# A Pebble Game for Internet-Based Computing

Greg Malewicz    Arny Rosenberg    Matt Yurkewych

U. Alabama
Tuscaloosa, AL, USA

U. Massachusetts Amherst
Amherst, MA, USA

## New Modalities of *Collaborative Computing:* Internet-Based Computing (IC)

- The *owner* of a massive job enlists the aid of remote *clients* to compute the job's tasks.
- The owner (server) allocates tasks to clients one task at a time.
- A client receives its $(k+1)$th task after returning the results from its $k$th task.

# Historical Web-Computing Applications

- Astronomical calculations

  *SETI@home:*

  ⟨`http://setiathome.ssl.berkeley.edu`⟩

  *XPulsar@home:*

  ⟨`http://xpulsar.tat.physik.uni-tuebingen.de`⟩

- Factoring large integers

  *The RSA Factoring by Web Project:*

  ⟨`http://www.npac.syr.edu/factoring`⟩

- Drug testing

  *The Intel Philanthropic Peer-to-Peer program:*

  ⟨`www.intel.com/cure`⟩

  *The Olson Laboratory Fight AIDS@Home project:*

  ⟨`www.fightaidsathome.org`⟩

# Some Grid-Computing Projects

General-purpose "virtual supercomputers"

- The World-Wide Grid ⟨`http://www.cs.mu.oz.au/~raj/grids/wwg`⟩
- TeraGrid ⟨`http://www.teragrid.org`⟩
- The GUSTO Grid ⟨`http://www.globus.org`⟩

Targeted application areas

- The LHC Computing Grid ⟨`http://lcg.web.cern.ch/LCG`⟩
- the UK e-Science Grid ⟨`http://www.escience-grid.org.uk`⟩

# Challenges in Internet-Based Computing

Focus on jobs that have *intertask dependencies*.

- Remote clients become available at unpredictable times.
  * We want to *enhance their utilization*.

- Remote clients are not dedicated, hence can be unexpectedly slow.
  * We want to *avoid "gridlock"*
    (No new tasks can be allocated, pending completion of
    already allocated ones)

# Our Overall Goal

Determine how to schedule a *DAG of tasks* in a way that

- *lessens the danger of gridlock*
- *enhances the utilization of available client resources*

# An Avenue for Idealization

By carefully monitoring clients' past performance and present resources, the server can:

- mitigate the degree of temporal unpredictability;
- match task complexities to client resources.

(Buyya-Abramson-Giddy, Kondo-Casanova-Wing-Berman, Sun-Wu)

## Our Contributions

1. A formal framework for studying scheduling for IC

2. Under idealized assumptions:

   (a) optimal scheduling strategies for familiar classes of dags

   - 2-D *evolving meshes*
   - (2-D) *reduction-meshes*
   - (binary) *reduction-trees*
   - *FFT dags*

   (b) a foundation for a decomposition-based scheduling theory for a large class of "well-structured" dags

## The Mathematical/Algorithmic Setting

## The Internet-Computing (IC) Pebble Game

- The Game is played on a (finite or infinite) dag $\mathcal{G}$, which represents the *job*.
  - Each node of $\mathcal{G}$ represents a *task*.
  - Arc $(u \rightarrow v)$ of $\mathcal{G}$ represents an *intertask dependency*:
    $\rightarrow$task $v$ cannot be *allocated* until its *parent* task $u$ is *executed*.

## “Equipment” for the Game

- The Players:
  - a single ***Server***, $S$ (the “owner”)
  - a (finite or infinite) set of ***Clients***, $C_1, C_2, \ldots$

- $S$ has limitless supplies of three types of ***Pebbles***:
  - ELIGIBLE BUT UNALLOCATED (EBU)
  - ELIGIBLE AND ALLOCATED (EAA)
  - EXECUTED (XEQ)

