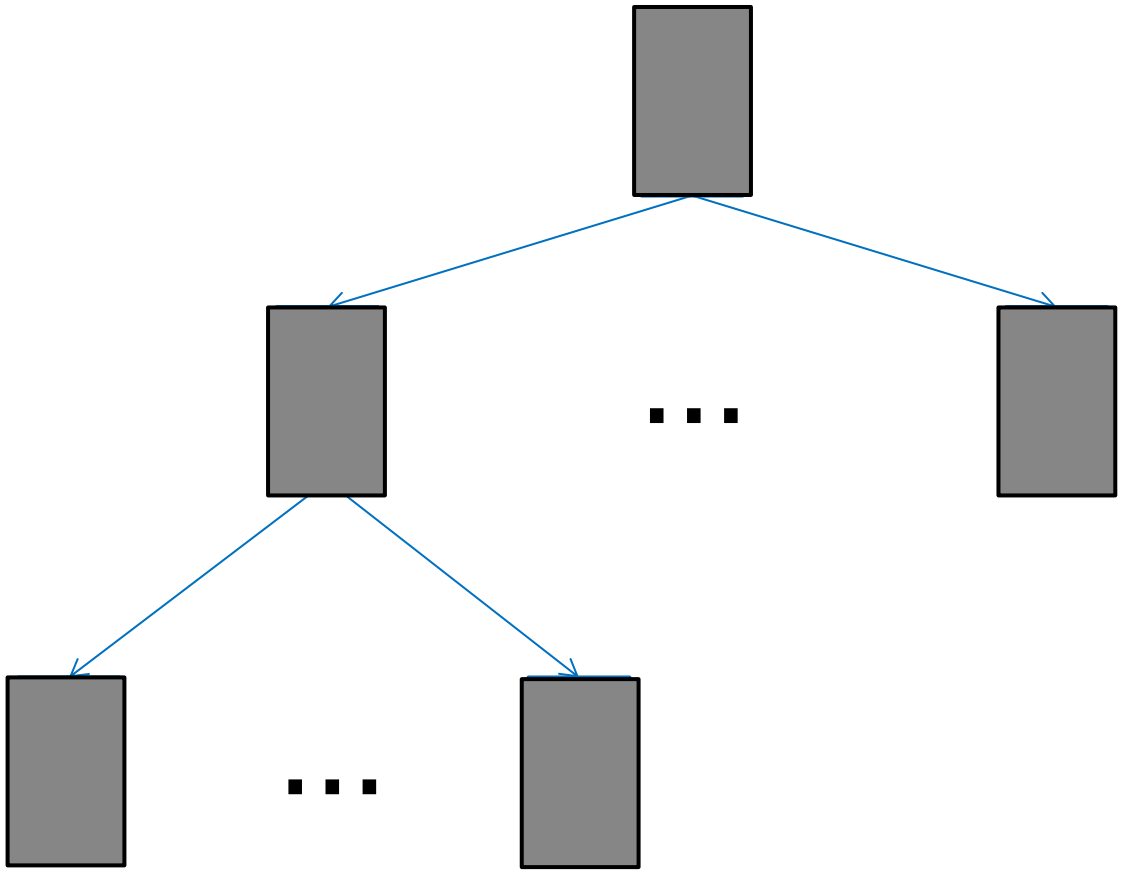


2: Distinguish among instances

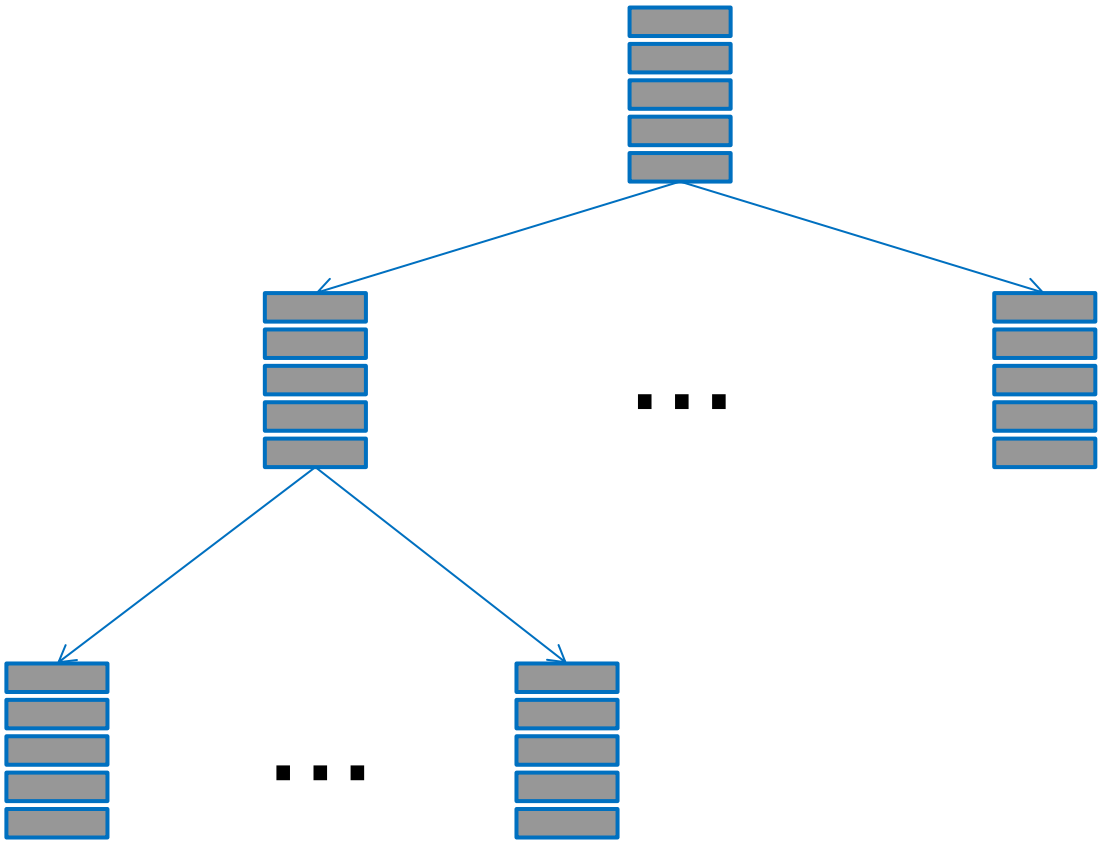


3: Specify the instances of each





Levels correspond to levels in the platform hierarchy, e.g., sockets, address spaces, …

