Incorporating more of a large app for improved analyzability

CnC Workshop, October 14th, 2017

Kath Knobe (Rice University)

Zoran Budimlić (Rice University)

Current state
Pros and cons

Computation steps are required:

- To be deterministic
- To be terminating
- Not to be part of a co-routine with other steps

These constraints are not about scheduling or placement

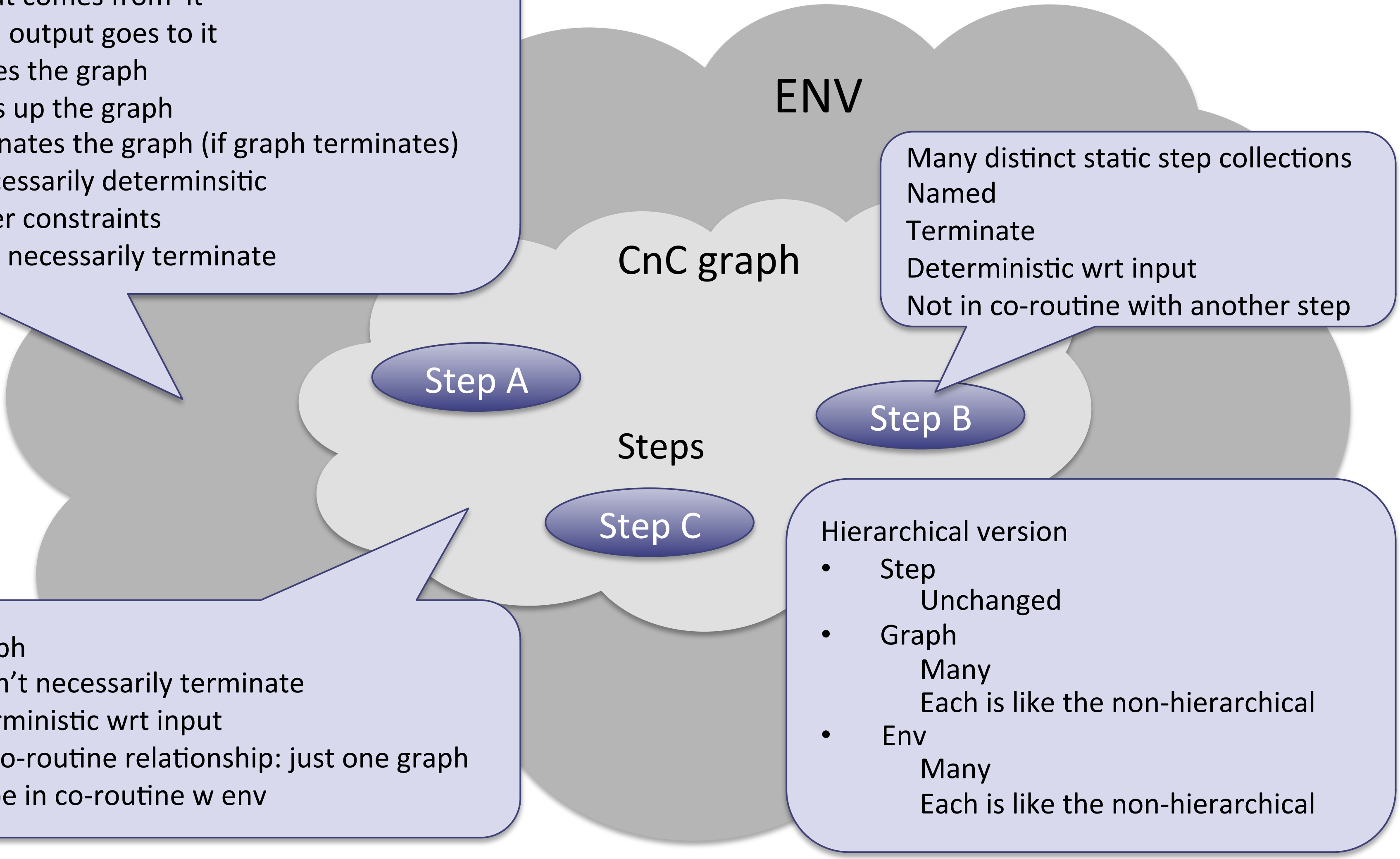
Because CnC obeys these constraints

+ it is highly analyzable and optimizable

- it only applies to some small apps or small pieces of larger apps

Current single graph

1 unnamed env
    Wrt co-routine relationship: just one env
All input comes from it
    All output goes to it
It creates the graph