## The Rules of the IC Pebble Game

**1.** $S$ places an EBU pebble on every unpebbled *source* (parentless) node of $\mathcal{G}$.

/*Unexecuted parentless tasks do not depend on the prior execution of any other task.*/

$$\boxed{\begin{array}{ccccc} \langle 0,0\rangle & \rightarrow & \langle 0,1\rangle & \rightarrow & \cdots \\ \downarrow & & \downarrow & & \\ \langle 1,0\rangle & \rightarrow & \ddots & & \\ \downarrow & & & & \\ \vdots & & & & \end{array}} \Longrightarrow \boxed{\begin{array}{ccccc} \boxed{\text{EBU}} & \rightarrow & \langle 0,1\rangle & \rightarrow & \cdots \\ \downarrow & & \downarrow & & \\ \langle 1,0\rangle & \rightarrow & \ddots & & \\ \downarrow & & & & \\ \vdots & & & & \end{array}}$$

**2.** When Client $C_i$ approaches $S$ requesting a task—

**a.** If $\mathcal{G}$ contains an EBU task, then $S$ gives $C_i$ one such task and replaces that task's pebble by an EAA pebble.

**b.** If no task of $\mathcal{G}$ is EBU, then this move is a no-op.

$$\boxed{\begin{array}{ccccc} \boxed{\mathrm{EBU}} & \to & \langle 0,1\rangle & \to & \cdots \\ \downarrow & & \downarrow & & \\ \langle 1,0\rangle & \to & \ddots & & \\ \downarrow & & & & \\ \vdots & & & & \end{array}} \quad\Longrightarrow\quad \boxed{\begin{array}{ccccc} \boxed{\mathrm{EAA}} & \to & \langle 0,1\rangle & \to & \cdots \\ \downarrow & & \downarrow & & \\ \langle 1,0\rangle & \to & \ddots & & \\ \downarrow & & & & \\ \vdots & & & & \end{array}}$$

**3.** When a Client returns (the results from) a task $v$—
(*Eventually! Here's where gridlock can occur.*)

**a.** $S$ replaces $v$'s EAA pebble by an XEQ pebble.

**b.** $S$ places an EBU pebble on each unpebbled task of $\mathcal{G}$ all of whose parents contain XEQ pebbles.

$$
\boxed{\begin{array}{ccccc}
\boxed{\mathrm{EAA}} & \rightarrow & \langle 0,1\rangle & \rightarrow & \cdots \\
\downarrow & & \downarrow & & \\
\langle 1,0\rangle & \rightarrow & \ddots & & \\
\downarrow & & & & \\
\vdots & & & &
\end{array}}
\Longrightarrow
\boxed{\begin{array}{ccccc}
\boxed{\mathrm{XEQ}} & \rightarrow & \boxed{\mathrm{EBU}} & \rightarrow & \cdots \\
\downarrow & & \downarrow & & \\
\boxed{\mathrm{EBU}} & \rightarrow & \ddots & & \\
\downarrow & & & & \\
\vdots & & & &
\end{array}}
$$

And the cycle continues ...

**4.** $S$'s goal is to ensure that every task $v$ of $\mathcal{G}$ *eventually* contains an $\mathrm{XEQ}$ pebble.

/*This modest goal is necessitated by the possibility that $\mathcal{G}$ is infinite. (There may be infinite plays that never pebble some nodes.)*/

# The Quality of a Play of the IC Pebble Game

- A Quality Measure should formalize the principle:

$$\textit{Better plays} \begin{cases} \textit{enhance use of remote resources} \\ \textit{have less chance of gridlock.} \end{cases}$$

*BUT...*

- *Adversarial Clients can confute any strategy the Server adopts!*

# Toward a Quality Measure for IC Pebbling, 1

We need some idealization, so we assume that ...

*By careful monitoring, the Server can (approximately) ensure that:*

> *The temporal unpredictability of Clients affects the timing, but not the order of task executions.*

By assuming that monitoring is perfect(!), we arrive at the ...

**Assumption.** *Tasks are executed in the order of allocation.*

## Toward a Quality Measure for IC Pebbling, 2

This assumption allows us to:

- Simplify the repertoire of pebbles—

  — lump EAA, XEQ pebbles together as EXECUTED pebbles

- Recalibrate "time" to be *event-driven*—

  — a node is executed at every step

- Derive scheduling guidelines whose actions are *totally under the control of the Server*