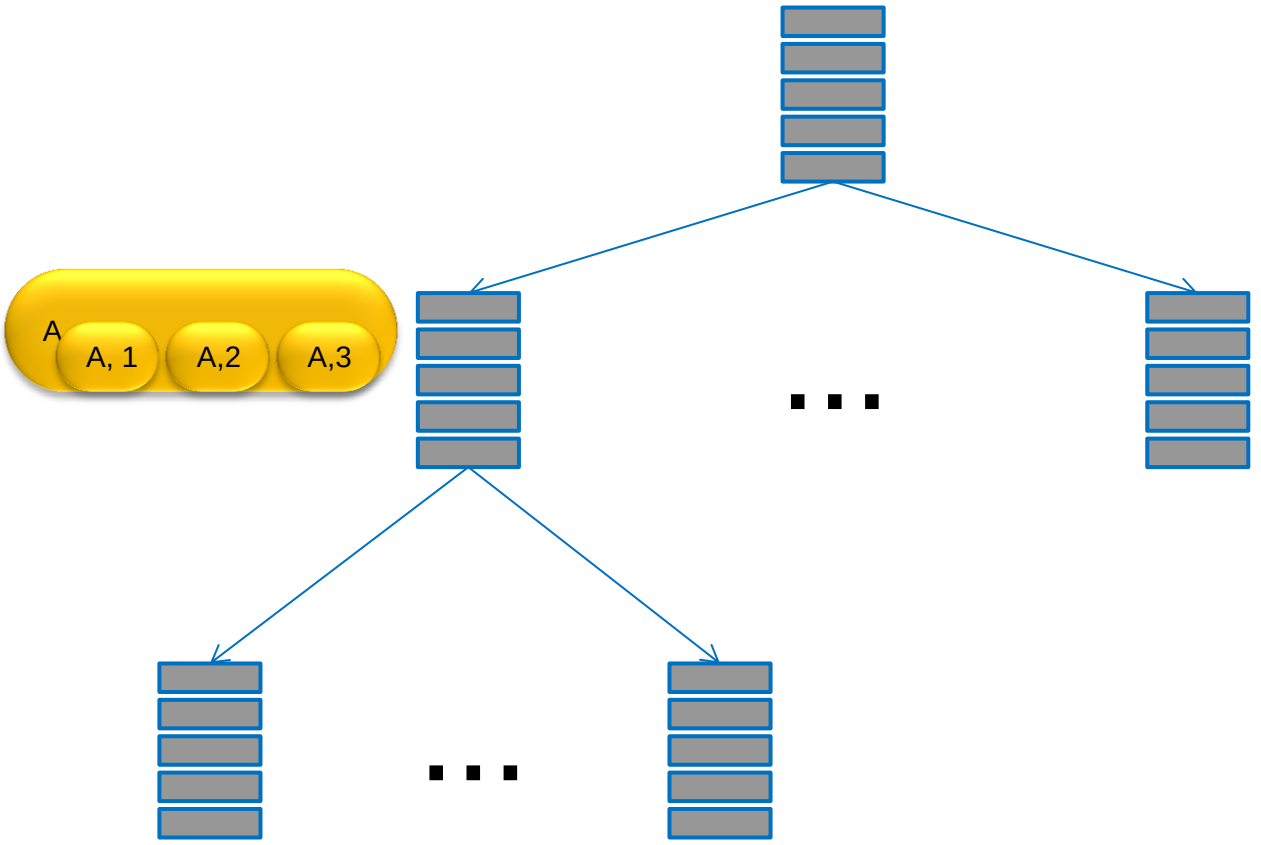


Shape conforms to platform
Queue

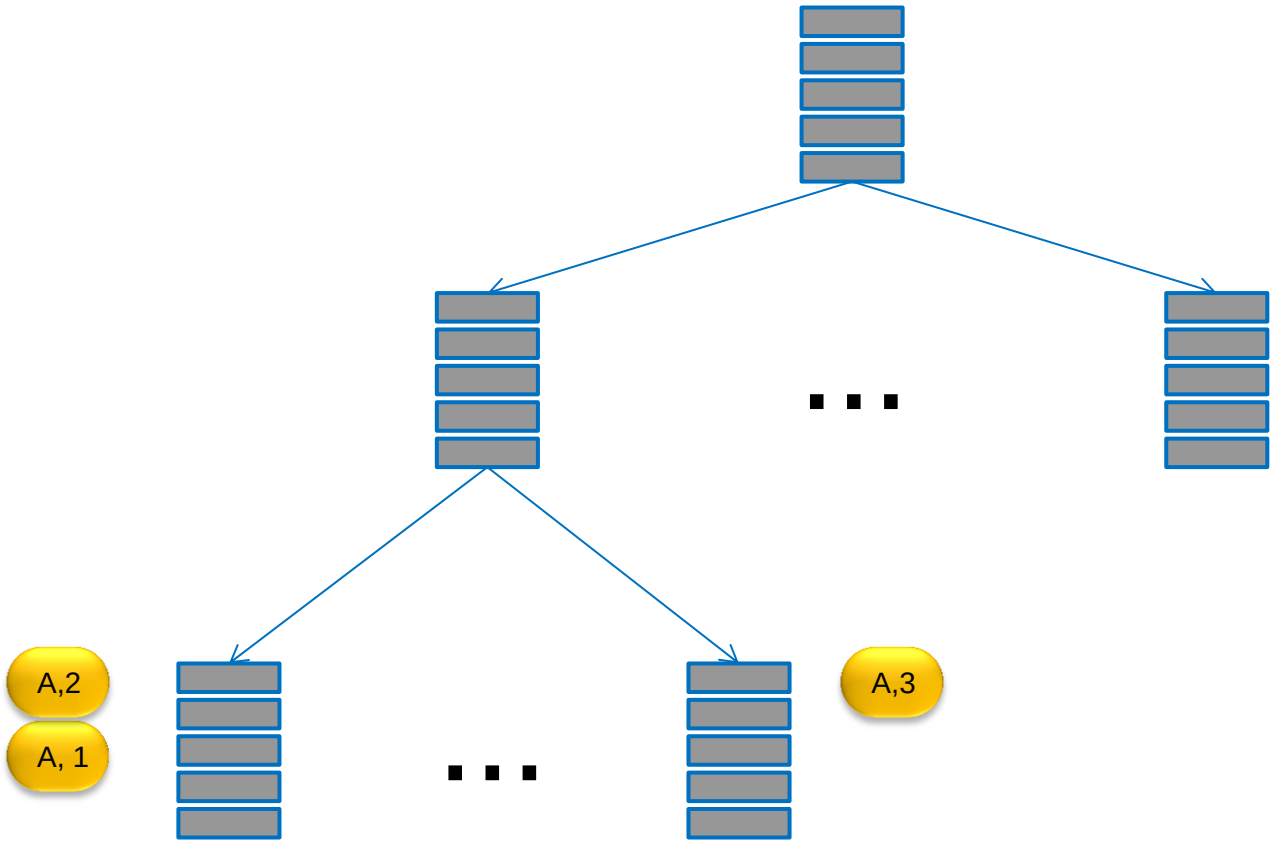
Overview of tuning execution model

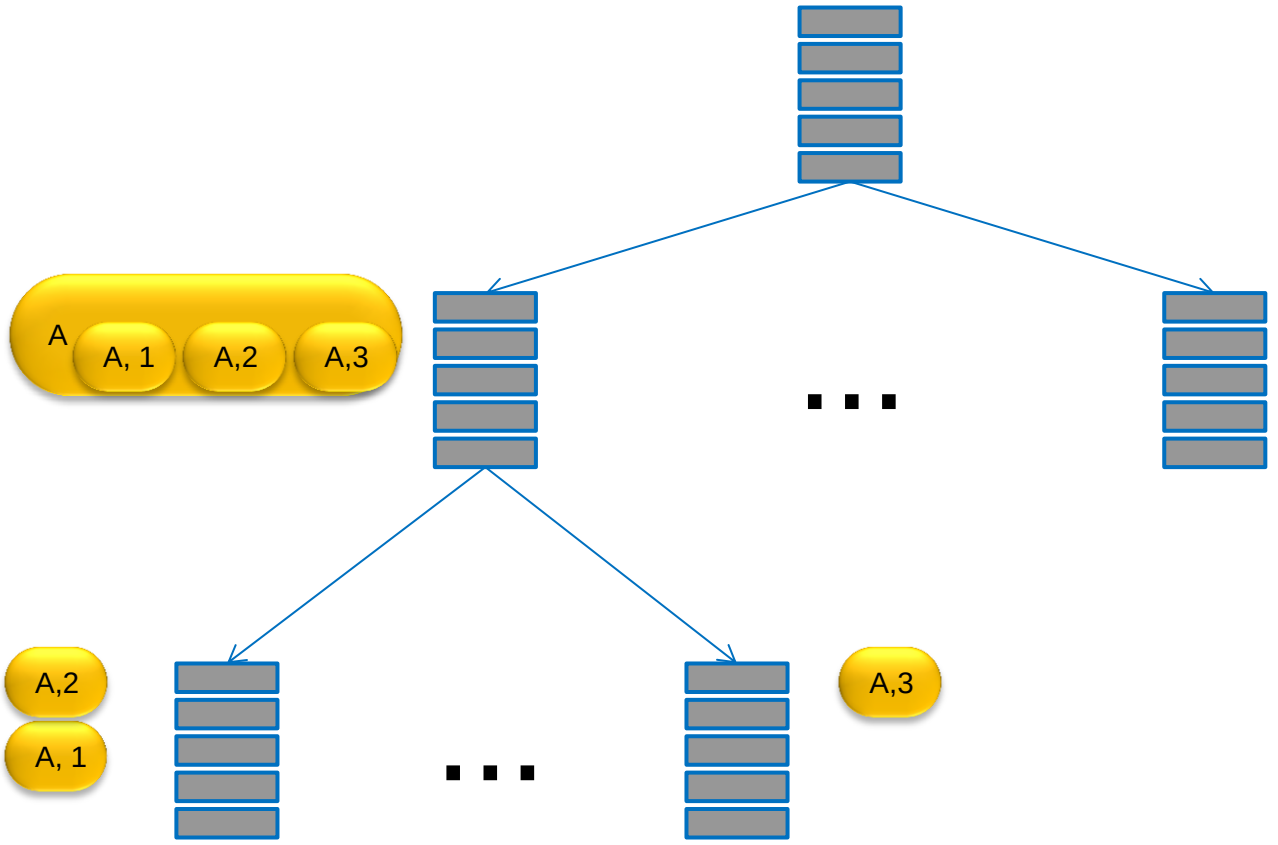
- Put top-level group at the top of tree
- Break it into its lower-level components
- Put these components down to children within the parent node
- Bottom level feeds into our normal runtime
- Acts as a staging area holding computation back

Tuning execution



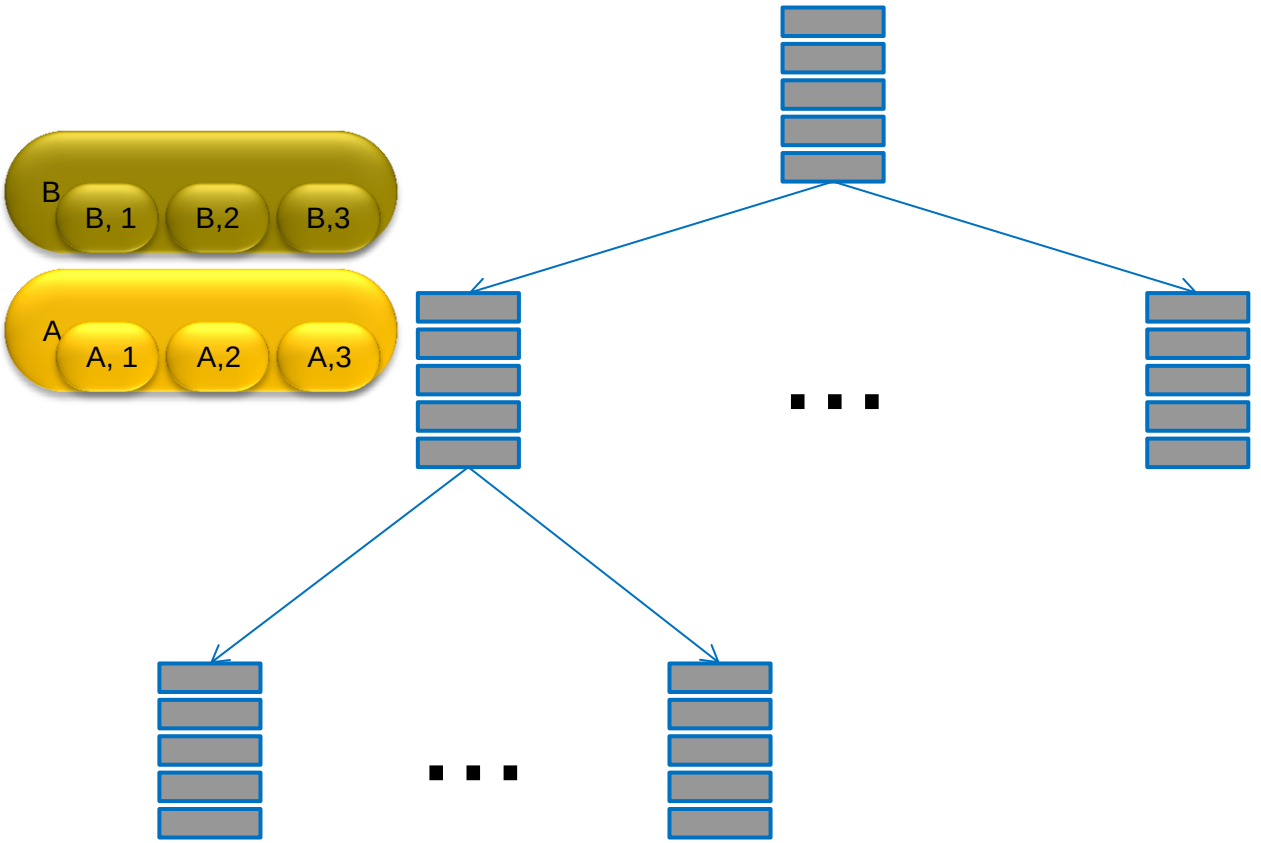
Tuning execution

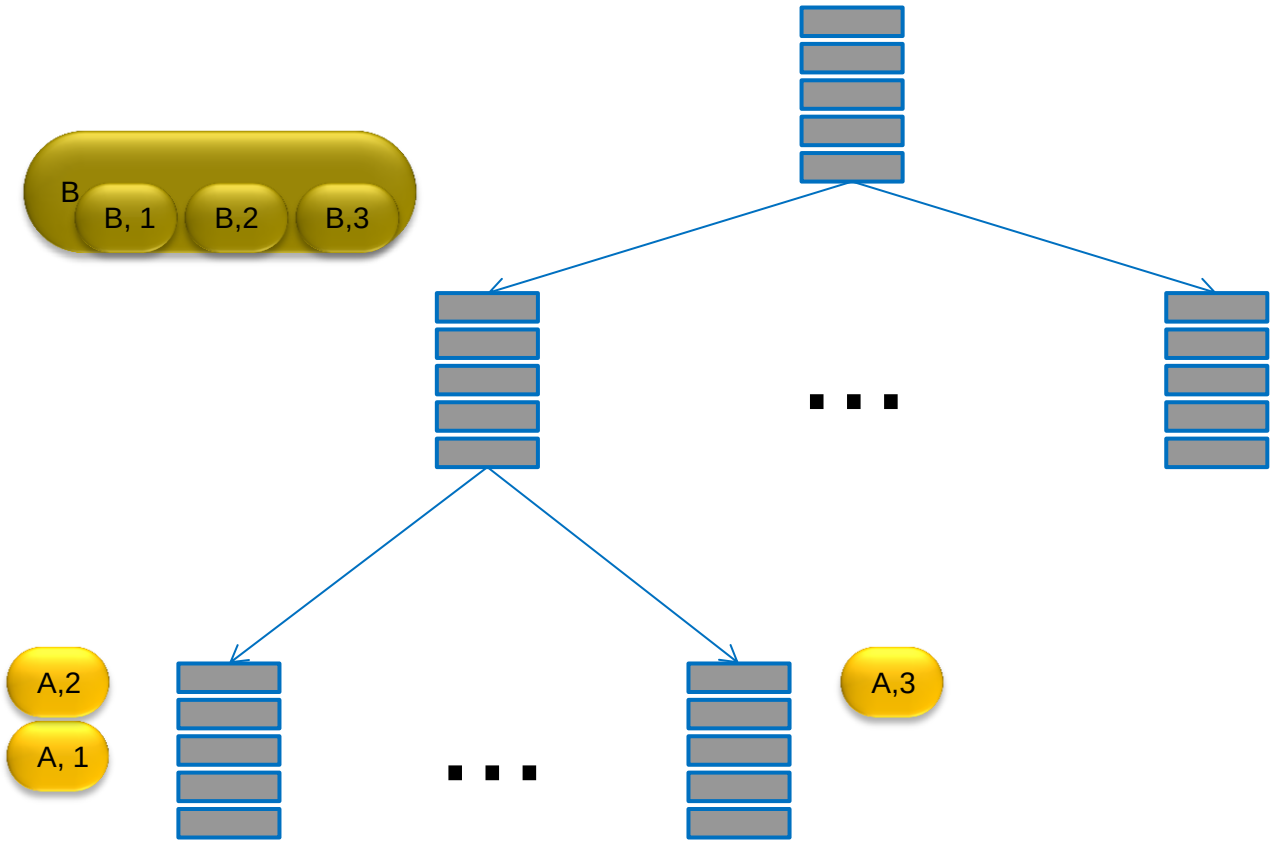


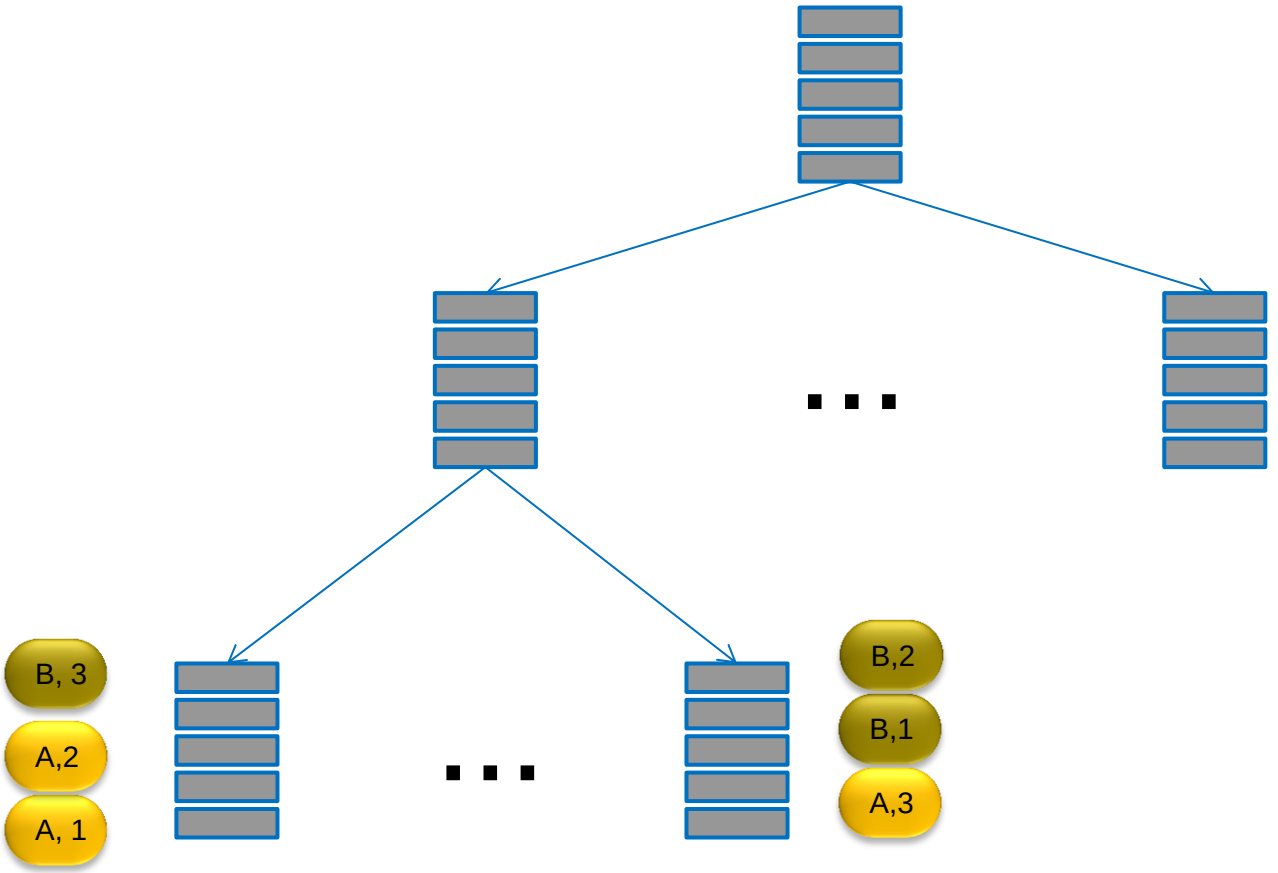


Outer group at a node.

