Beyond Array-based CnC Apps

CnC Workshop, October 14th, 2017

Kath Knobe (Rice University)

Zoran Budimlić (Rice University)

Current state

- Consider a presentation at a large conference:

```
X(row, col, iter) = Func(
        X(row, col, iter-1),
        X(row+1, col, iter-1),
        X(row, col+1, iter-1),
        X(row-1, col, iter-1),
        X(row, col-1, iter-1)  )
```

- Everyone knows what this means
  - Domain experts can read & write
  - Optimizers can optimize
  - Runtimes can execute

- The array-based community is well served by + and −

+1 and −1
are about semantics of arrays

- They are not about how arrays are stored
  - e.g., row-major, col-major, space-filling curves, double buffering the iterations, …
- This semantic level communication in array-based apps used:
  - In domain-specs
  - In analyzers
  - In optimizers
  - In runtime

Tree-based apps are everywhere

- Tree-based applications might use pointers
- Runtimes can execute but …
  - Harder to write
  - Harder to read
  - Harder to analyze
  - Harder to optimize

We need to create a new situation for trees

The goal is

- To address the semantics of trees
  - not about how trees are stored
- So the tree-based community is also well served
  - Domain experts can write
  - Domain experts can read
  - Optimizers can analyze and optimize
  - Runtimes can execute

Basic idea 1:
Step vs dependence functions

Current thinking:

- Steps
  - CnC knows about the role of steps
  - Steps and domain-specific libraries of steps are (will be) reusable among set of domain-specific apps
- Dependence functions
  - CnC knows about the role of dependence functions
  - Dependence functions and domain-specific libraries of dependence functions will be reusable among set of domain-specific apps
- Step code and the dependence functions from application writer are passed to the analysis/optimization/execution phases

New thinking:

- Steps
  - Unchanged
- Dependence functions
  - *Dependence functions* and domain-specific libraries of dependence functions: are also be reusable among set of domain-specific apps
  - CnC knows their role but doesn't know the computation

Basic idea 1

- Current thinking: Array
  - *Array*-specific analyses and optimizations of CnC graphs*:*
    Can be reusable among set of array-based applications and implementations

- New thinking: Domain
  - *Domain*-specific analyses and optimizations:
    should also be reusable among set of *domain*-specific applications and implementations

  - *Tree*-specific analyses and optimizations:
    Should be reusable among set of *tree*-specific applications and implementations

Basic idea 2

Current thinking

- CnC knows about
  - the role of steps
  - the role of dependence functions
- Step code and the dependence functions from application writer are passed to the analysis/optimization/execution phases
- Reusable libraries (current thinking)
  - Of step codes
  - CnC graphs

Future thinking

- Reusable domain-specific libraries also includes
  - Dependence functions

New role: Community-expert

- Add to existing roles of domain-expert and tuning-expert
- What is a community?

  Community has
  - A shared set of computations (reusable steps and subgraphs)
  - A shared set of data structures (arrays, trees, …)
  - A shared set of dependence functions
  - A shared set of analyzes & optimizations on the community structures

  Community is not
  - A specific CnC application
  - All CnC applications
  - A specific way of storing the data structures
- CnC array-based community is already within our (subconscious) thinking

  Implied: shared set of dependence functions, e.g., +1, -1, north, nextIteration, …
- Proposal:

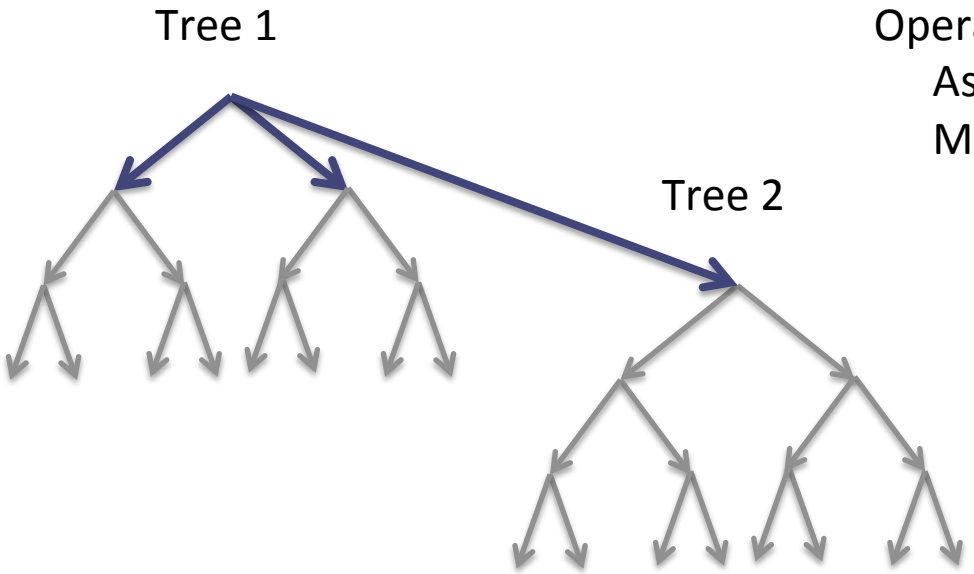
  To provide some support for tree-based community