It starts up the graph
It terminates the graph (if graph terminates)
Not necessarily determinsitic
No other constraints
Doesn't necessarily terminate



Many distinct static step collections
Named
Terminate
Deterministic wrt input
Not in co-routine with another step

1 graph
Doesn't necessarily terminate
Deterministic wrt input
Wrt co-routine relationship: just one graph
Can be in co-routine w env

Hierarchical version

- Step
  - Unchanged
- Graph
  - Many
  - Each is like the non-hierarchical
- Env
  - Many
  - Each is like the non-hierarchical

Classic flat perspective …

- We don’t know anything about the steps except what each produces and consumes. Each step is *opaque* (blackbox*).*
- The graph is *transparent (*analyzable/optimizable)*.*
- We don’t know anything about the env except what it produces and consumes. The env is *opaque* (blackbox).

… carries forward to hierarchy

Goals

- Include more of the full app
- Generalize existing work
- Extend the scope of optimization

Opaque or transparent

Old view

- The env is opaque
- The graph is transparent
- A step is opaque

New View

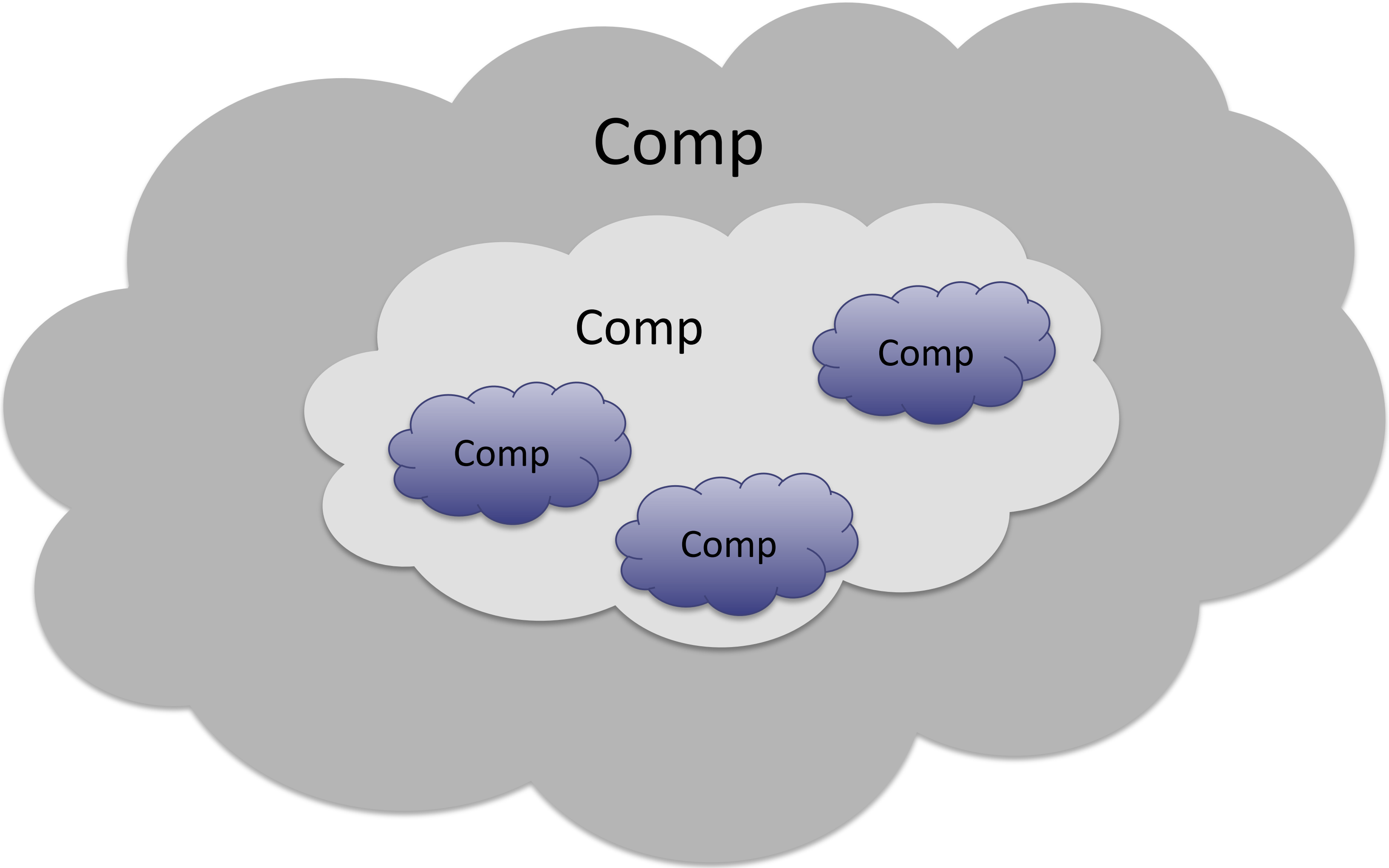
- Any computation can be transparent or opaque
- Terms transparent/opaque are *relative*:
  - **From** B
    A is transparent
  - **In this discussion**
    We're viewing A as opaque
- Might be a software engineering choice
  - System requirements and goals
  - Stage of development
  - Task at hand
  - …