Components of Outer Group stay within children of that node


B
B, 1
B,2
B,3
A
A, 1
A,2
A,3

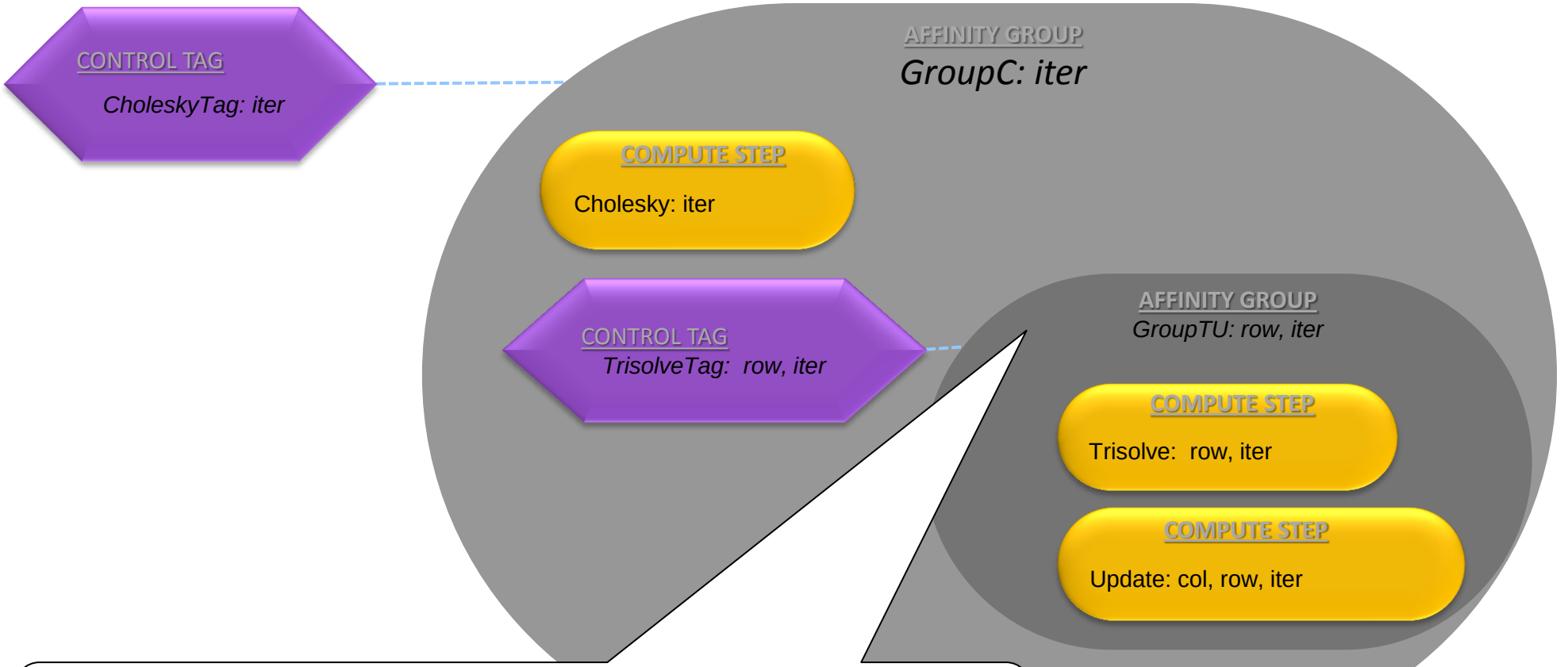

B
B, 1
B,2
B,3
A,2
A, 1
A,3



If group A arrives at a node before group B

The components group A arrive at the child node before the components of group B

Space-specific tuning



- **Replicate components of this group across nodes**
- **Distribute components of this group across sockets**
- **Distribute components of this group across nodes via func()**

Time-specific mappings

Relative time among components within a group.

	ordered	**unordered**
Non-overlapping	serial/barrier	exclusive
overlapping	priority	arbitrary

Productivity - tuning

Isolate domain and tuning

For example: Cholesky we have one domain spec and 5 or 6 tuning specs

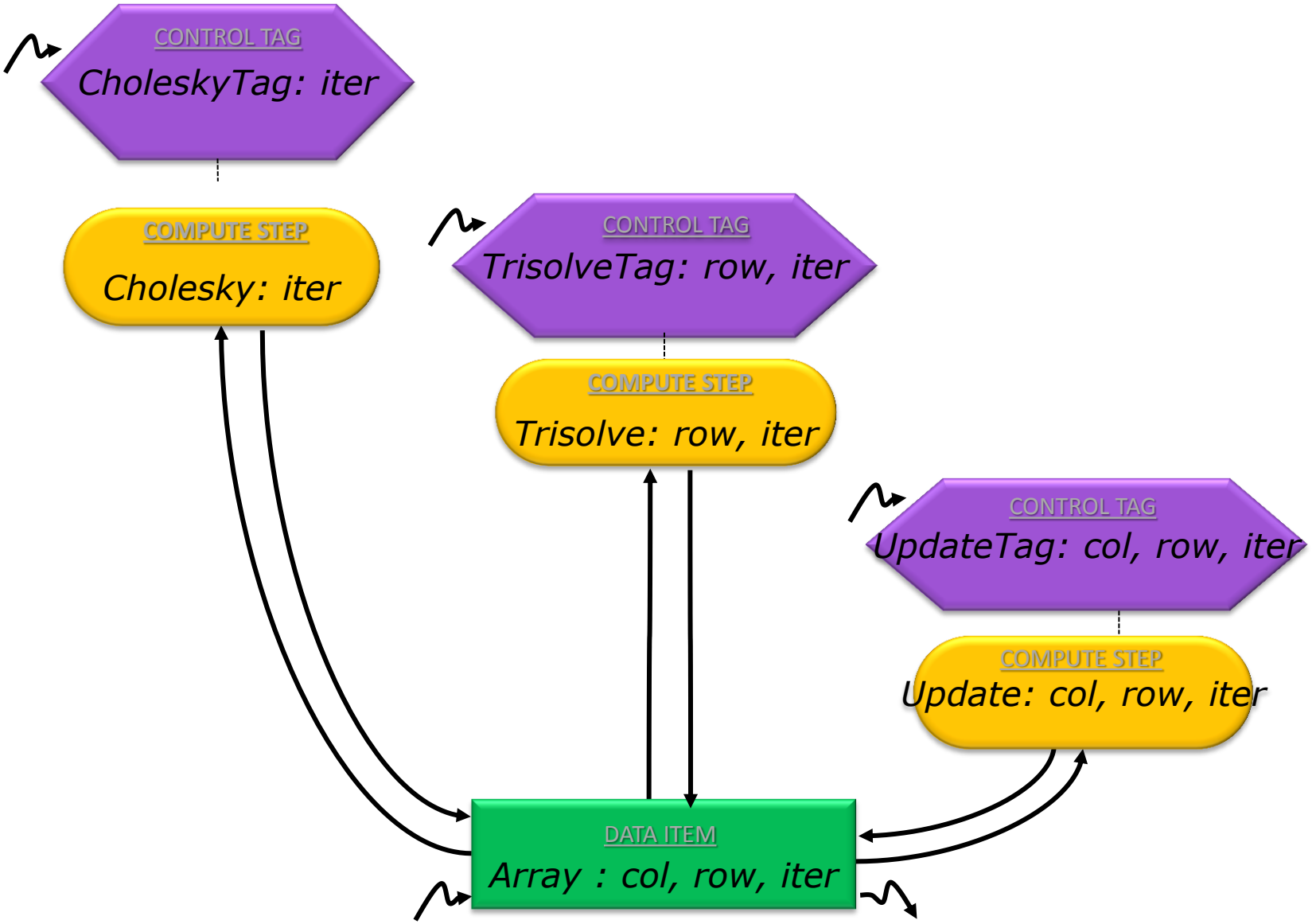
- May not need tuning
- Domain expert can focus on chemistry, image processing, …
  - doesn’t need to see the tuning spec
- Tuning expert can focus on performance
  - doesn’t need to wade through domain code

Interactions? Sure. But at the level of the specs

Outline

- Intro
- Checkpoint/restart
- Tuning language
- Adaptive computing

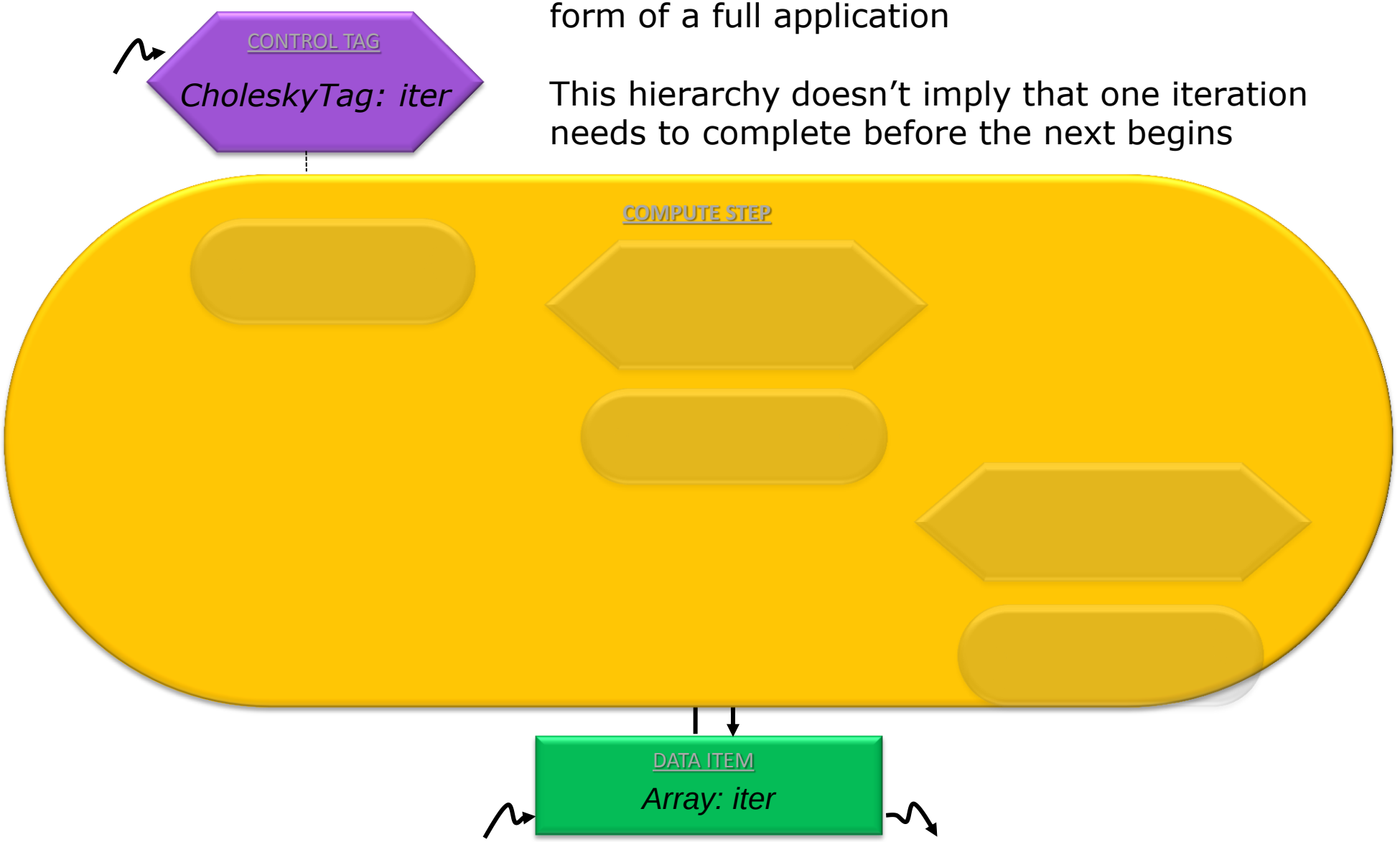
Hierarchy


CONTROL TAG
CholeskyTag: iter
COMPUTE STEP
Cholesky: iter
CONTROL TAG
TrisolveTag: row, iter
COMPUTE STEP
Trisolve: row, iter
CONTROL TAG
UpdateTag: col, row, iter
COMPUTE STEP
Update: col, row, iter
DATA ITEM
Array : col, row, iter