  So we need to
  - have a better understanding of the role of community, in general
  - develop specific support for a tree-based community

Initial work

- An approach (or a set of approaches) to tagging nodes in tree-community
  - Intuitive
  - Optimizable
- Domain-specific dependence functions applied to tree-node tags.
  - For arrays:
    - +1, -1, applied to row, col, iteration or
    - NextRow(), prevIteration()
  - For trees:
    - Parent(), firstChild(), nextSib(), nextTree(), …
- Might want a community-specific algebra [Zoran]
  - Parent(firstChild(n)) is n
  - FirstChild(parent(n)) may or may not be n
  - NextSib(previousSib(n)) = previousSib(nextSib(n)) unless n is the first or the last child

Sequence of operators on binary trees: 3 dependences



Operator

Assumed to be the same within a tree

Might be different among trees

A tiling can follow

- Any one dependence
- Any pair of dependences
- All three

(de)composition on trees

Assume:

A given tree is a given operation.

Distinct trees might have distinct operations.

	Data	**Computation**
Homogeneous The children: the same code Within a tree	Arrays Trees	Like SPMD
Heterogeneous		

Among trees: similar to heterogeneous hierarchical decomposition

Assume:

A given tree is a given operation.

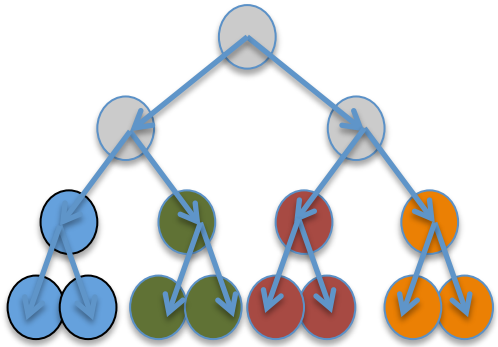
Distinct trees might have distinct operations.

	Data	**Computation**
Homogeneous		
Heterogeneous	Like a struct	Like MIMD

The children: different code Among trees

Example of possible tiling within a single tree.