# IC Quality/Optimality of a Play of the Game

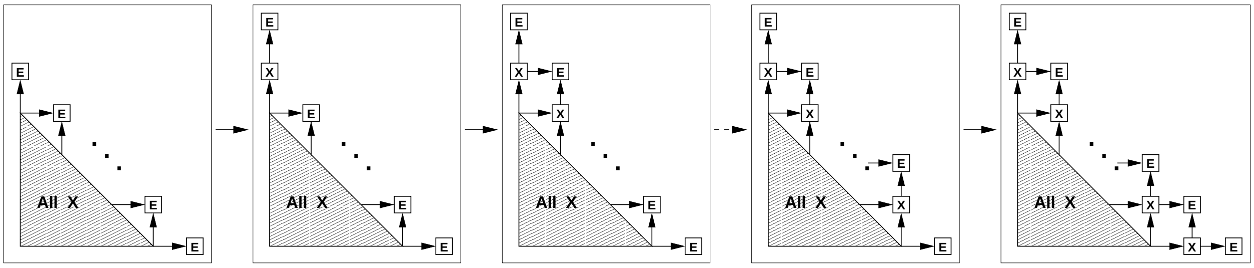
The **IC quality** of a play of the Game:

—the rate of producing ELIGIBLE pebbles — *the larger, the better.*

A play of the Game is **IC optimal**:

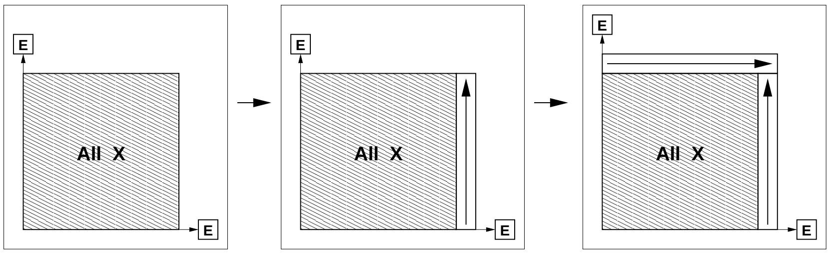
—It *maximizes* the number of ELIGIBLE nodes *for all steps $t$*.

# How Important is IC Quality/Optimality?



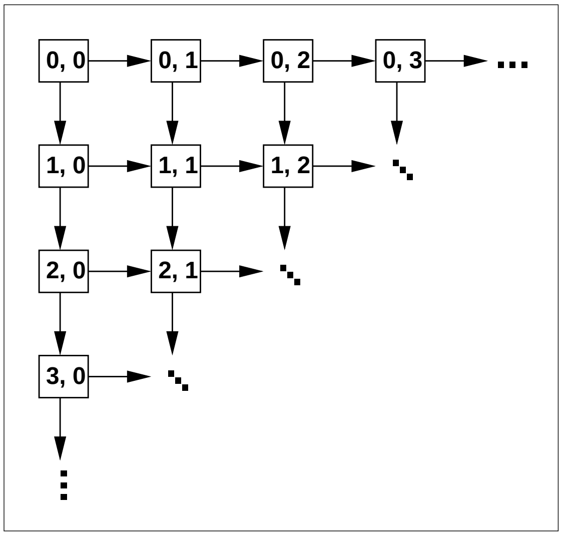

⇑ Roughly $\sqrt{T}$ ELIGIBLE nodes at time $T$ ⇑

⇓ Never more than $3$ ELIGIBLE nodes ⇓



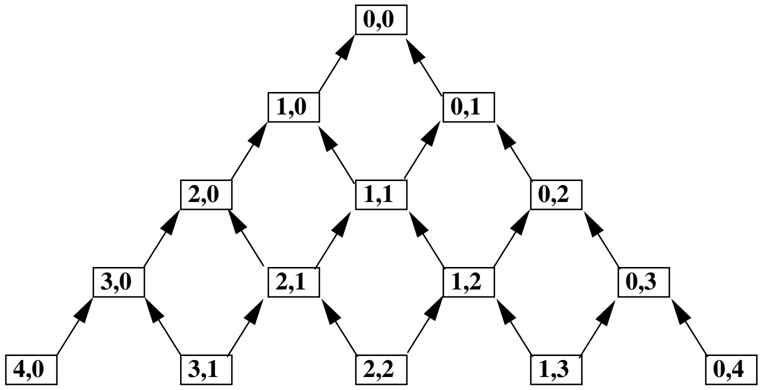
# Optimal Schedules for Some Familiar Dag-Families

# Evolving (2-Dimensional) Mesh-Dags




Evolving mesh-dags model wavefront computations.