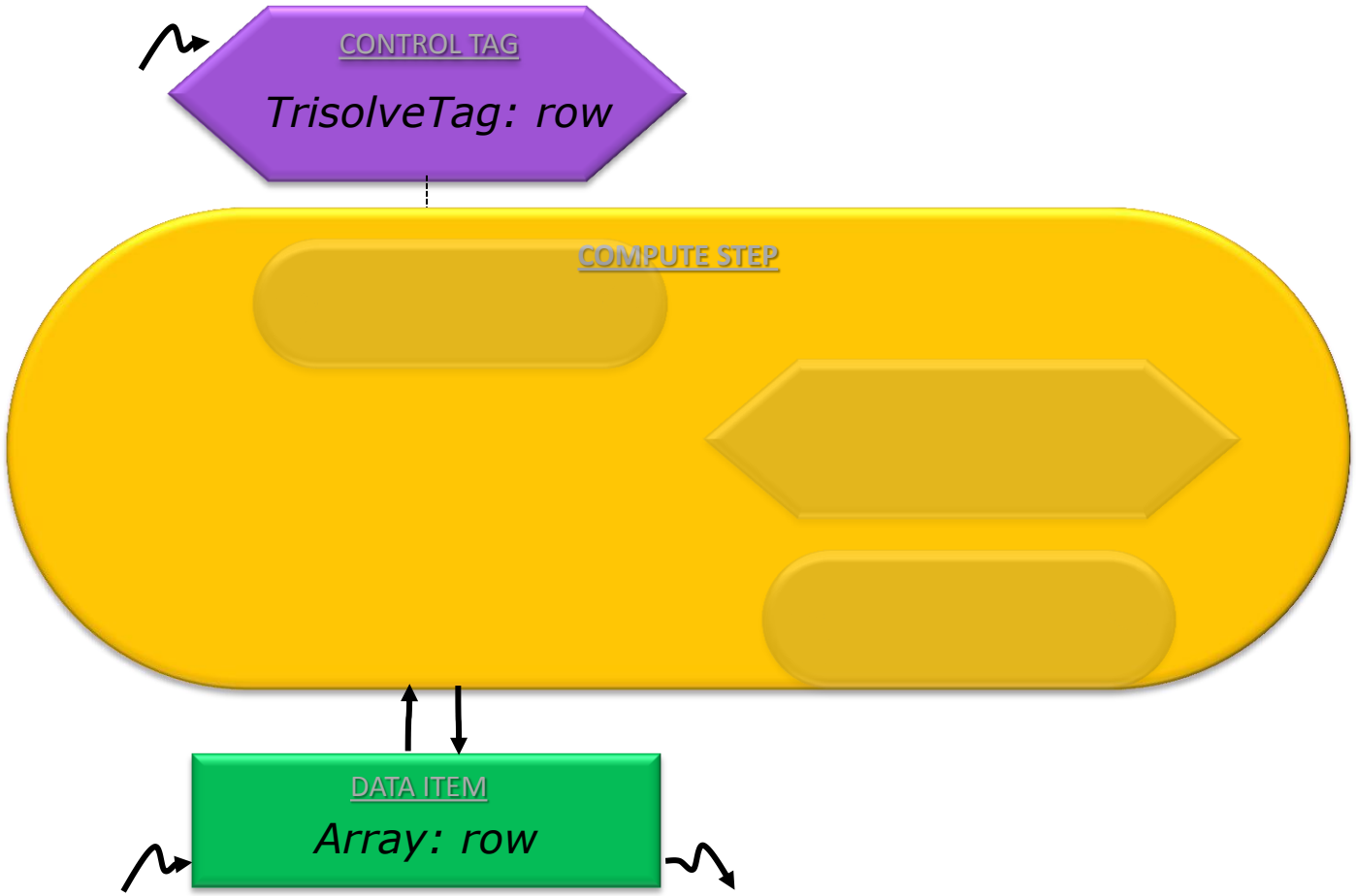
Hierarchical domain spec

Each application node in the hierarchy has the form of a full application

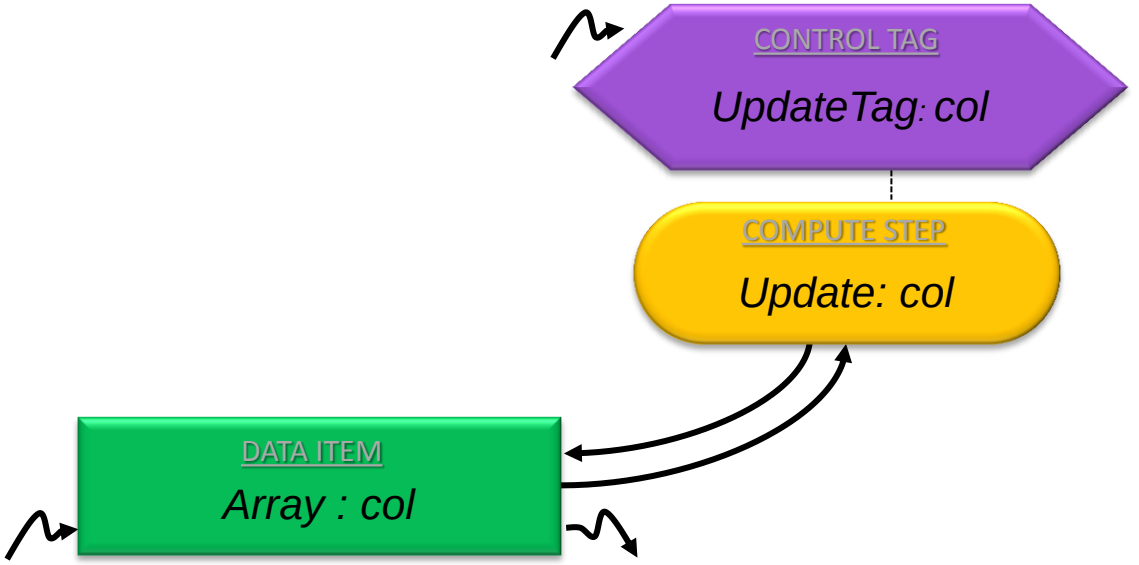
This hierarchy doesn't imply that one iteration needs to complete before the next begins



Hierarchical domain spec



Hierarchical domain spec



Motivation: Highly adaptive computing for exascale

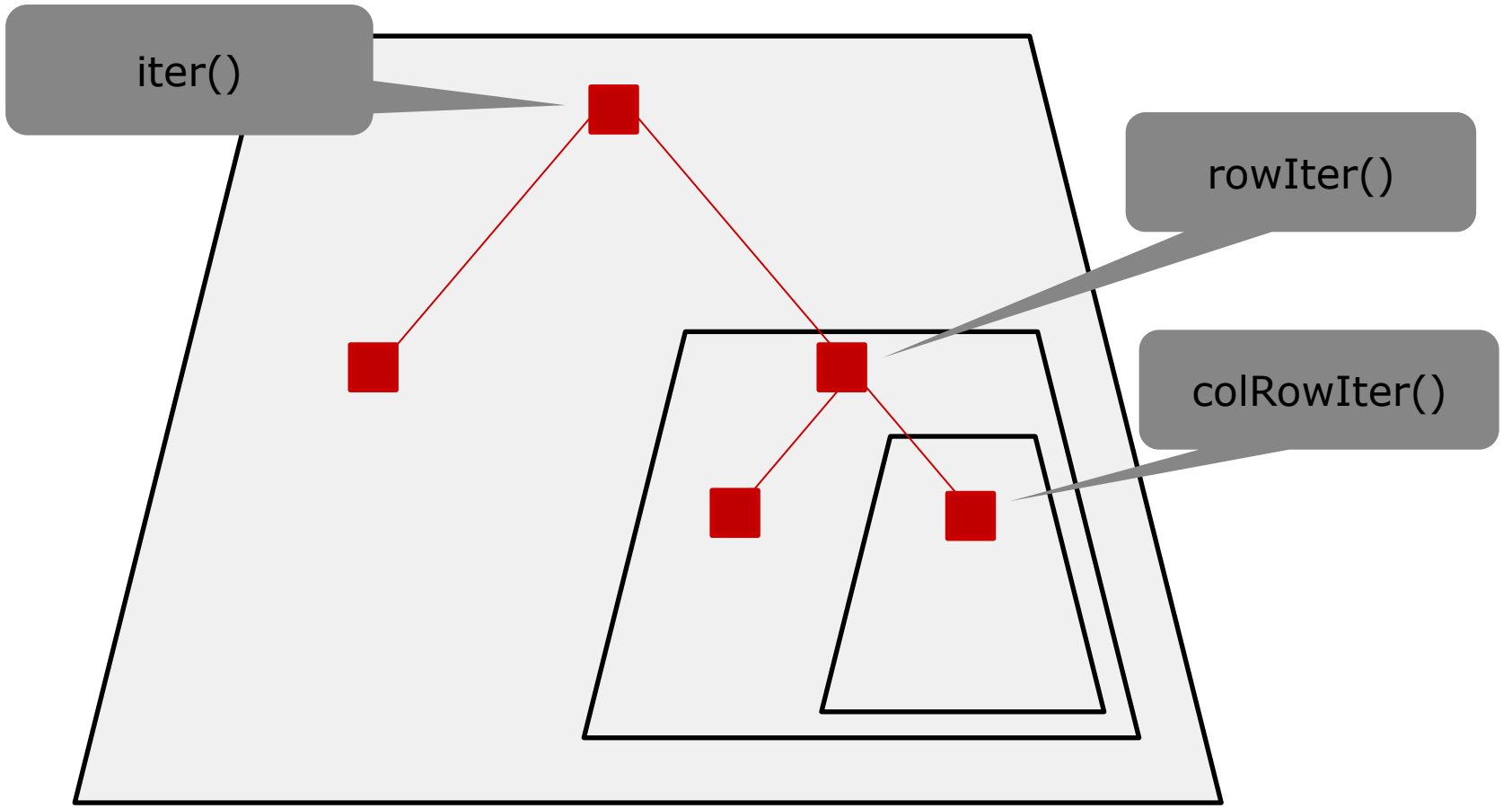
Critical exascale issues (based on UHPC and X-Stack)

Require the ability to move currently executing parts of the app to another *place* in the platform or to a later *time*.

- Resilience
  - Fragile components
  - Lots of them
- Power management
  - Power components on/off
  - Power components up/down
- Self-aware computing
  - Modify mapping based on feedback
- Change of goals
  - Between power and time to solution, for example

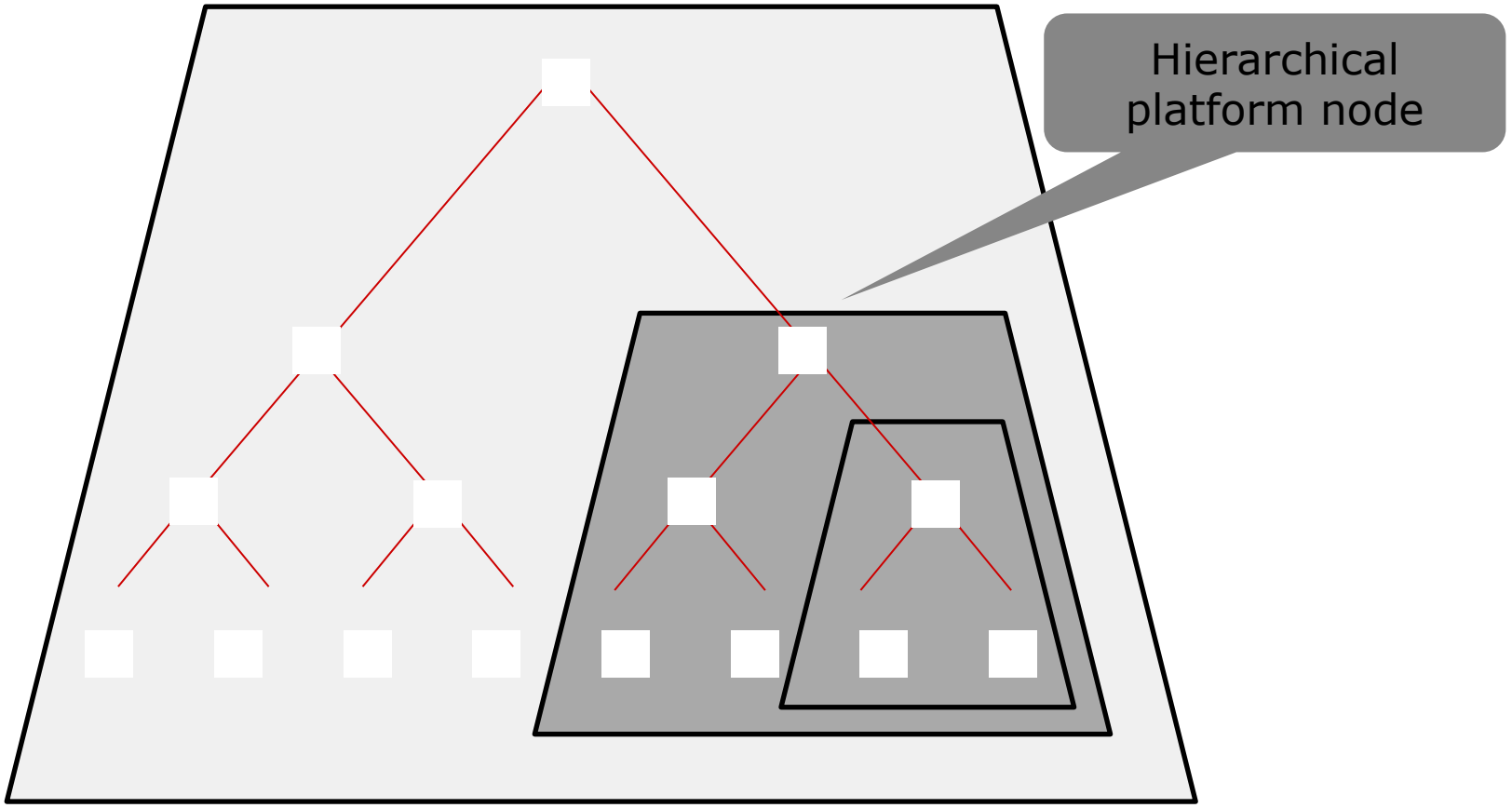
Abstract view of application hierarchy

A node at any level has the form of a full application
Input, computation, output



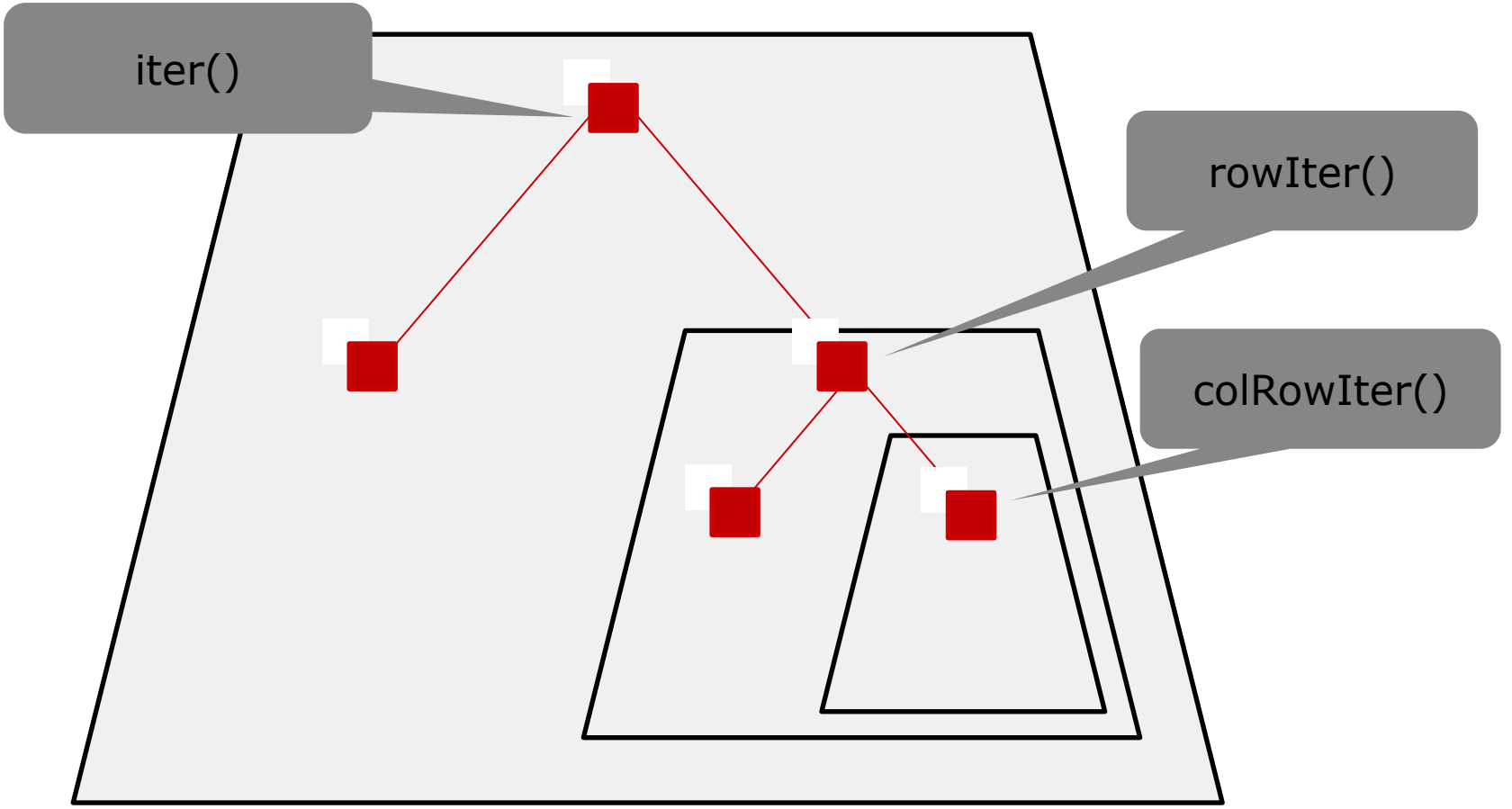
Abstract view of the platform hierarchy

A node has the form of a full machine at each level:
a subtree of the memory hierarchy + set of cores