Proposal: generalize to *computations*

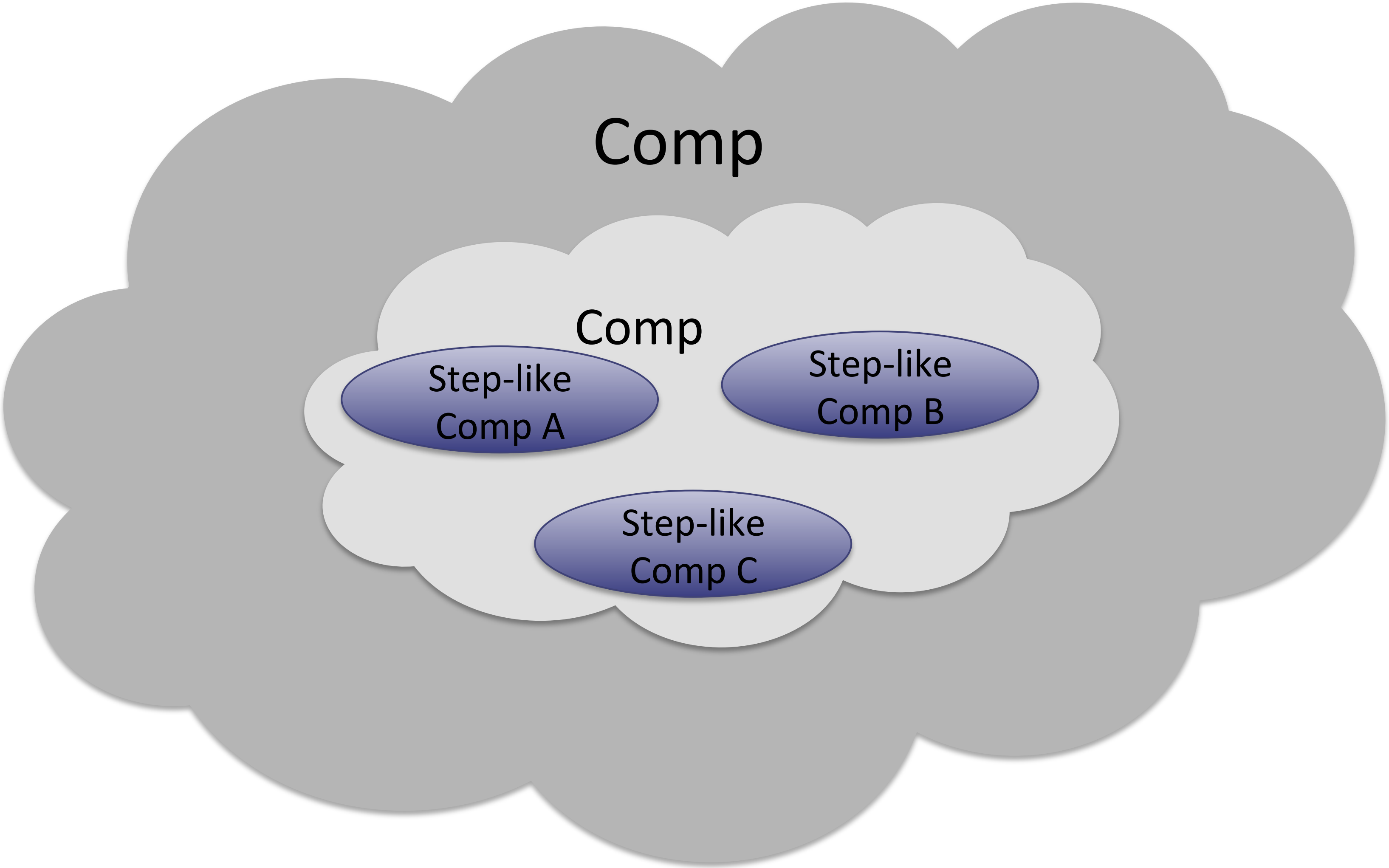
- Computations are
  - Producers and/or consumers
  - Named
  - Have no requirement to terminate
  - Have no requirement to be deterministic
  - Can be in co-routine with another computations
  - Any computation may be tagged

- Multiple distinct named computations might:
  - Replace our single unnamed env
    - Producing data/control
    - Consuming data/control
  - Appear within a CnC graph