This is the tree analog of an array tile
Within a tree => homogeneous tiles



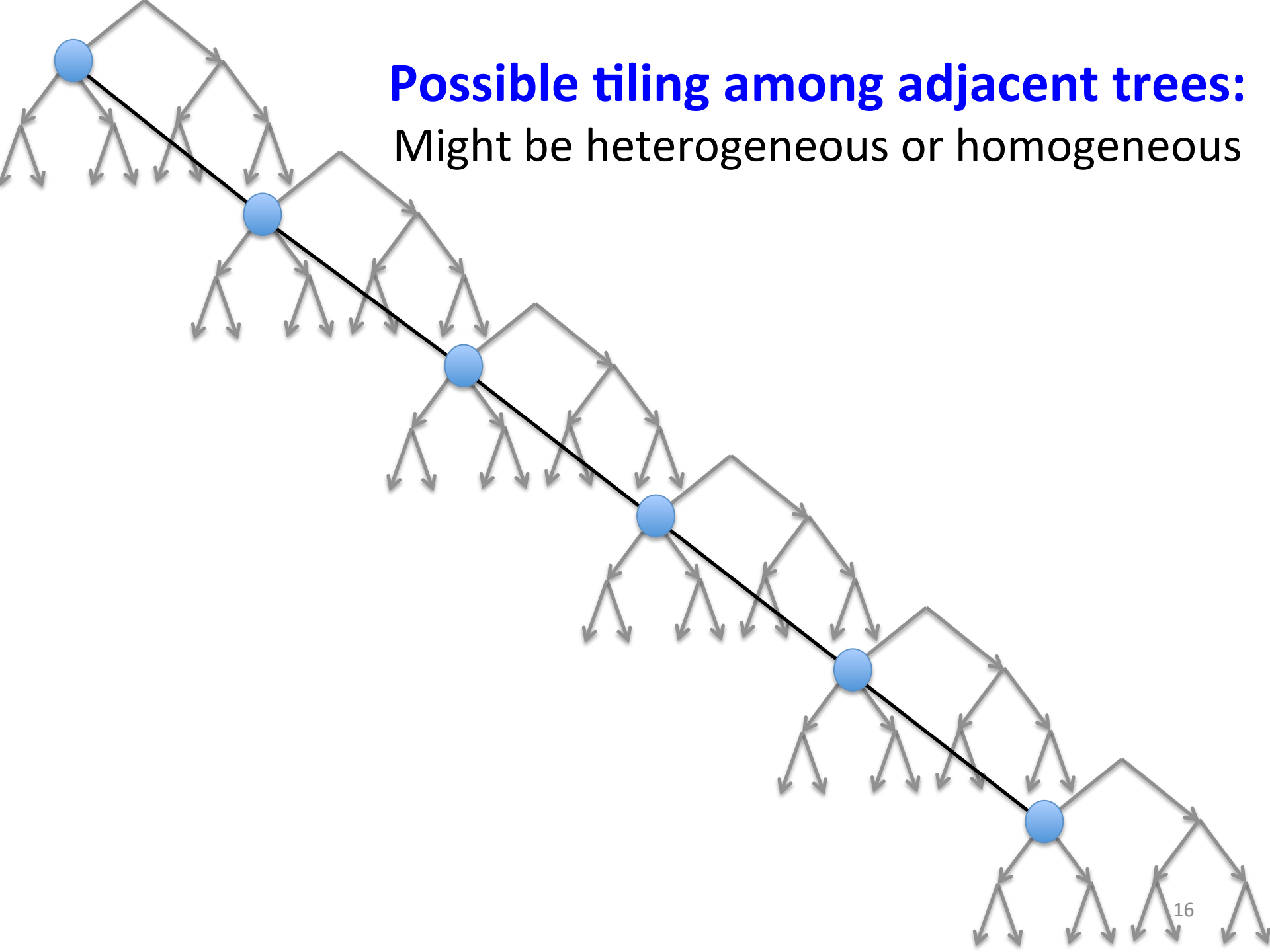
Heterogeneous/homogeneous

Conjecture:

- Doesn't impact tagging, e.g., <nodesInTree, treeIDs>
- Does impact the code for the steps
  - Homogeneous: has the code for one operation
  - Heterogeneous: has code for multiple operations

Possible tiling among adjacent trees:

Might be heterogeneous or homogeneous



Next

- Develop a way of tagging tree steps and items
- Agree on a set of tree dependence functions

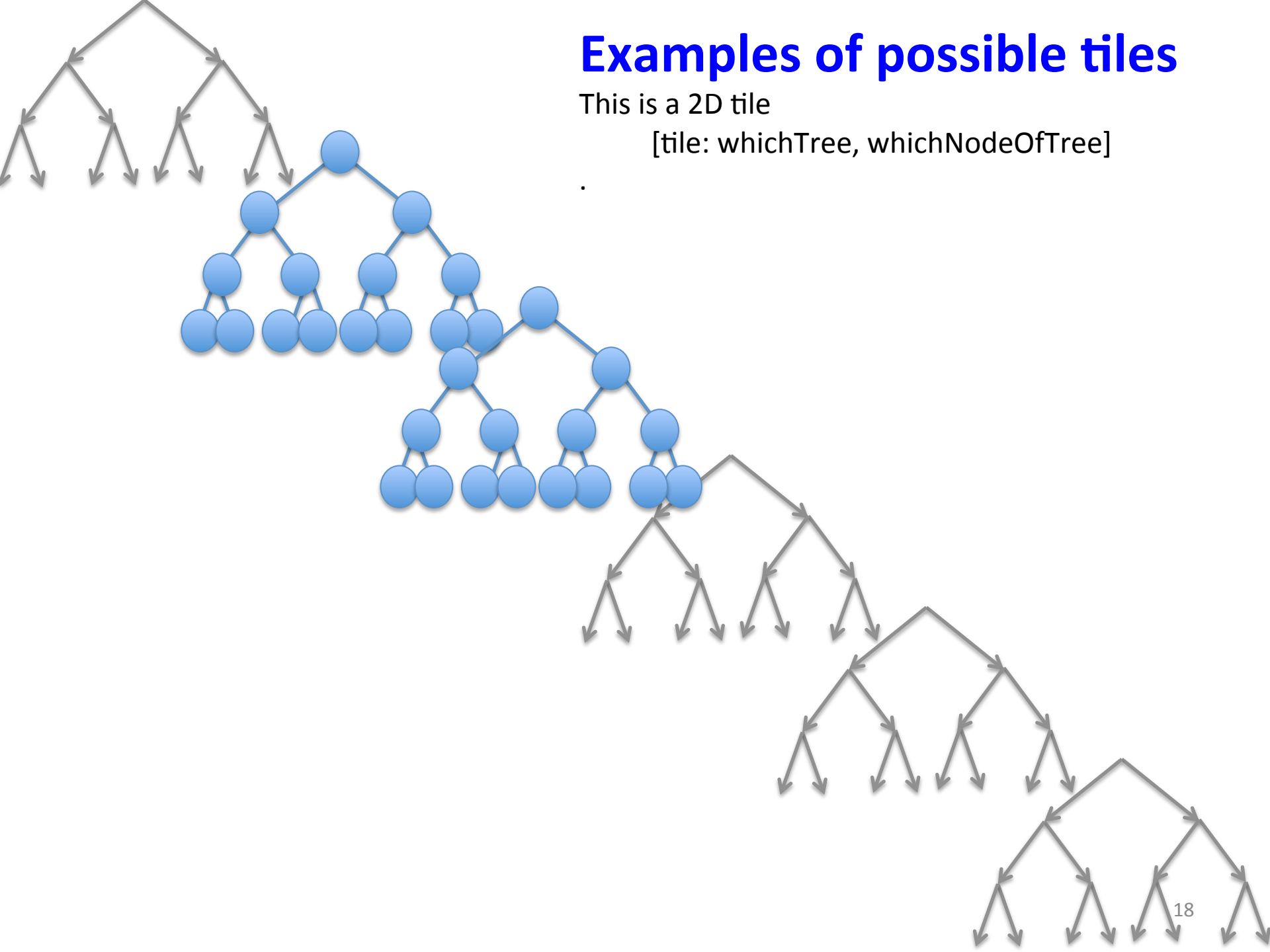
  E.g., LeftChild, RightChild, Parent, NextTree, PrevTree
- Develop ways of executing heterogeneous tiles
  - Alter step code
  - Static scheduling of the set of steps, each homogeneous
  - …
- Develop a variety of walks using various tilings