# (2-Dimensional) Reduction-Meshes

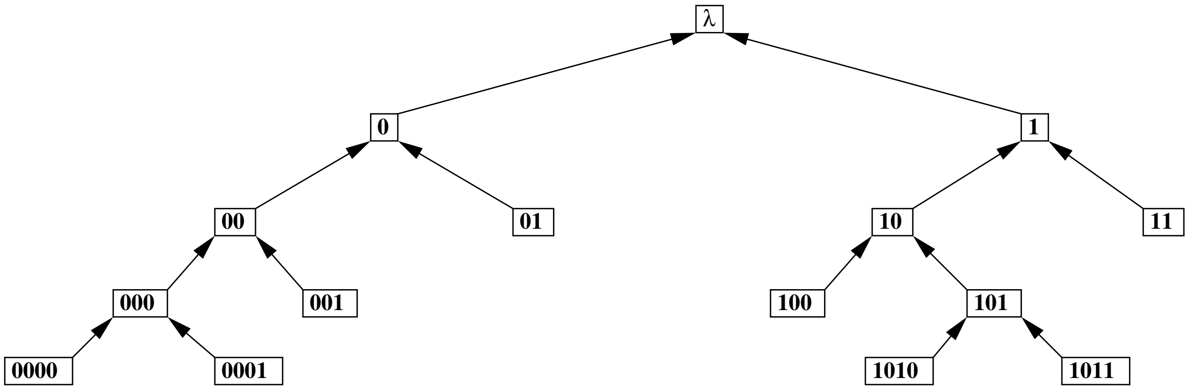



Reduction-mesh (a/k/a *pyramid dags*) computations:

—have mesh-structured dependencies

—can be executed via sweeps up the mesh.

# (Binary) Reduction-Trees

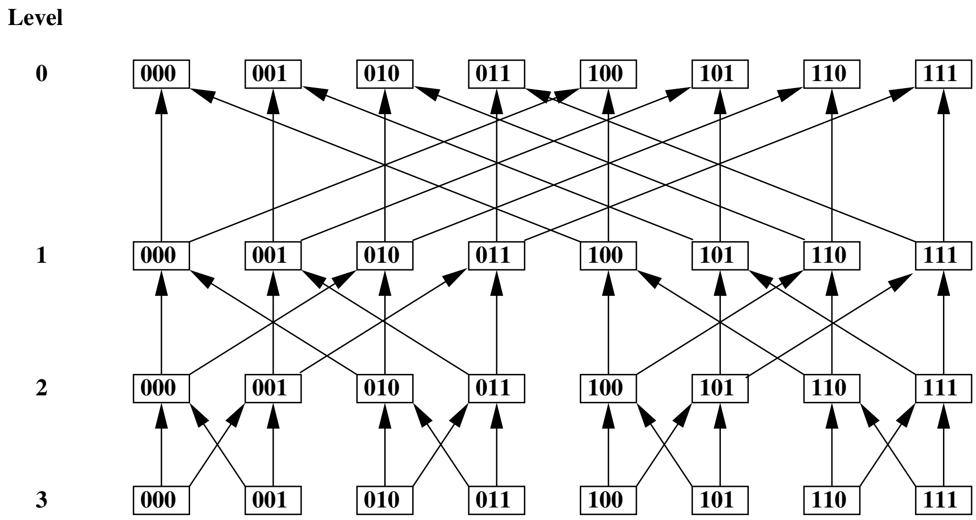



Reduction-tree computations (full binary tree-dags):

—have tree-structured dependencies

—can be executed via sweeps up the tree.

# FFT Dags




FFT dags model convolutional computations.