Generalize to Computations



But with no loss of optimization potential



Comp

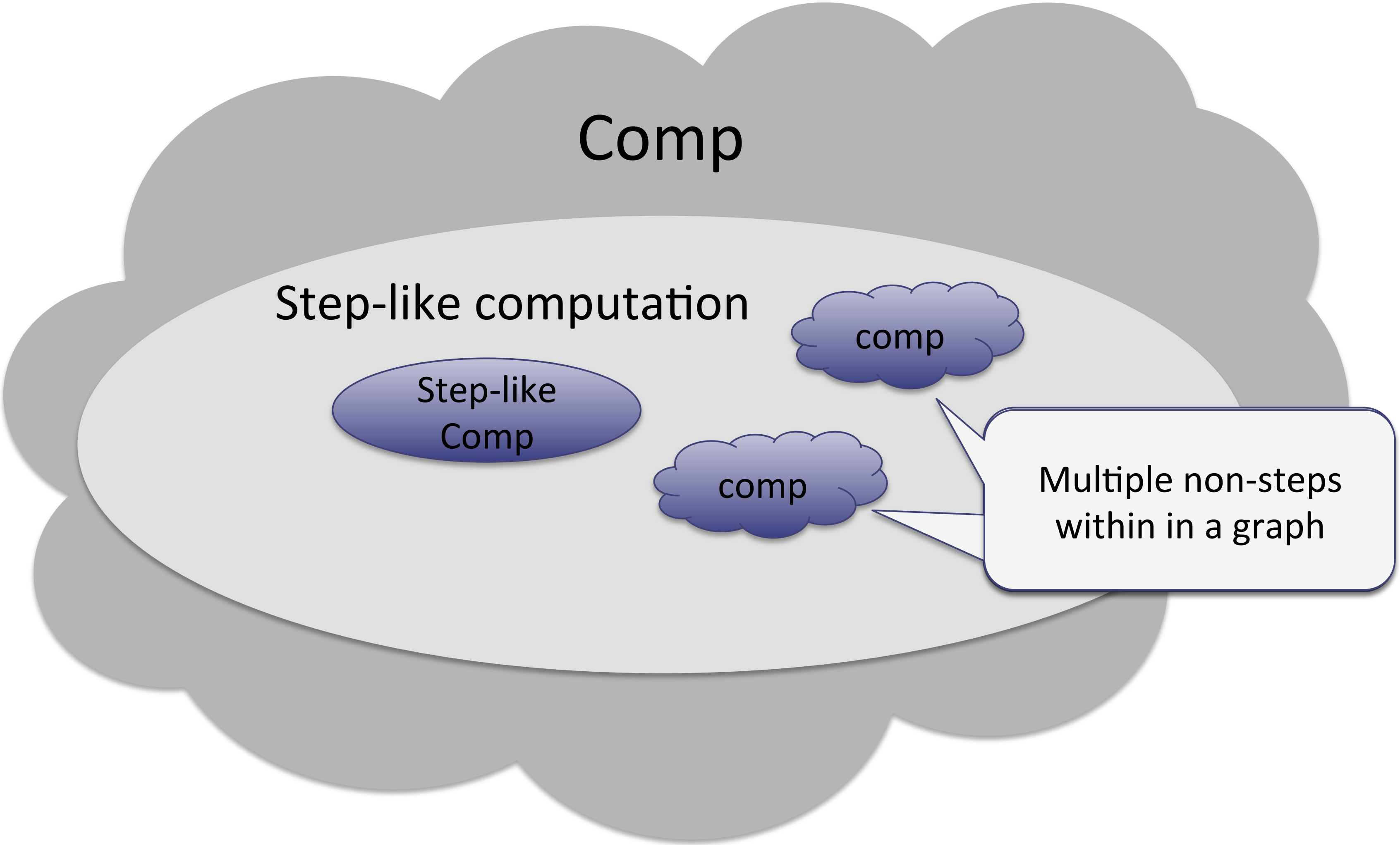
Comp

Step-like Comp A

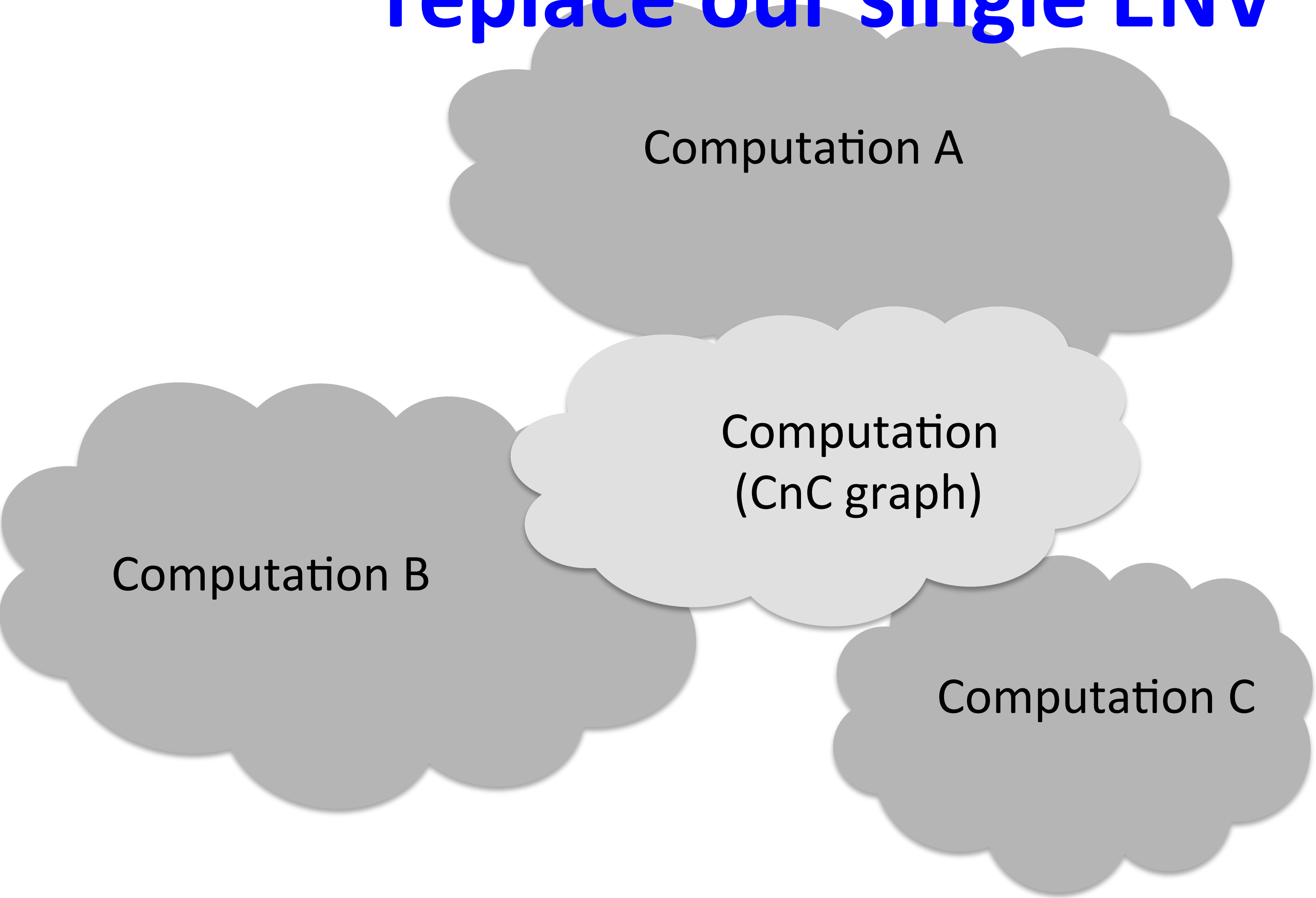
Step-like Comp B

Step-like Comp C

More flexibility: include more of the full application



Multiple computations replace our single ENV



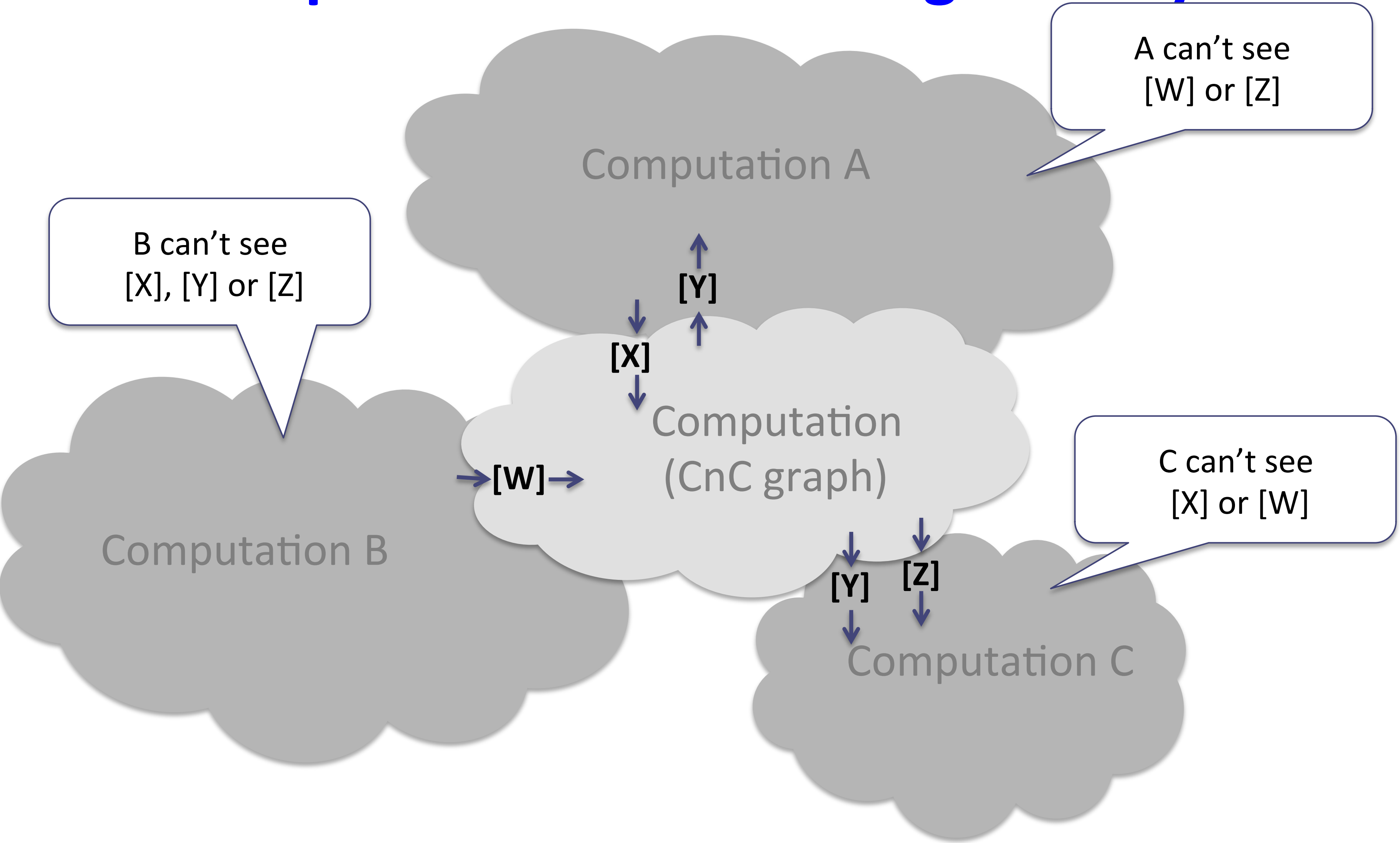
Computation A

Computation (CnC graph)

Computation B

Computation C

Non-step computations with distinct I/O: Improved understanding & analysis



Characteristics of non-step computations

- Even if they are not step-like we might be able to use whatever we know in analysis and optimization

- But for a non-step we might know
  - It terminates even if it's not deterministic
  - It's deterministic even if it might not terminate
  - It is deterministic and terminates but
    involved in co-routine

  Even if the non-step is *opaque*

- The normal attributes of a step
  - If it's tagged, it can be *control-ready, data-ready, ready, executed*
  - If computation isn't guaranteed to complete it may never become *executed*

Boarder crossing

Boarders

- Within a level of hierarchy

  One computation to another: step/step or graph/graph

- Across levels of hierarchy

  Step/graph

For separate development (libraries or within an app) allow renaming of the collections and reordering of the indices

- Arithmetic computation (dependence functions)

  (foo: j, k) does a put of [x: j, k] and

  (bar: j, k) does get calling the same instance [x: j, k+1]

- For libraries or separately developed components even within the same project.