Examples of possible tiles

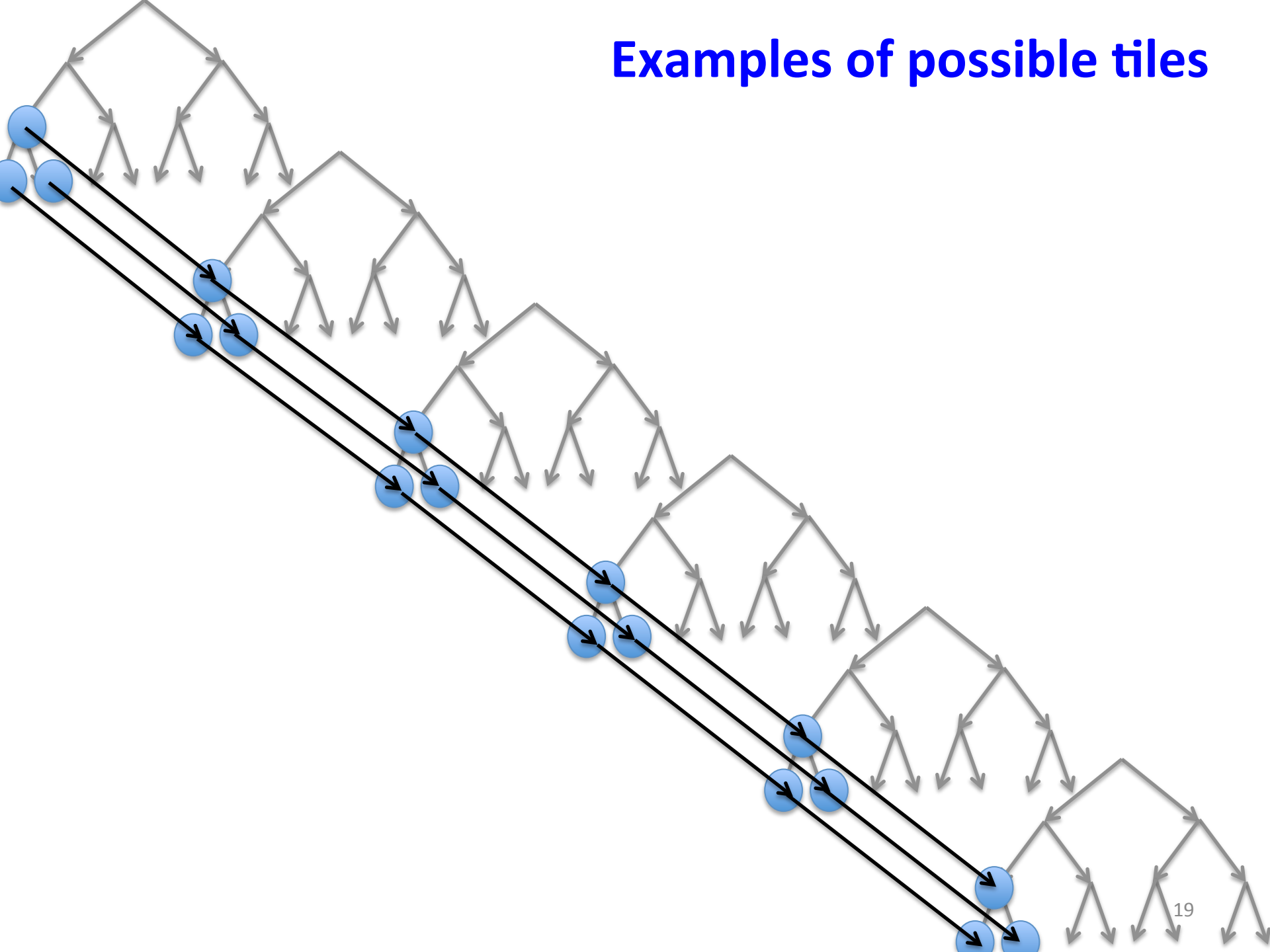
This is a 2D tile

[tile: whichTree, whichNodeOfTree]