Abstract app maps to abstract platform

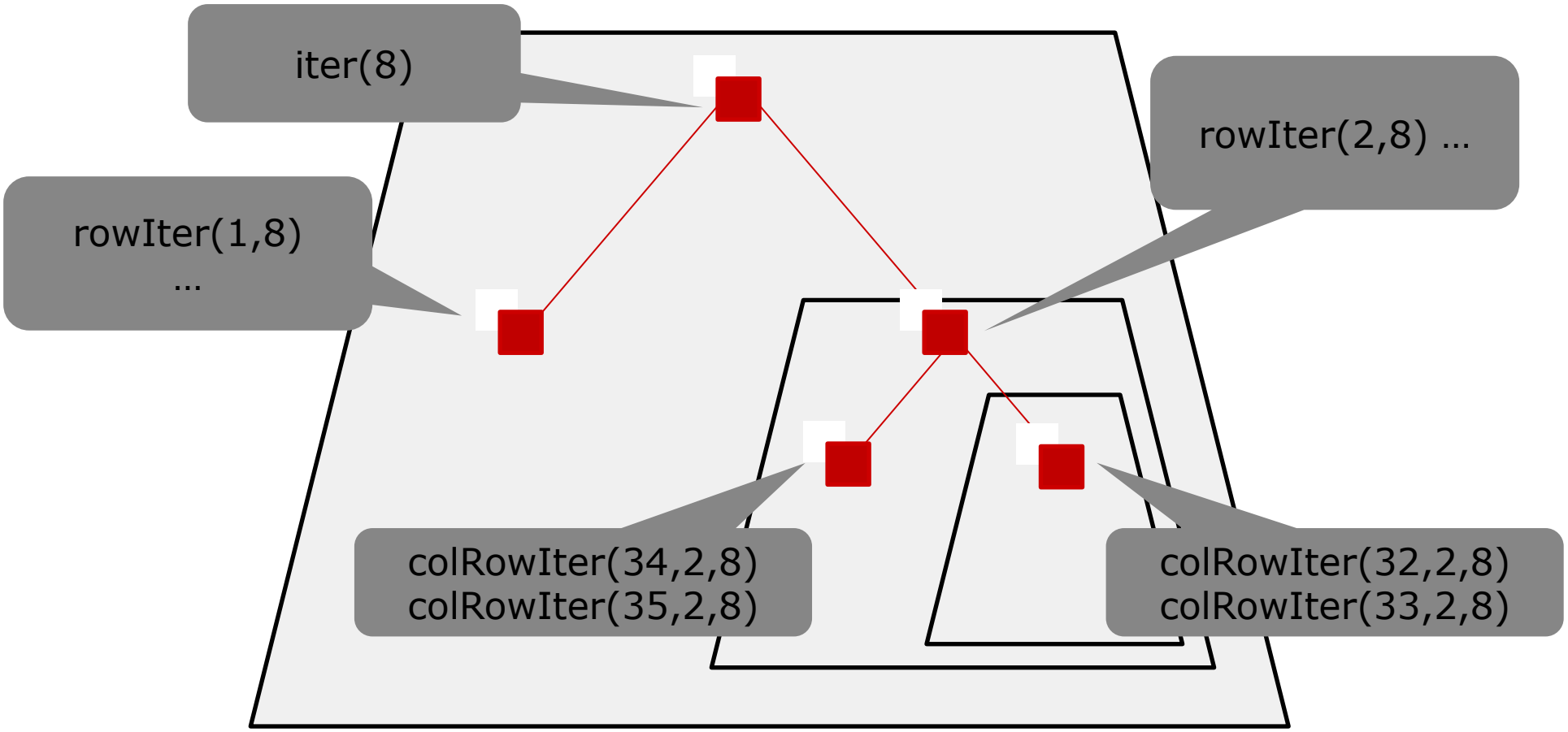
Assume the shape of platform hierarchy corresponds exactly to the shape of the application