## IC-Optimal Schedules for these Common Dags

**Theorem.** *A schedule for*

$$\left\{ \begin{array}{l} \text{an evolving mesh-dag (2- or 3-D)} \\ \text{a reduction-mesh} \\ \text{a reduction-tree} \\ \text{an FFT dag} \end{array} \right\}$$

*is IC optimal* iff *it is parent oriented.*

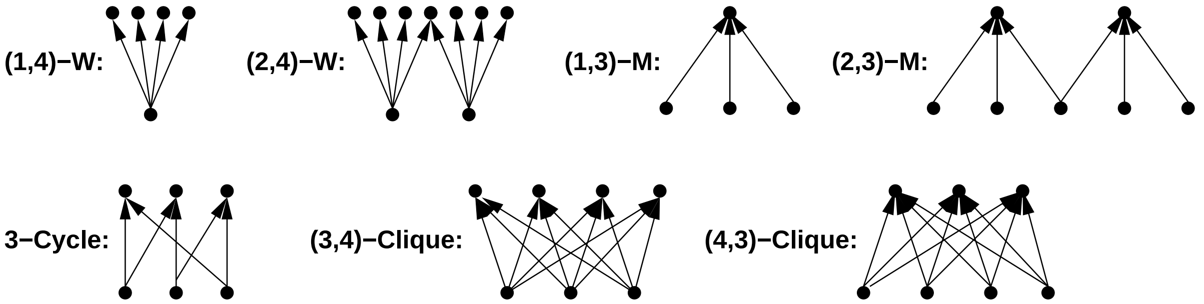
*Parent oriented*: A node's parents are executed sequentially.

- Meshes: level by level, sequentially across each level
- Tree-dags: in "sibling pairs" (nodes that share a child)
- FFT-dags: in "butterfly pairs" (nodes that share two children)

**Toward a Decomposition-Based Scheduling Theory:**
**1. Conceptual Underpinnings**

# 1. Select a Set of “Building Block” Dags

Start with *bipartite “building block” dags* that we know how to schedule optimally. A small sampler:




**Theorem.** *Any schedule for these building blocks that executes all sources sequentially is IC optimal.*

## 2. Establish "Priorities" among the Building Blocks

Say that $\begin{cases} \mathcal{G}_1 \text{ admits an IC-optimal schedule } \Sigma_1 \\ \mathcal{G}_2 \text{ admits an IC-optimal schedule } \Sigma_2 \end{cases}$

$\underline{\underline{\mathcal{G}_1 \rhd \mathcal{G}_2}}$ means:

The following schedule is IC optimal: $\begin{cases} 1. \text{ follow } \Sigma_1 \text{ on } \mathcal{G}_1 \\ 2. \text{ follow } \Sigma_2 \text{ on } \mathcal{G}_2 \end{cases}$

**Lemma.** *The relation $\rhd$ is transitive.*

# Priorities among Some Building Blocks

| | | | |
|---|---|---|---|
| $\mathcal{W}_{s,d}$: | $s$-source, degree-$d$ W-dag | $\mathcal{C}_s$: | $s$-source Bipartite Cycle |
| $\mathcal{M}_{s,d}$: | $s$-sink, degree-$d$ M-dag | $\mathcal{Q}_s$: | $s$-source Bipartite Clique |

| $\triangleright$ | $\mathcal{W}_{s',d'}$ | $\mathcal{M}_{s',d'}$ | $\mathcal{C}_{s'}$ | $\mathcal{Q}_{s'}$ |
|---|---|---|---|---|
| $\mathcal{W}_{s,d}$ | $d' \le d$ **or** $d' = d$ **and** $s' \ge s$ | all $s', d'$ | all $s'$ | all $s'$ |
| $\mathcal{M}_{s,d}$ | No | $d' \ge d$ **or** $d' = d$ **and** $s' \le s$ | No | X |
| $\mathcal{C}_s$ | No | all $s', d'$ | $s' = s$ else X | $s' = s$ else X |
| $\mathcal{Q}_s$ | No | X | No for $s' = s$ **else** X | $s' = s$ **else** X |

"X" $\Rightarrow$ no IC-optimal schedule exists.