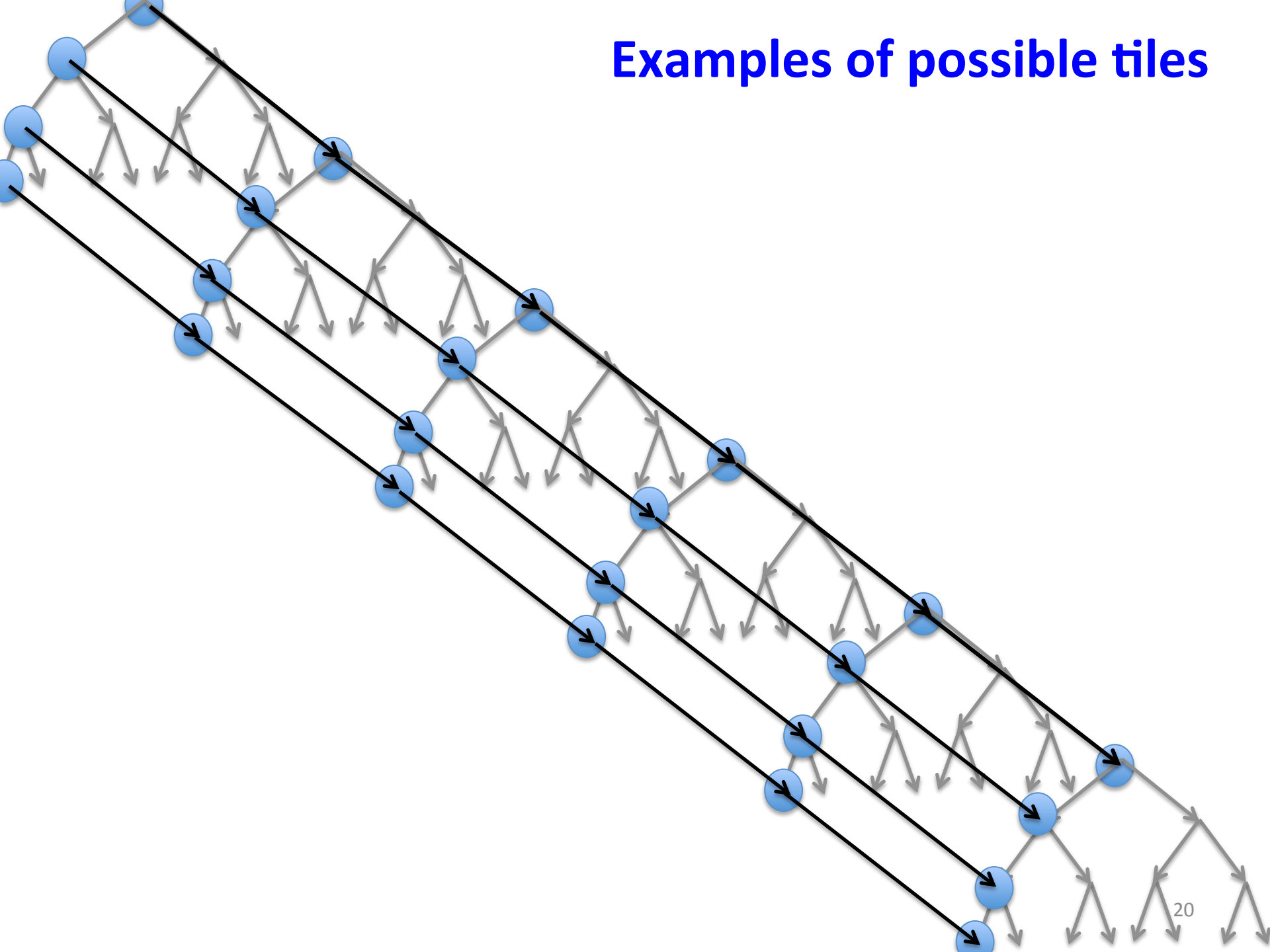
.