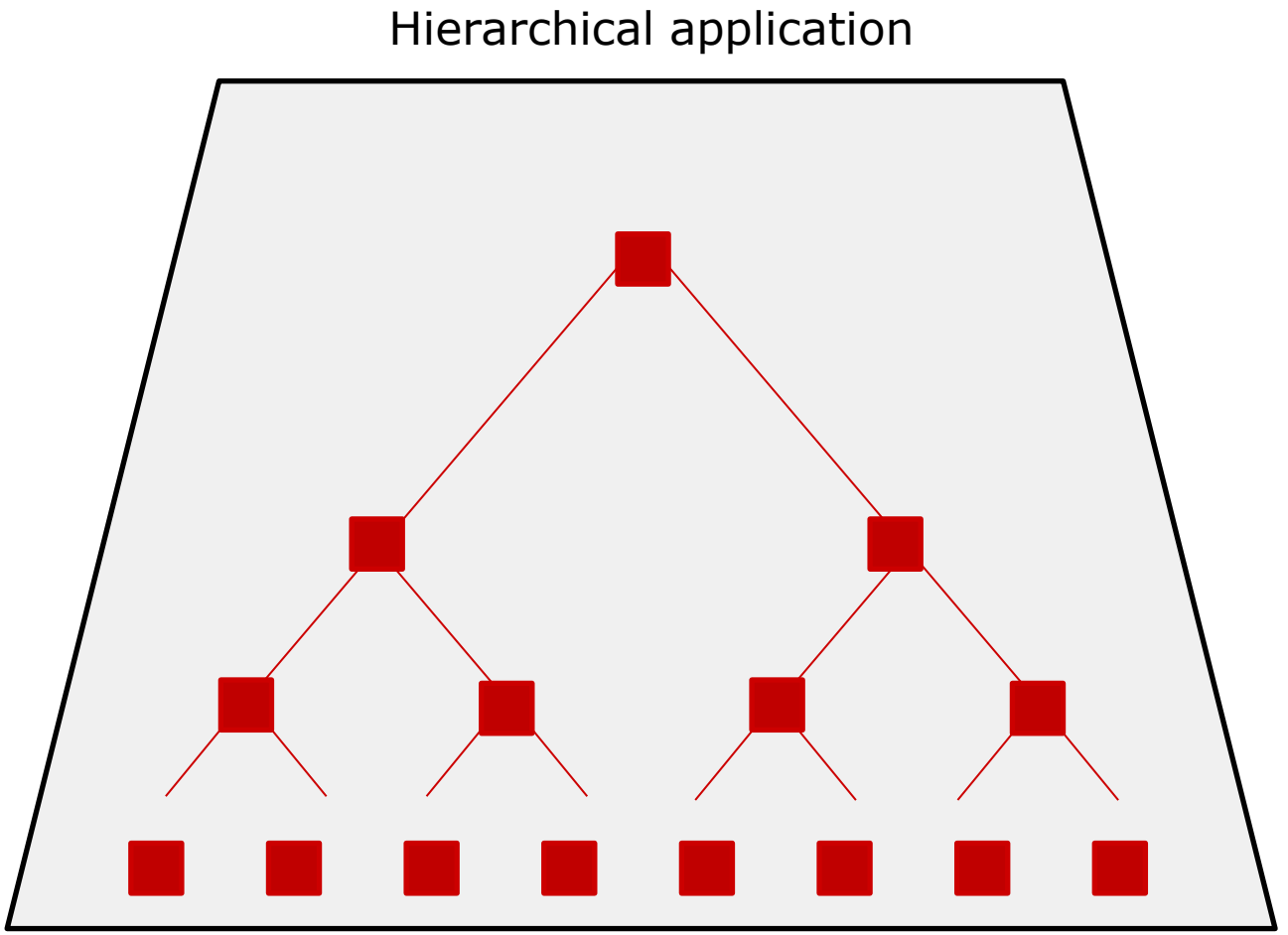
The mapping is direct

Actual mapping

many application nodes => a single platform node

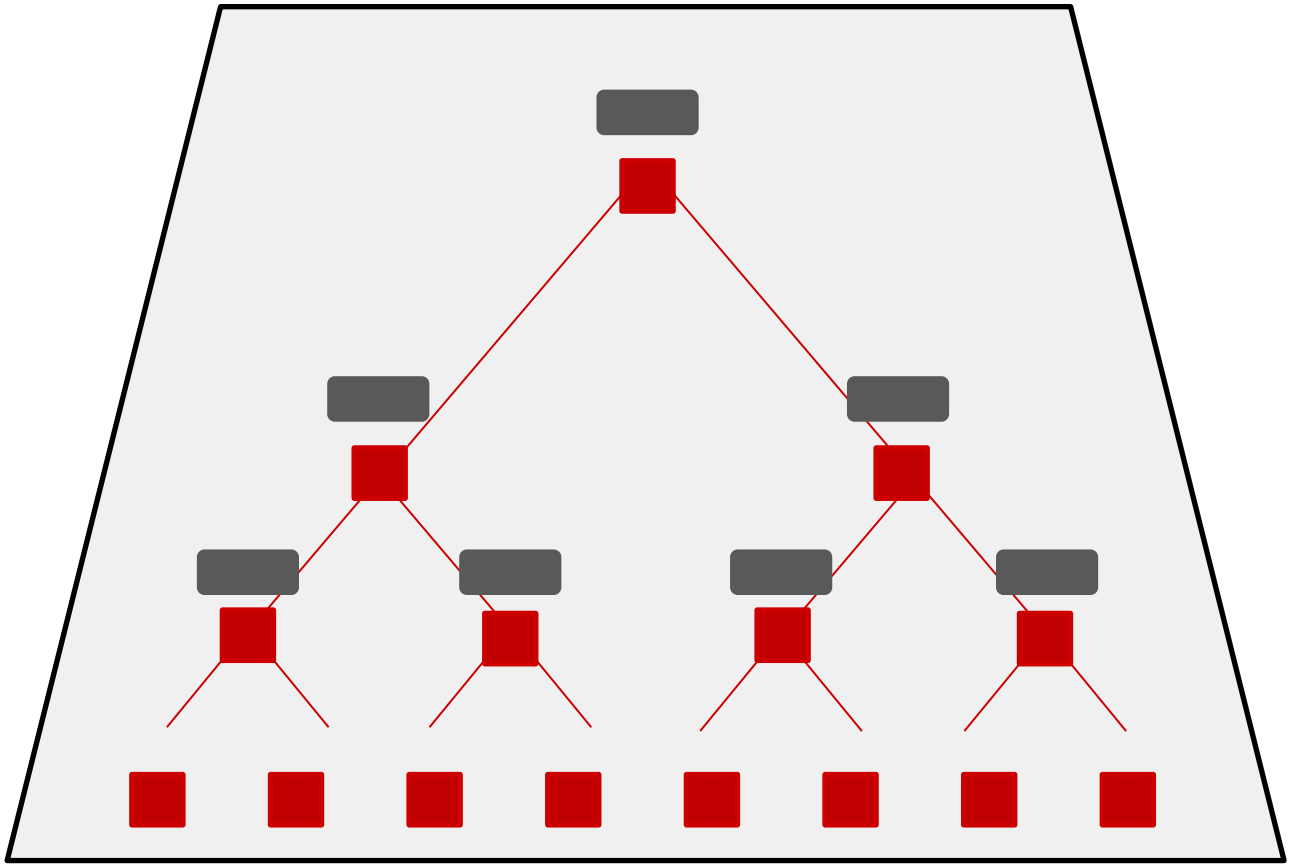


Hierarchical checkpoint/restart

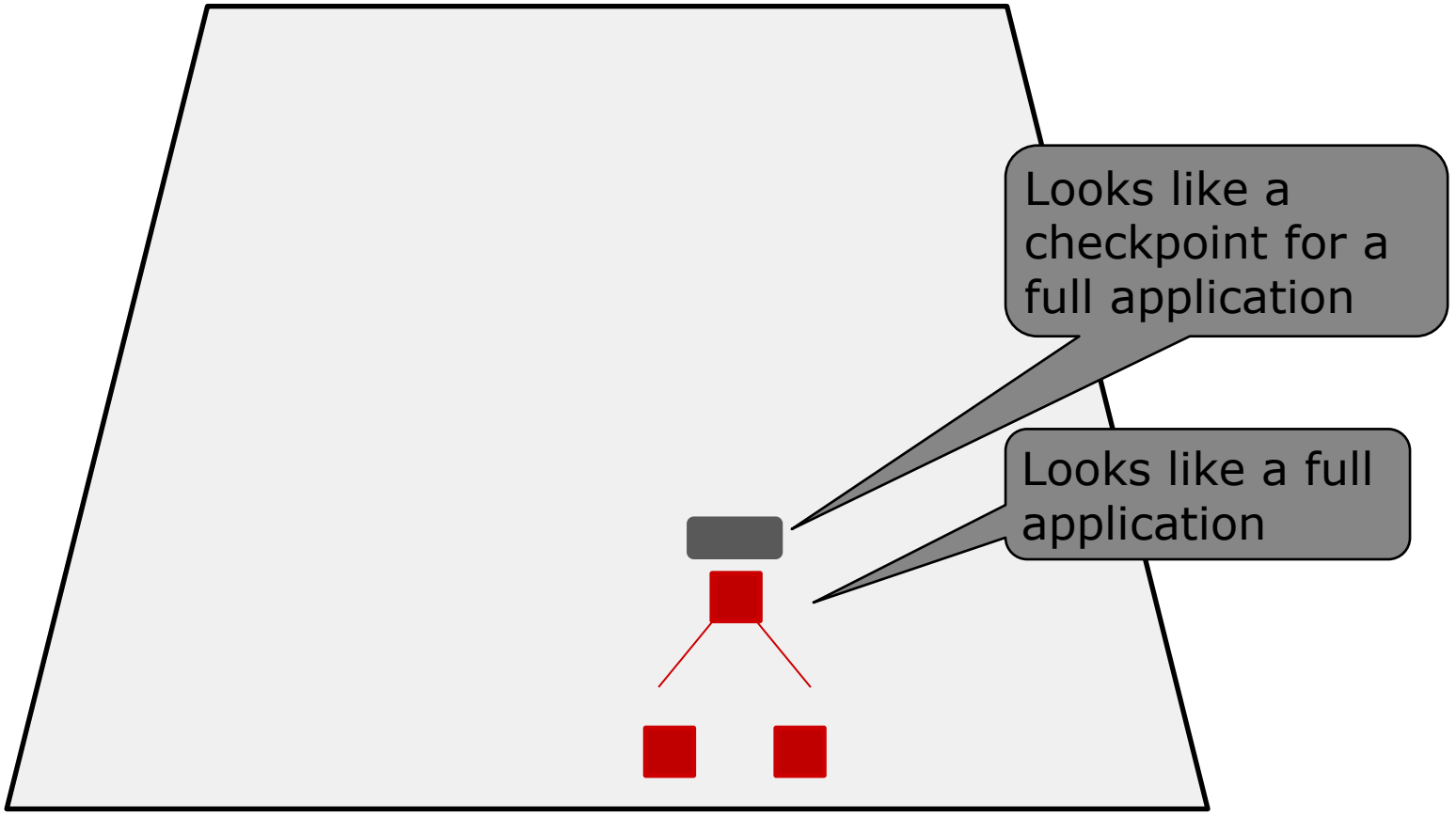


Hierarchical checkpoint/restart

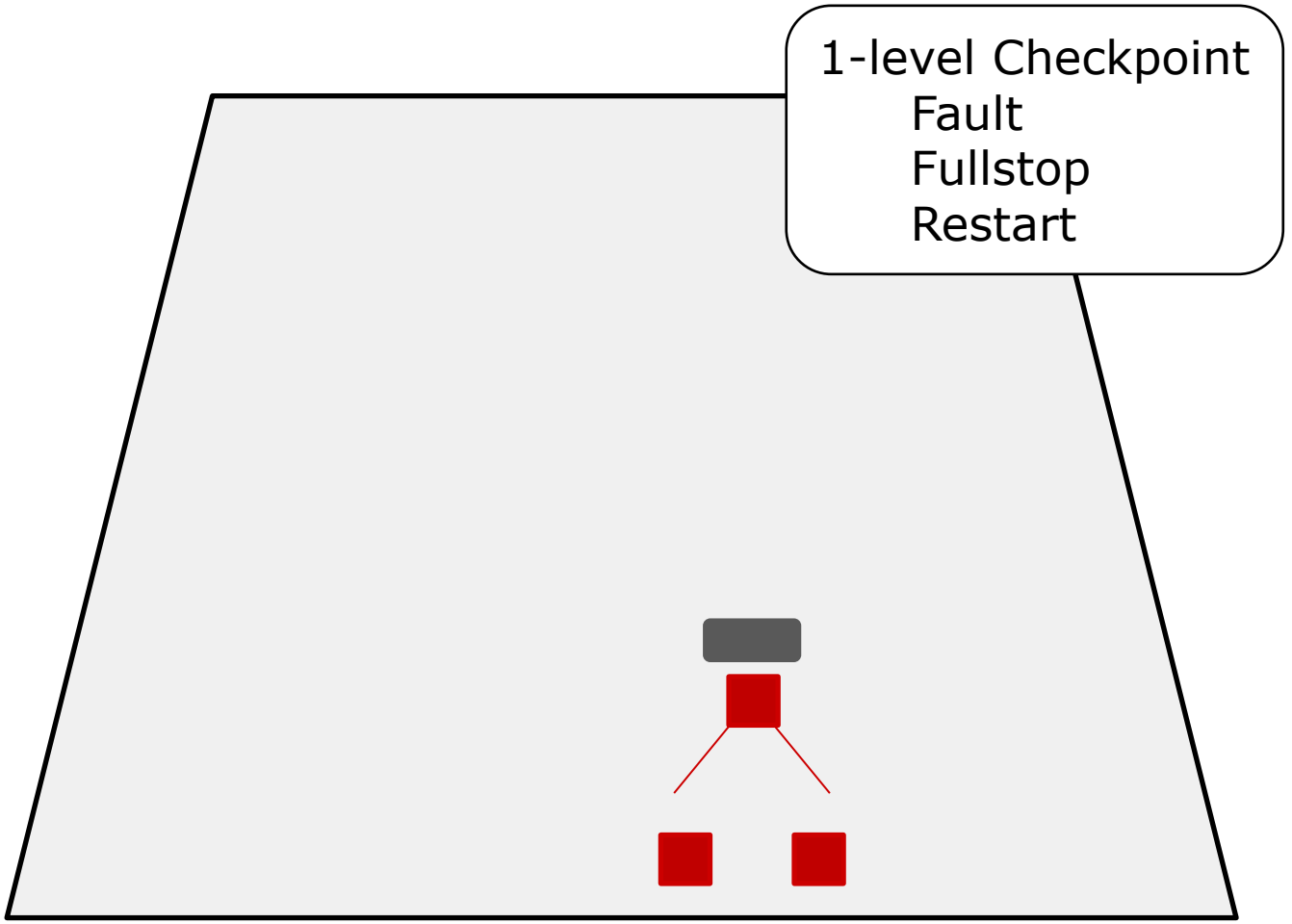
Checkpoint for a graph is held with its parent step