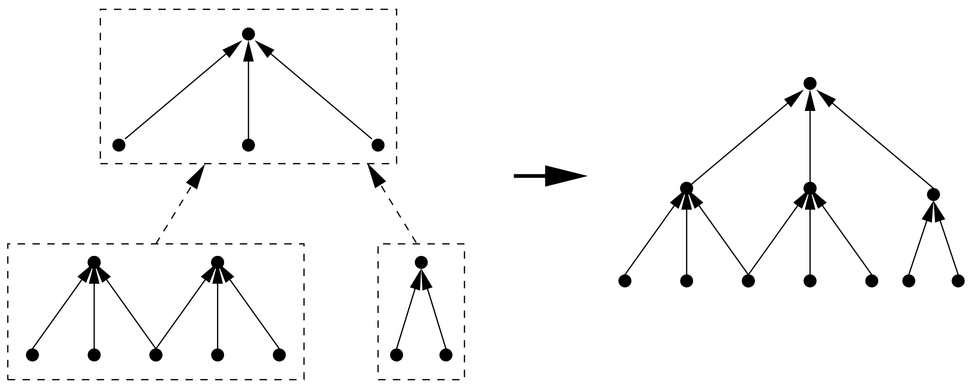
## Complex Dags via "Composition," 1

<u>Compose $\mathcal{G}_1$ with $\mathcal{G}_2$</u>:

*Identify some $k$ sources of $\mathcal{G}_2$ with some $k$ sinks of $\mathcal{G}_1$.*

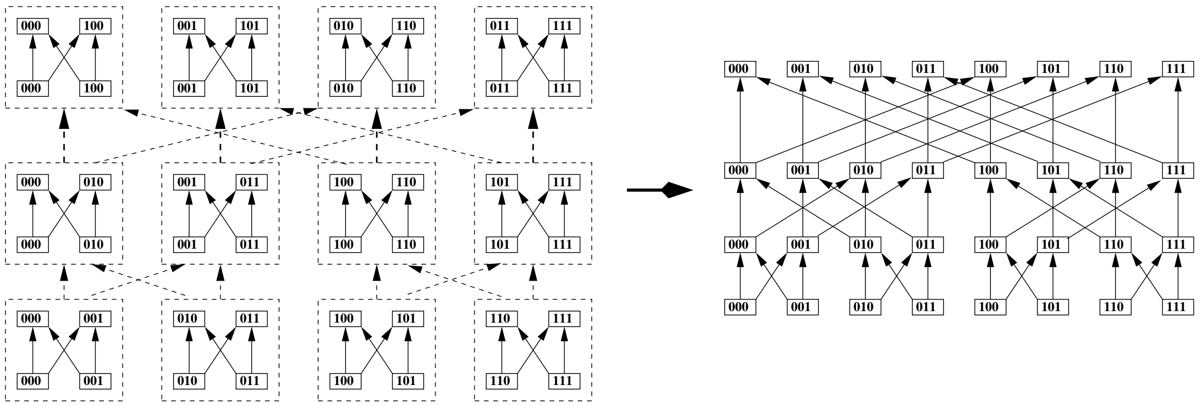
The dag obtained is *composite of type* $\underline{\mathcal{G}_1 \Uparrow \mathcal{G}_2}$.

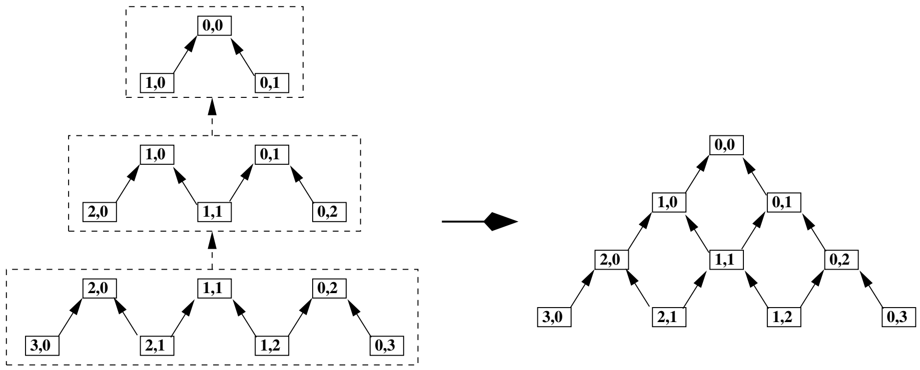
<u>Example</u>: $\mathcal{M}_{1,2} \Uparrow \mathcal{M}_{2,3} \Uparrow \mathcal{M}_{1,3}$. (*Composition is associative.*)



# Complex Dags via "Composition," 2

Our "familiar" dags are compositions of building blocks.