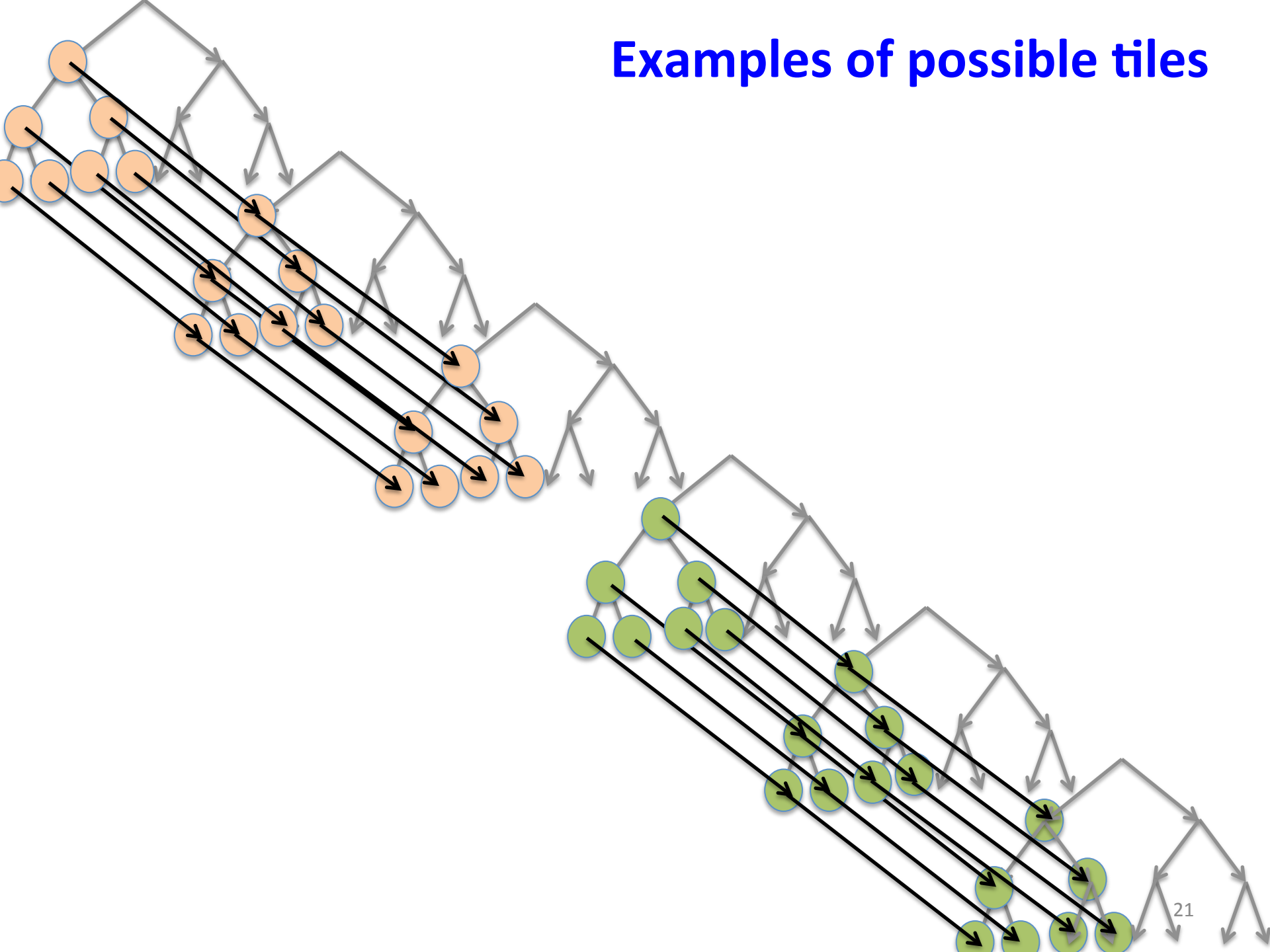
Examples of possible tiles



Examples of possible tiles



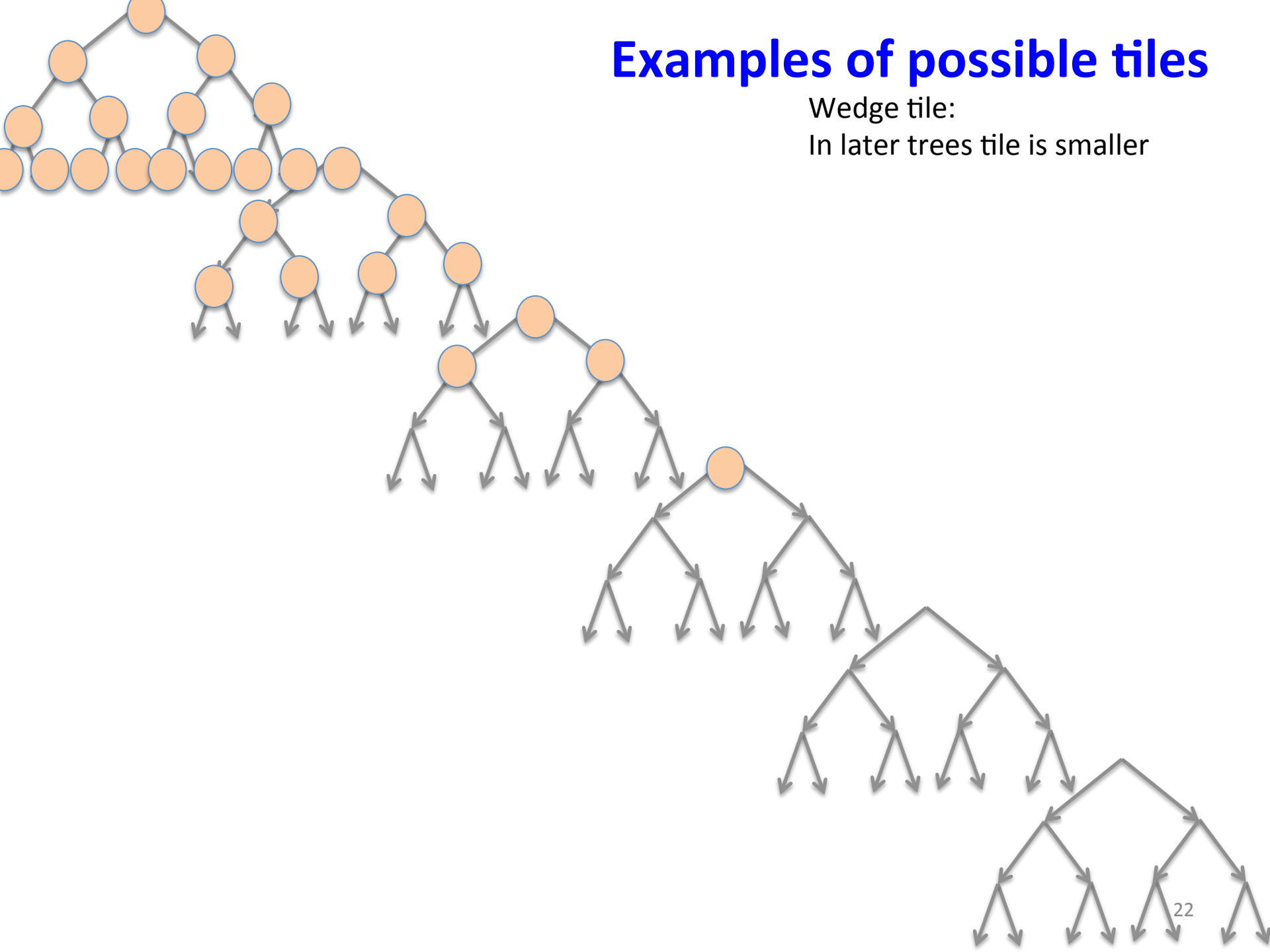
Examples of possible tiles



Examples of possible tiles

Wedge tile:

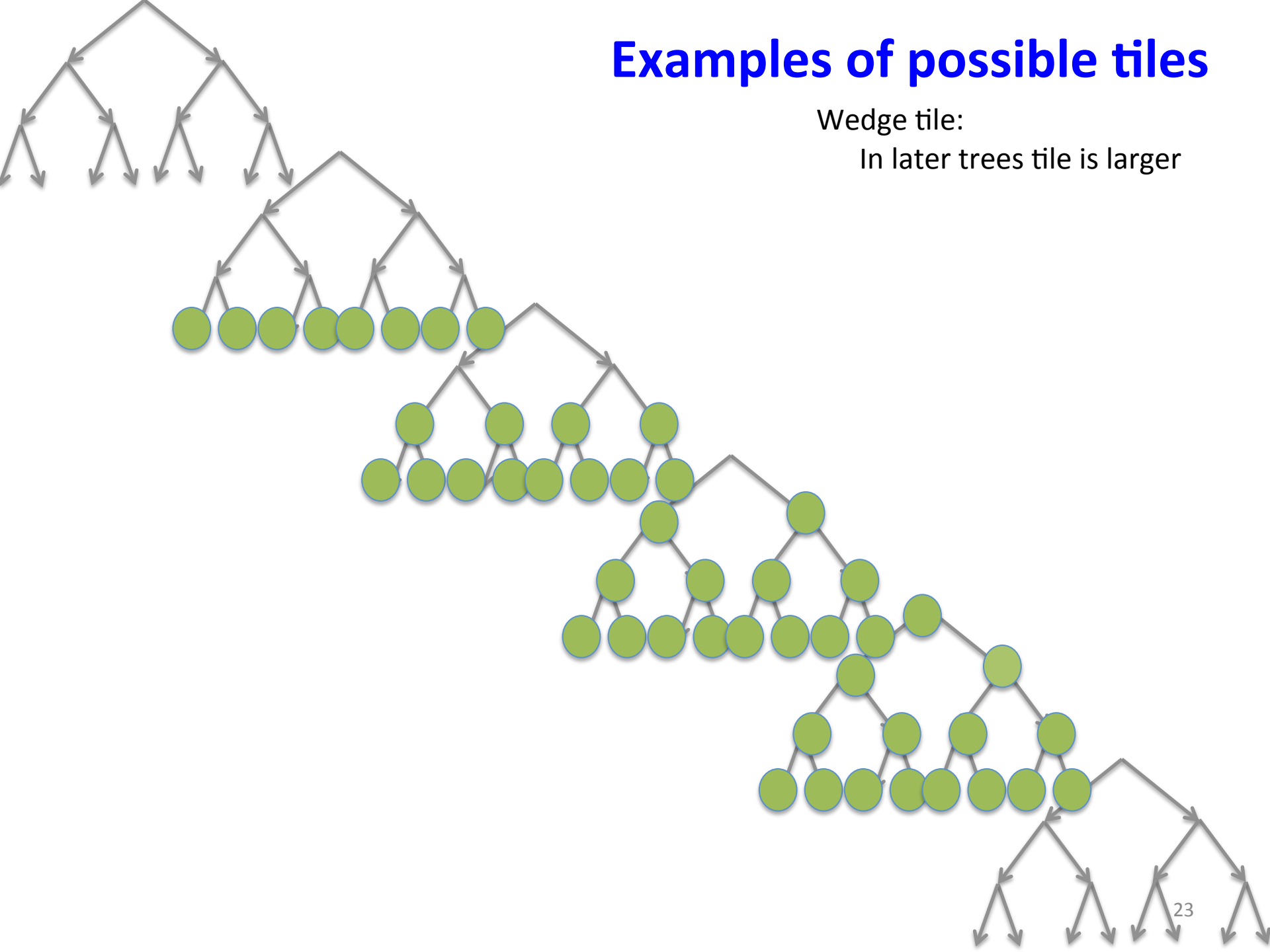
In later trees tile is smaller



Examples of possible tiles

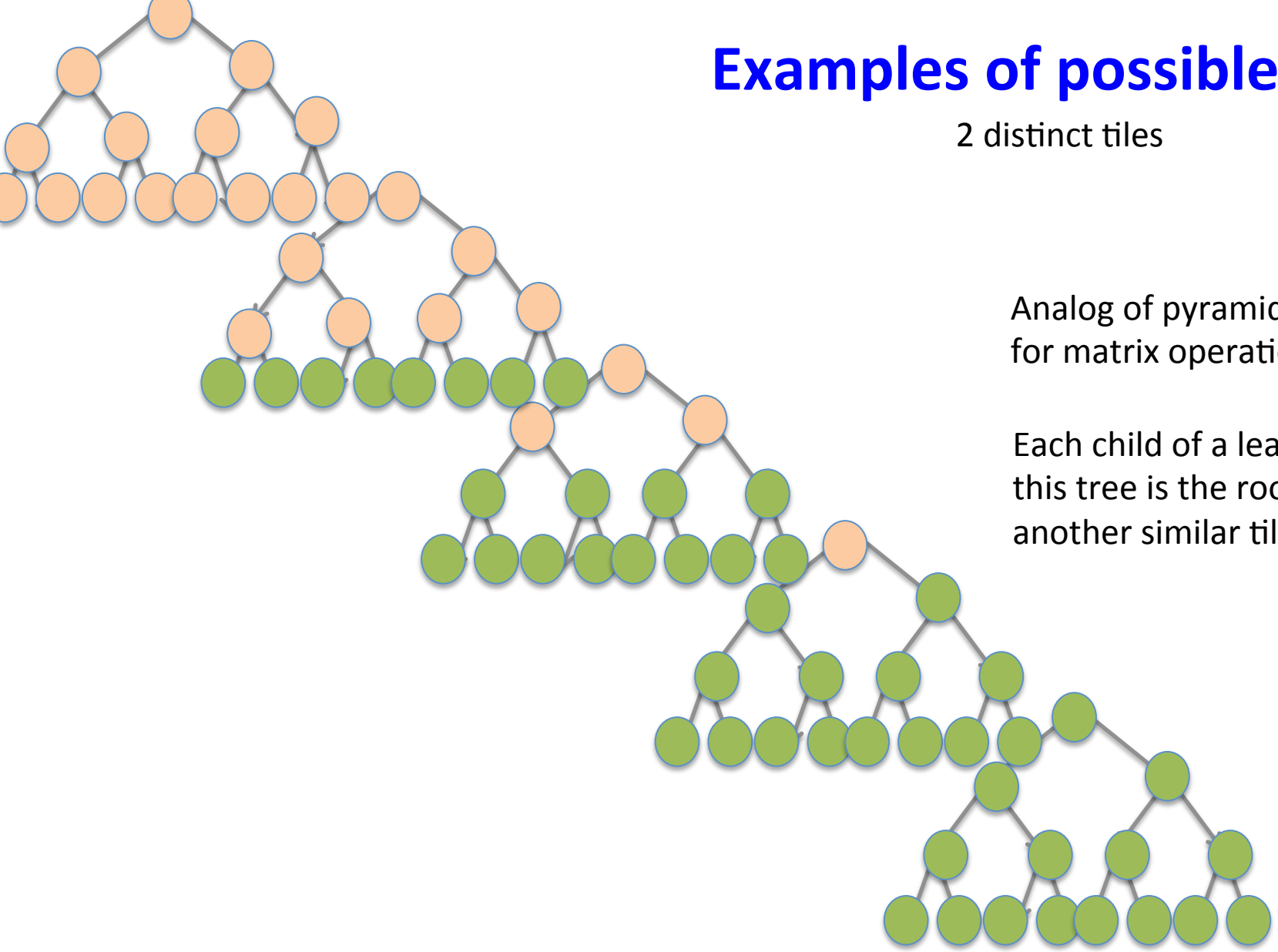
Wedge tile:

In later trees tile is larger



Examples of possible tiles

2 distinct tiles



Analog of pyramid tiles for matrix operations

Each child of a leaf node in this tree is the root of another similar tiling.

For SPX

- Develop the role of community
- Revisit array-based apps with the notion of communities in mind
  Maybe we should look (briefly) at one other community.
  Generalizing from 2 cases is a bit shaky.
- Develop a CnC tree-based community
  Assess existing state
  if necessary improve it
    Routines for the tree-base dependence functions: parent, first-child, next-tree etc.
    A set of tree-specific analyses, optimizations and runtimes for CnC
  result: improve portability within the tree-base community
- Develop experience with the idea of a tree-based CnC community
  Building applications in this class
  Developing optimizations for this community
  Developing analyses for this community
  Developing runtimes for this community
- Assess the idea of role of community

Conversations

- Now

  ⇒ Conversation: About tuning a specific app

  Applications expert & tuning expert

- New

  ⇒ Two conversations

  Community expert, applications expert & tuning expert

  - One about tuning a specific app
    - Application expert & tuning expert in context of a given community
  - One about tuning support for a specific community of apps
    - Community expert, application expert & tuning expert

Summary

- 3 distinct roles
- A range of ways it plays out
  - One person plays all three roles
  - Two roles + a negotiation between the domain expert and the tuning expert
  - 3 people / 3 roles
  - Community + two roles (the current array case)
- This perspective isolates the work of the community & simplifies the roles of the
  - The application developer
  - The tuning expert
- It provides a capability for the domain community to define a set of dependence functions and computations that
  - Is intuitive for the application expert
  - Can be used effectively by the tuning expert to optimize domain-specific applications.

Conclusion

- CnC doesn't know about arrays or trees but
  It does know about dependence functions
- Introduced new role: community
- A community shares a specific set of
  - Dependence functions
  - Structures
  - Analyses
  - Optimizations
  - Runtimes
- Examples:
  - Array-based community
  - Tree-based community
  - Other suggestions welcome