  (foo: j, k) does a put of [x: j, k]

  (bar: j, k) does get [y: k+1, j]

Non-step computations might have some step-like attributes

- If a computation terminates, is deterministic and isn't involved in co-routines, it is a step.
- But a non-step computation may have *some* attributes. For example
  - Might be known to terminate even if it's not deterministic
  - Might be deterministic even if it's not known to terminate
- Suppose: we associate with non-step computations the subset of attributes that actually do apply
  - Even if the computation is opaque these might be useful for analysis/optimization
  - Need to extend the rule for attribute propagation involving non-steps
  - Need to create rules for propagating properties (deterministic, …) in the hierarchy

Legal transformations on non-step computations

- Decomposition styles on non-step computations
  - If a computation is tagged it might be homogeneously decomposed
  - If the child of a node is a graph the node can be heterogeneously decomposed
- Transformations
  - We are now allowing co-routines. We can transform a step-like computation into 2 computations that result in a co-routine.
  - Merge of 2 nodes
    - If each terminates, the merger terminates
    - If each Is deterministic, the merger is deterministic
  - If there are no co-routines among the components, its components can be serialized
- Can we incorporate "constraints on hierarchy" work into this view?

Advanced

Creation, I/O, destruction

- When we talk about CnC we often assume the CnC graph exists.
- In our current systems some non-CnC component, called env, creates it, provides input, starts it up, receives outputs and shuts it down.
- How that's done hasn't been really part of CnC itself and varies among systems.

Already assuming significant support for universal CnC

- Classic flat CnC
  - Names are statically known but
  - Indices may be : statically known, input, computed by the application
- Hierarchical CnC
  - Homogeneous decomposition
    - The "name" at some level in the hierarchy includes what looks like an index above.

      ["x": 3] decomposes to … , ["x, 3": 4], …
    - Note: the "3" above might be statically known, input or computed from input data
    - This implies

      The "names" might be: statically known, Input or computed from data

      New instances of a statically known graph can be dynamically computed

      Why not allow graphs be: input or computed from input data?
- The top level our any hierarchy is identical to that for our Universal CnC app
- We might be moving closer to support for Universal CnC app
  - The CnC spec itself is input to universal CnC and executed.

Next

- This was all about computation.
- We need to support data that isn't single assignment.

Conclusions
Claims/hopes

- By including non-step computations
  - We allow inclusion of more of the customer app to be analyzed and optimized
- By identifying static step characteristics (deterministic, …) that apply to non-step computations
  - We can incorporate them into analyzes and optimizations
- By applying dynamic step attributes (data-ready, …) to non-step computations as appropriate
  - We can take them into account to make better scheduling and placement decisions

Future

- Evaluate the general idea in the context of a real (but small) app

  One that includes co-routines, non-determinism, non-terminating computations

- Investigate legal analyses and optimizations
- Implement: first in our flat version
- Update the constraints on hierarchy work to incorporate these ideas

END

One more kind of “computation”

- for dependence functions within a level or at the transition from one level in the hierarchy to another
  - A grain change: Coarse/Fine
  - Names might be altered
  - Tags components might have distinct names and order
  - The value of a tag component in one might be a function of the corresponding tag component in the other

- What are its possibilities for these computations?
  - Probably should terminate, & not involved in co-routines
  - Deterministic?
  - Opaque? Transparent?
  - Might be generated from a spec
  - …

Flatten hierarchy

- The conversion between grains in hierarchy is now explicit
  - There are ordering constraints wrt parent and child
  - The call/return looks like an arbitrary constraint
  - We could flatten them to remove that constraint
- It's really 3 ordered but distinct computations, each could be placed and scheduled
  - The coarse-to-fine conversion
  - The lower computation itself
  - The fine-to-coarse conversion
- This grain changing code could be useful even in a flat graph

OR among non-step computations

All must have

- As before:

  same i/o signature

- What about same optimization characteristics??

  deterministic, terminating,…