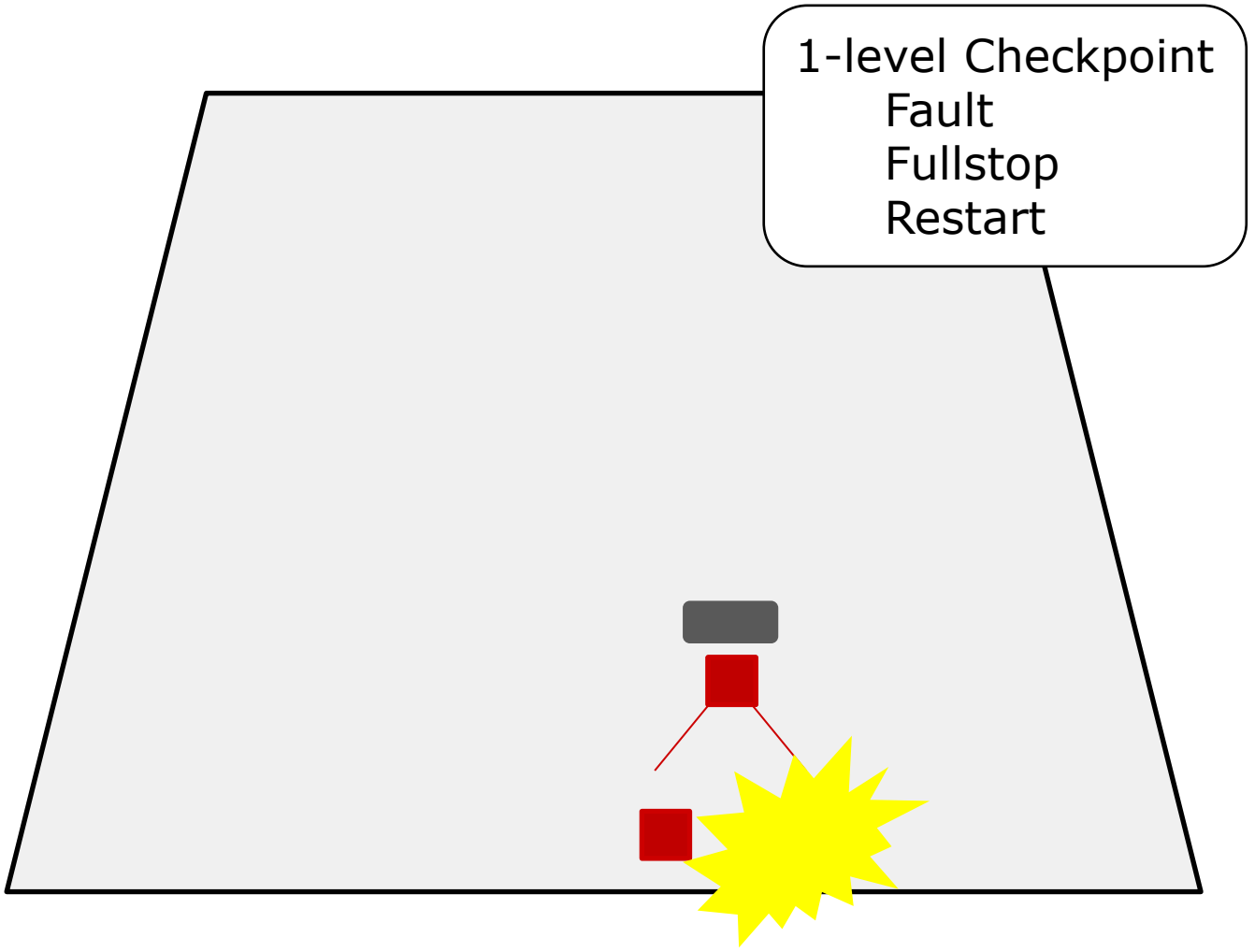
Hierarchical checkpoint/restart



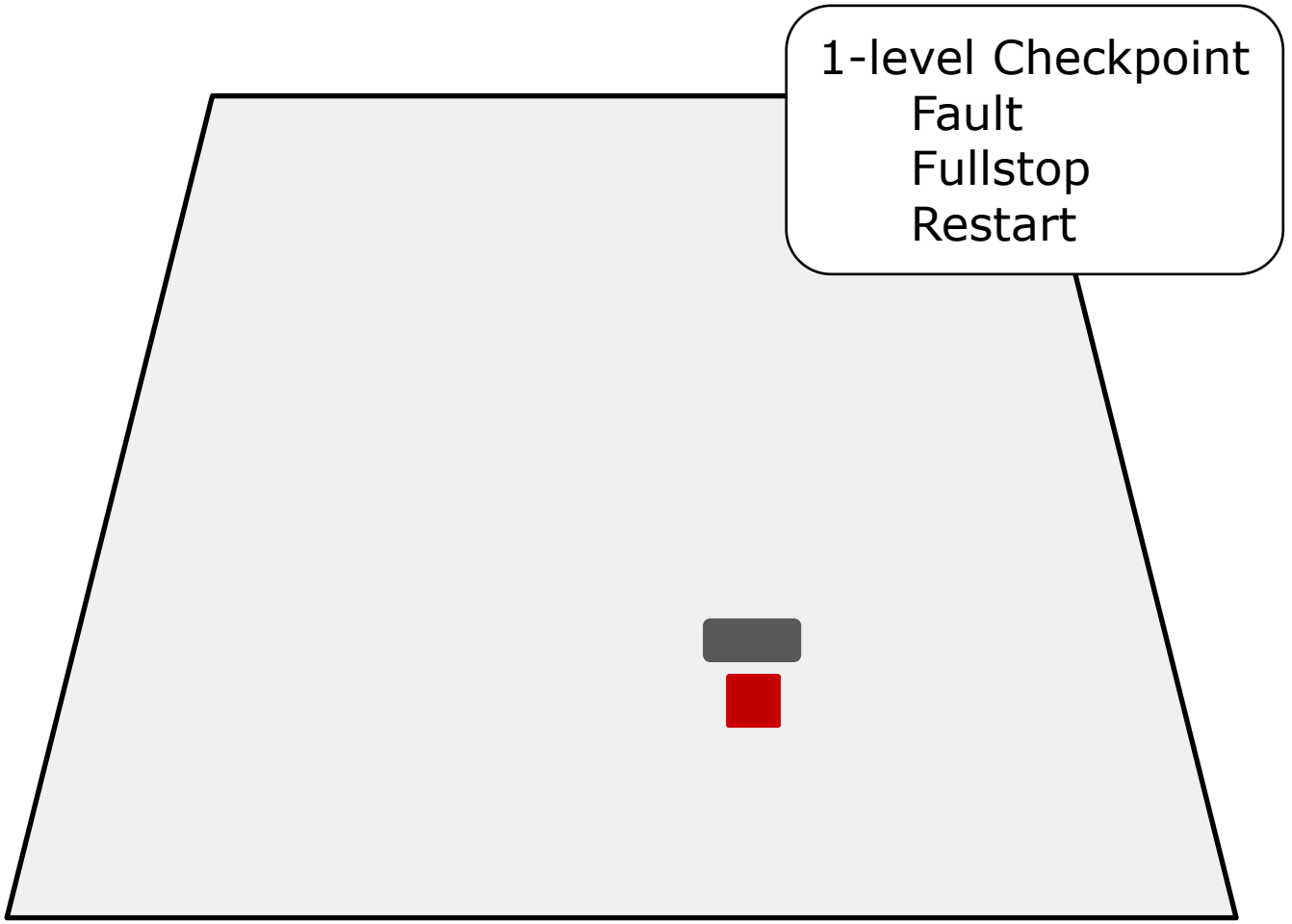
Hierarchical checkpoint/restart



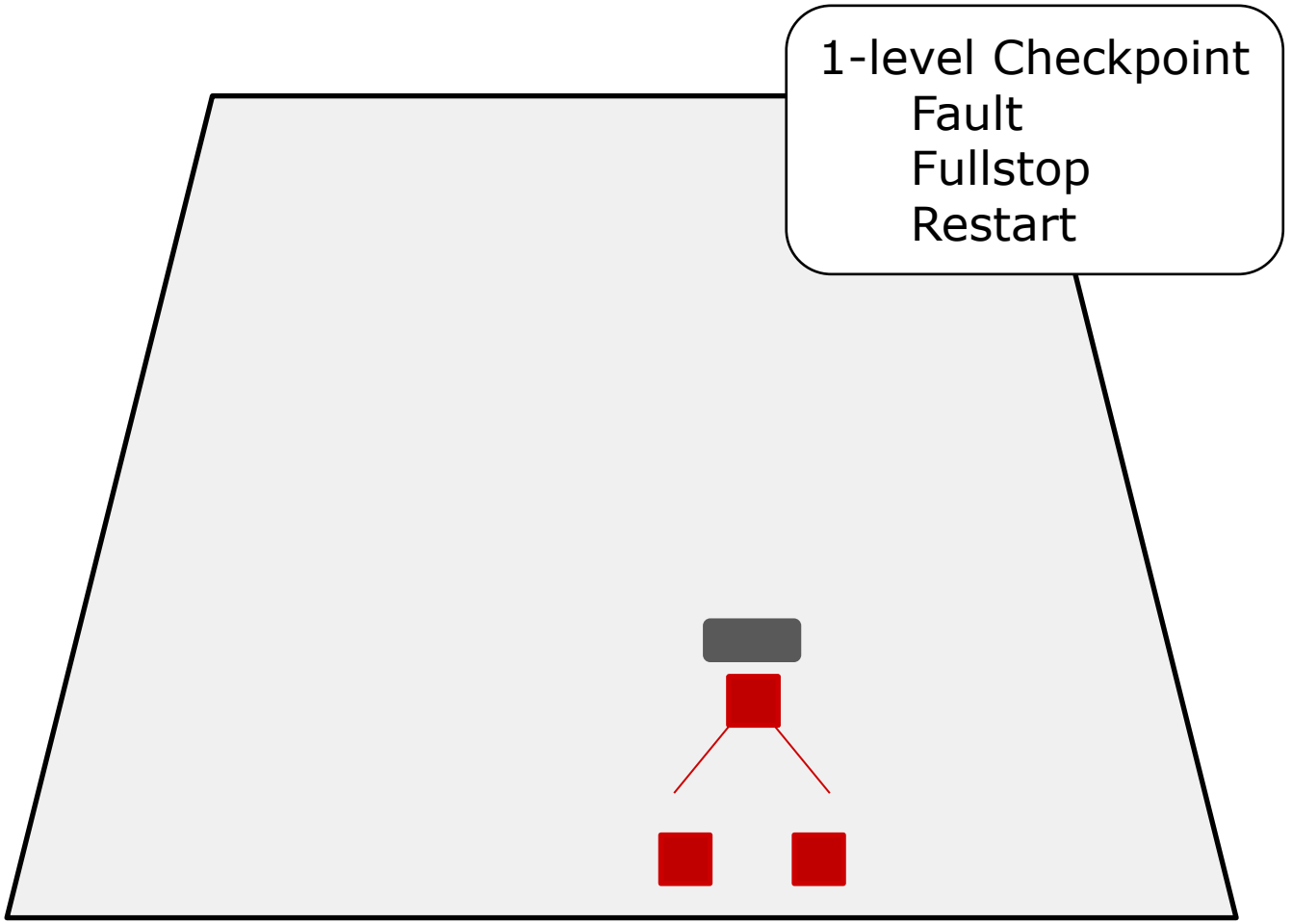
Hierarchical checkpoint/restart



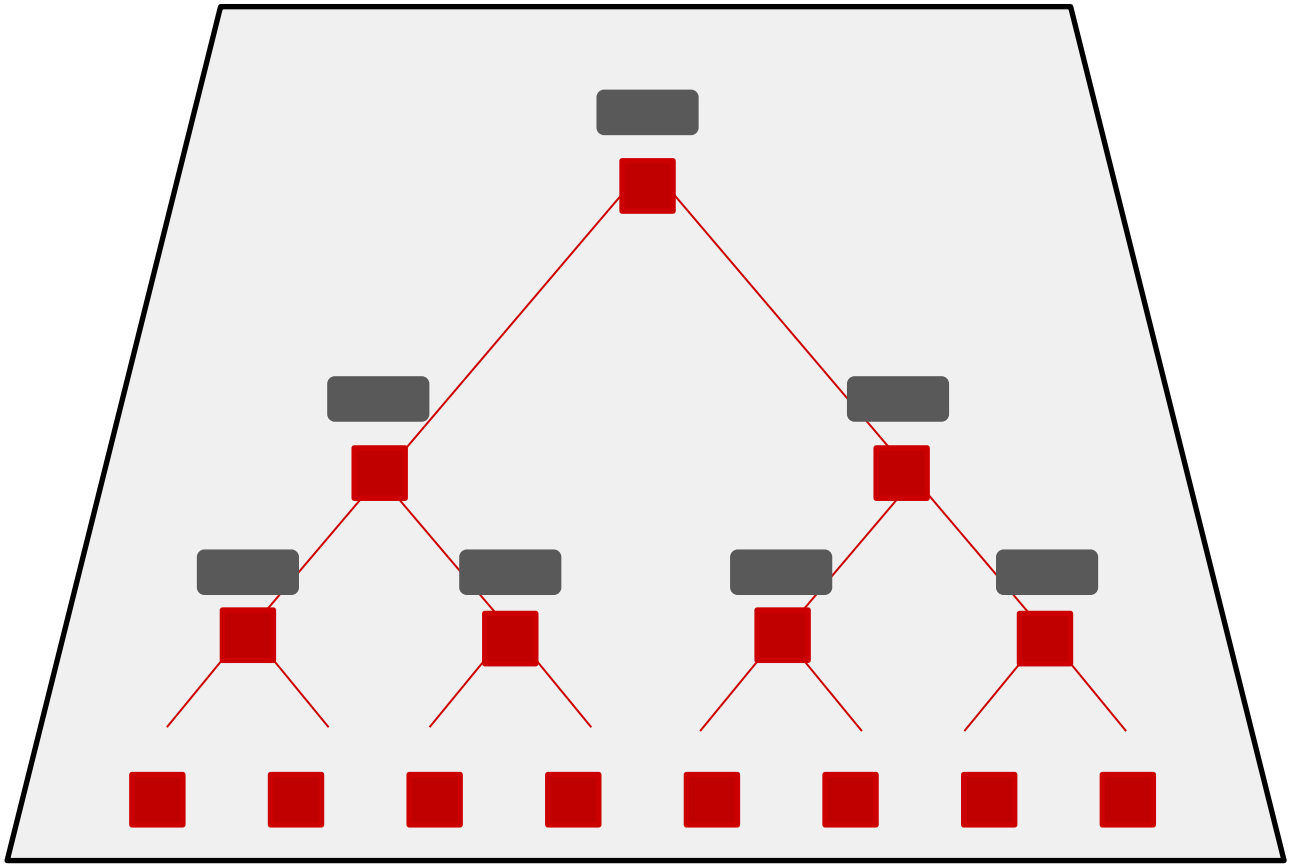
Hierarchical checkpoint/restart



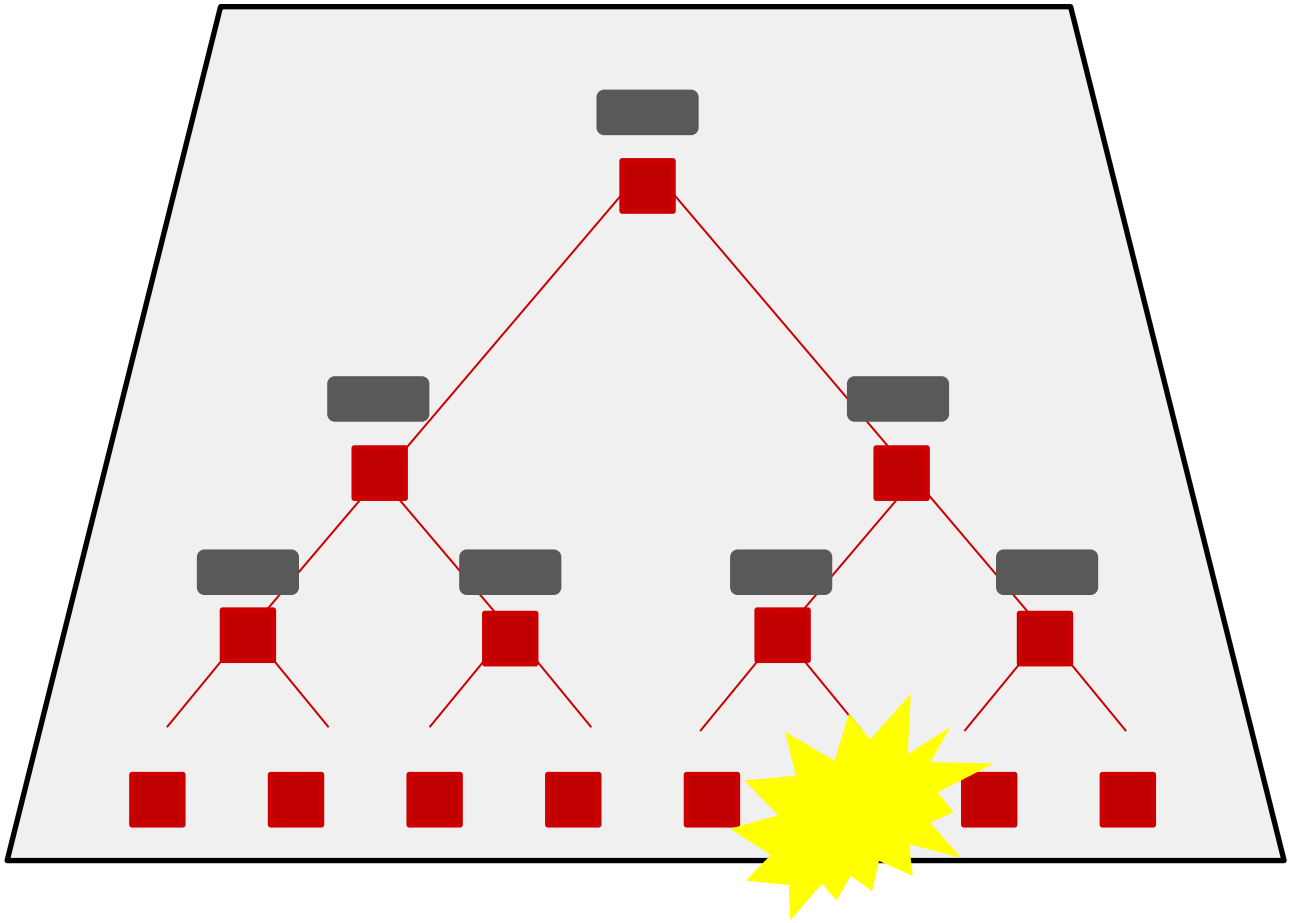
Hierarchical checkpoint/restart



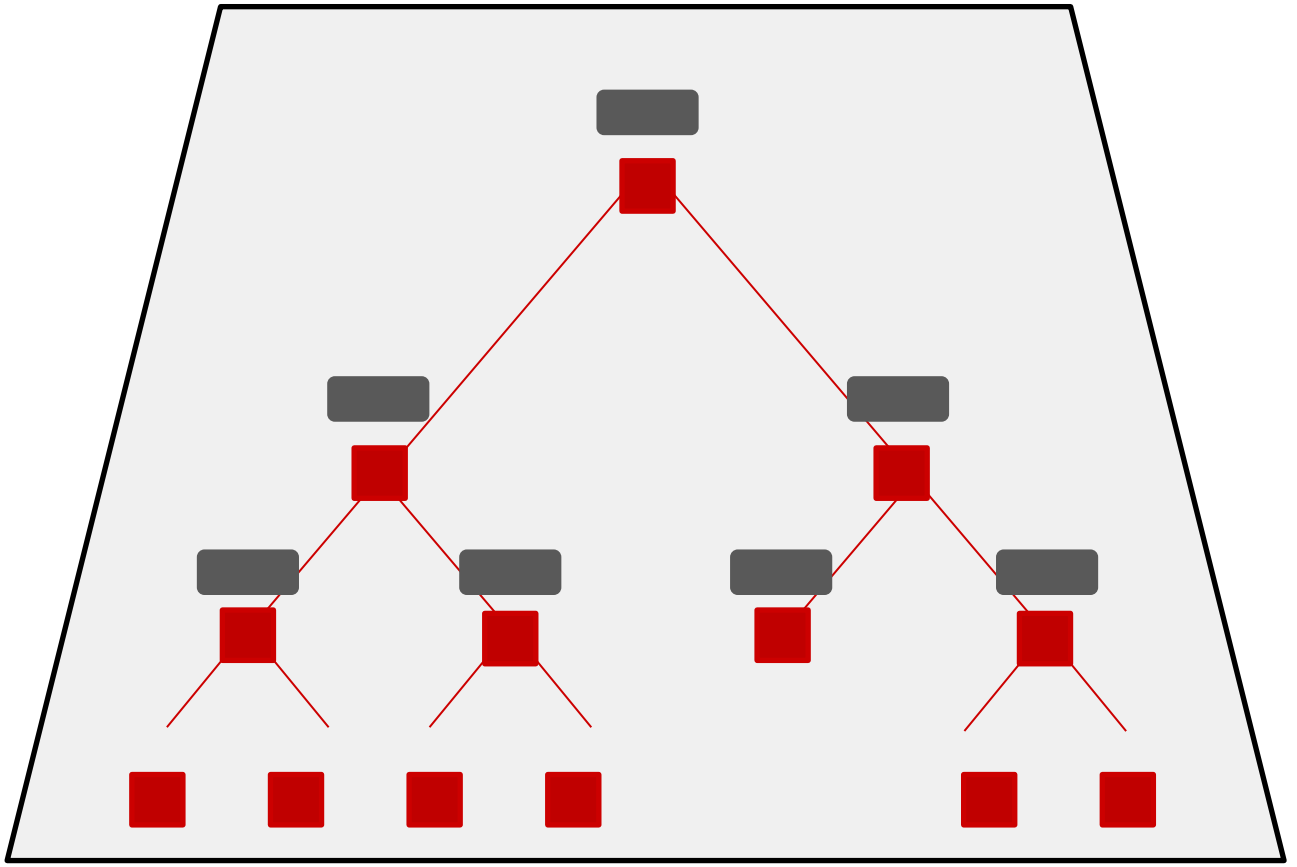
Hierarchical checkpoint/restart



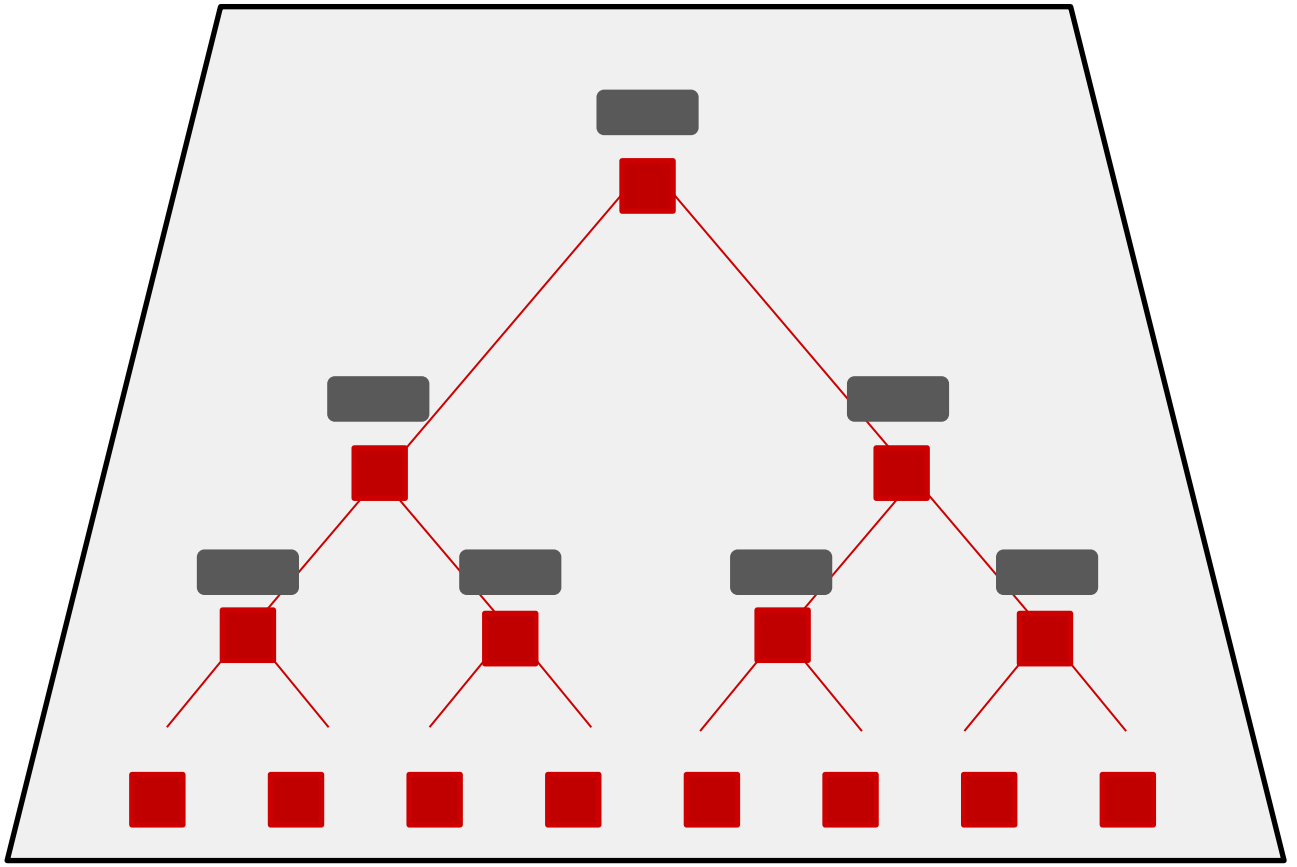
Hierarchical checkpoint/restart



Hierarchical checkpoint/restart



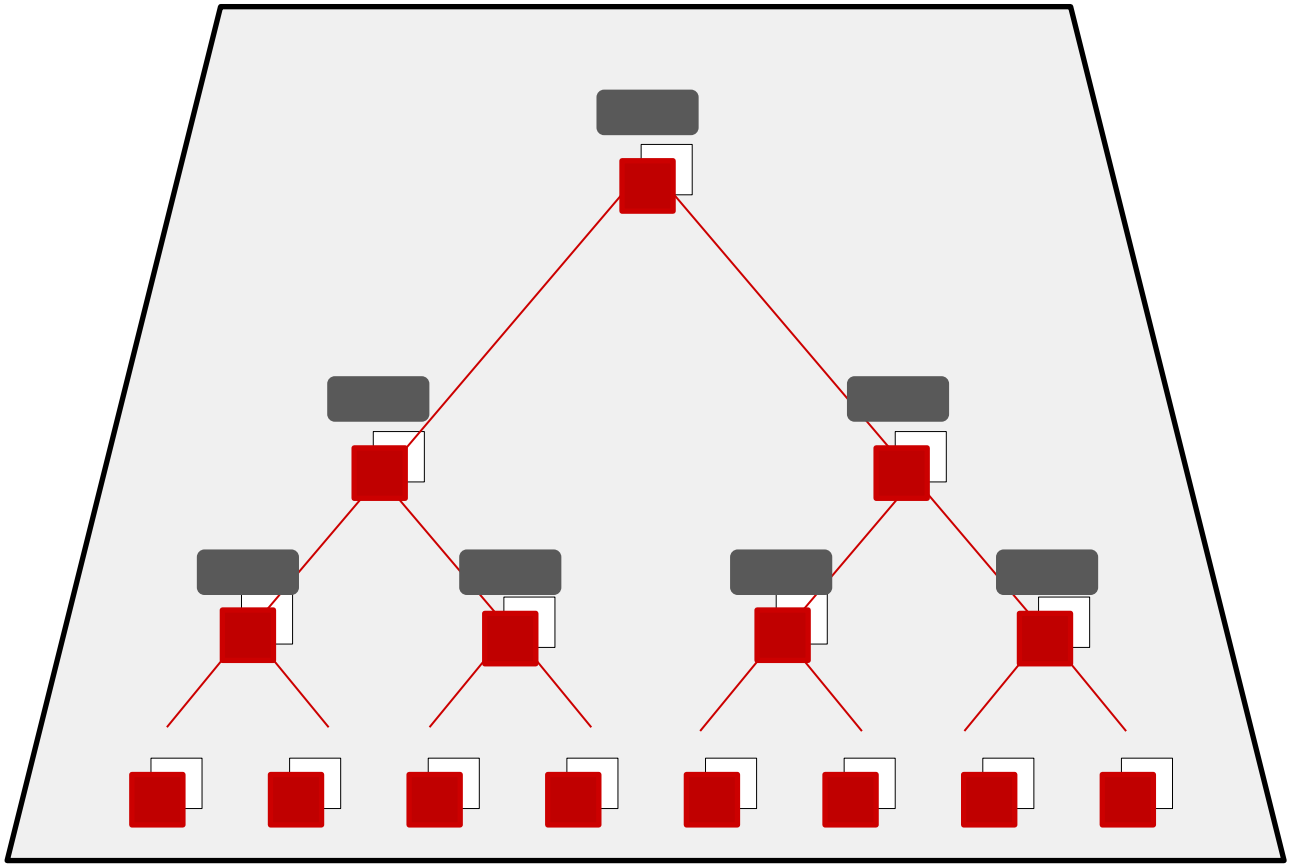
Hierarchical checkpoint/restart



From above: step simply looks like it took longer than expected.

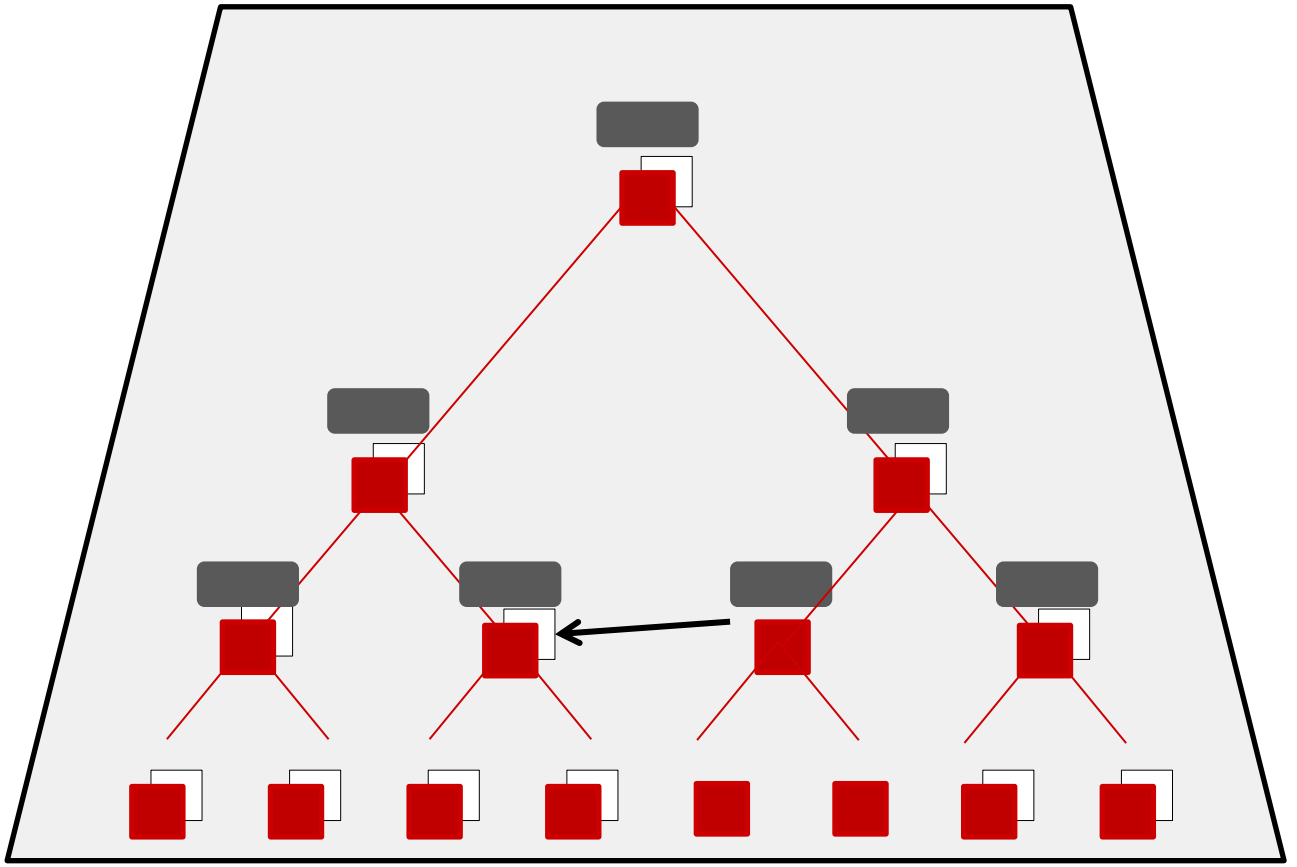
Checkpoint with full stop at one node looks like checkpoint/continue for the whole program

Hierarchical checkpoint/restart elsewhere



The application nodes are at real platform nodes

Hierarchical checkpoint/restart elsewhere



- Key: checkpoint is associated with the program hierarchy
  Not with the platform hierarchy
- Note that the move causes a locality glitch

Hierarchical checkpoint/restart: Summary

- Each hierarchical program node looks like a whole program
- Each hierarchical checkpoint looks like a whole checkpoint
- Can restart an application node from a checkpoint at any level
  - at another platform node
  - at a future time
- Checkpoint/stop/restart at a node

  => Checkpoint/continue of whole app

Adaptivity

- If we can move parts of an executing application to another *place* in the platform or to a later *time* because of failure

- We can move parts of an executing application because we choose to:
  - Power management
    - Power components on/off