# Why "Composition" Is Important

**Theorem.** *If the dag $\mathcal{G}$ is a composition of connected bipartite dags $\{\mathcal{G}_i\}_{1 \le i \le n}$, of type*

$$\mathcal{G}_1 \Uparrow \mathcal{G}_2 \Uparrow \cdots \Uparrow \mathcal{G}_n$$

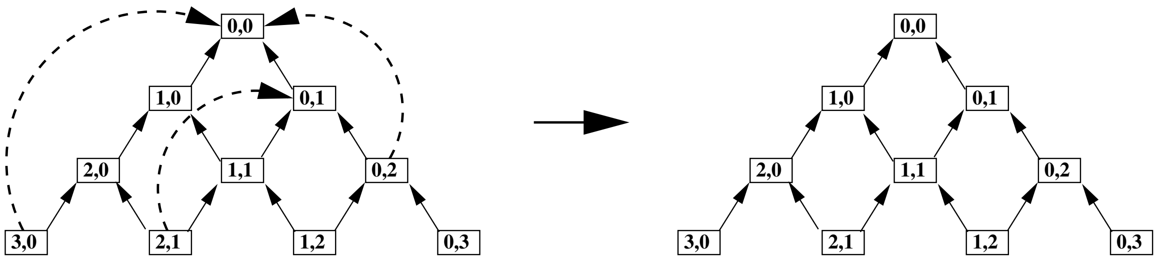
*and if*

$$\mathcal{G}_1 \rhd \mathcal{G}_2 \rhd \cdots \rhd \mathcal{G}_n \qquad (\mathcal{G} \text{ is a } \underline{\rhd\text{-linear composition}})$$

*then executing $\mathcal{G}$ by executing the $\mathcal{G}_i$ in $\rhd$-order is IC optimal.*

**Toward a Decomposition-Based Scheduling Theory:**
**2. Algorithmic Development**

# 1. Eliminate All Shortcuts from $\mathcal{G}$




- Composite dags have no shortcuts.
- One can remove shortcuts in (low-degree) polynomial time.
- *Any IC-optimal schedule for the shortcut-free version of $\mathcal{G}$ is IC optimal for $\mathcal{G}$.*

## 2. Parse $\mathcal{G}$ into Building Blocks

1. Start from the dag $\mathcal{G}$'s sources.
2. Detach a maximal connected bipartite subdag whose sources are sources of $\mathcal{G}$.
3. Recurse.

**Theorem.** *If $\mathcal{G}$ is composite, then this process identifies a unique sequence of bipartite building blocks $(\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ such that $\mathcal{G}$ is composite of type $\mathcal{B}_1 \Uparrow \mathcal{B}_2 \Uparrow \cdots \Uparrow \mathcal{B}_n$.*

## 3. Look for a $\rhd$-Linearization of $\mathcal{G}$

**Theorem.** *Say that:*

- *$\mathcal{G}$ is composite of type $\mathcal{B}_1 \Uparrow \mathcal{B}_2 \Uparrow \cdots \Uparrow \mathcal{B}_n$;*
- *for all $\mathcal{B}_i, \mathcal{B}_j$ with $i \neq j$, either $\mathcal{B}_i \rhd \mathcal{B}_j$ or $\mathcal{B}_j \rhd \mathcal{B}_i$.*

*Then $\mathcal{G}$ is a $\rhd$-linear composition whenever the following holds.*

> *Whenever $\mathcal{B}_j$ is directly composed with $\mathcal{B}_i$ in a construction of $\mathcal{G}$, we have $\mathcal{B}_i \rhd \mathcal{B}_j$.*

(The proof combines a sort on $\rhd$ with a topological sort on the composition structure of $\mathcal{G}$.)

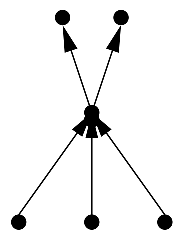
Work in Progress

# A Sampler of Projects in Progress

1. Augment the repertoire of building-block dags.

2. Extend the priority relation $\triangleright$ to include topological order.

   Order of composition
   (rather than $\triangleright$-priority)
   may force a schedule
   to execute $\mathcal{G}_1$ before $\mathcal{G}_2$ .

3. Determine how to incorporate schedules that execute building-block dags in an interleaved—rather than sequential—fashion.