    - Power components up/down
  - Self-aware computing
    - Modify mapping based on feedback
  - Change of goals
    - Between power and time to solution, for example

Analyzability

Characteristics that support analyzability

- Dynamic single assignment
- No side-effects
- Deterministic
- Optional tag functions and edge annotations
- Both control and data dependences are explicit
- Serializable

What might the analysis support

- Determine when data item is dead [DAMP 2009: Zoran Budimlic, et. al.]
- Optimize generic runtime [CPC 2013: Kath Knobe, Zoran Budimlic]
- Mapping computation steps to time/platform [PDP 2013: Frank Schlimbach, et. Al.]
- Mapping data items to memory (or to each other) [Europar2012: Dragos Sbirlea, et. al.]

Portability

- Memory
  - Single processor
  - Shared memory
  - Distributed memory (Intel)
- Heterogeneous platforms
  - CPU/MIC (Intel – not released)
  - CPU/GPU/FPGA (Rice)

Productivity

- Deterministic
- Separation of domain spec from tuning spec
- Potential for tools at the level the domain expert understands
  - Computation steps, data item in the domain spec (not processors, threads, …)

CnC on Intel's WhatIf site:

http://software.intel.com/en-us/articles/intel-concurrent-collections-for-cc

CnC on Rice's Habanero site:

https://wiki.rice.edu/confluence/display/HABANERO/CNC

CnC'13 workshop

Sept 23-24 Co-located with LCPC'13 in Santa Clara, CA

Call will appear on www.lcpcworkshop.org

Open CnC weekly meeting discussion

To get on mailing list send mail to kath.knobe@intel.com


intel
Software

Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

http://intel.com/software/products

Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804