Kinetic Dependence Graphs

M. Amber Hassaan,

Donald Nguyen,

Keshav Pingali

The University of Texas at Austin

Dependence Graph Scheduling

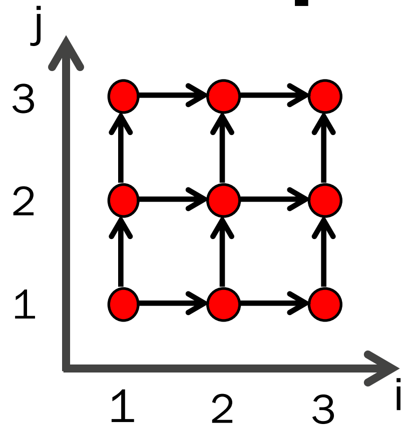
```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```

Application Programmer

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```

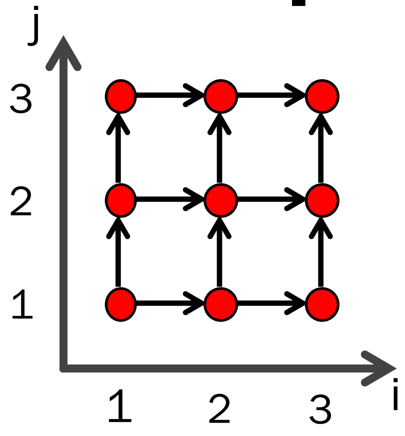
Application Programmer

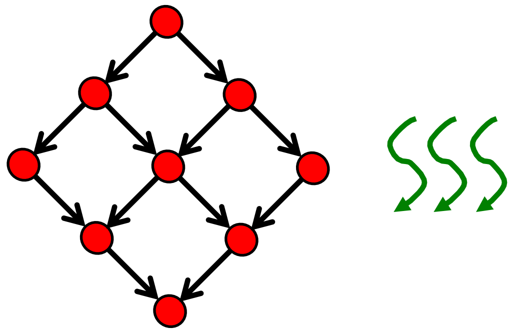


Compiler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





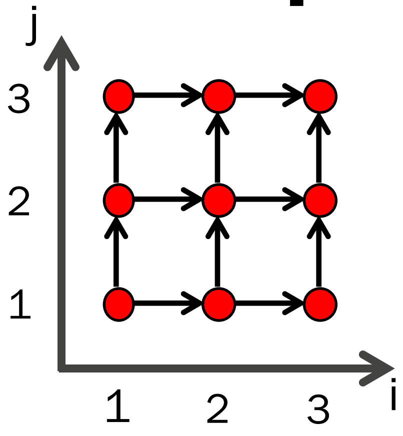
Application Programmer

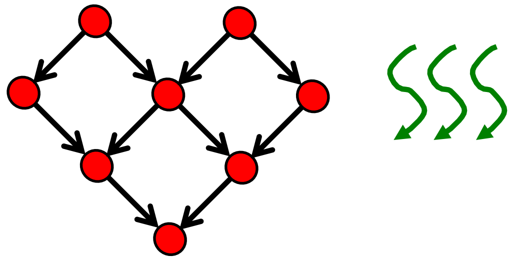
Compiler

Runtime Scheduler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





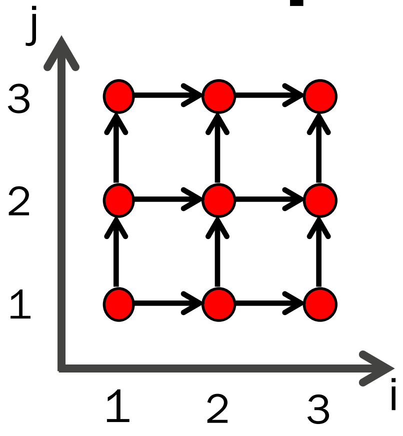
Application Programmer

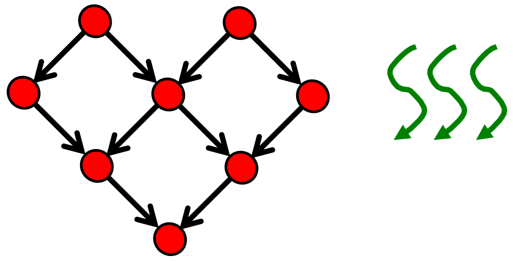
Compiler

Runtime Scheduler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





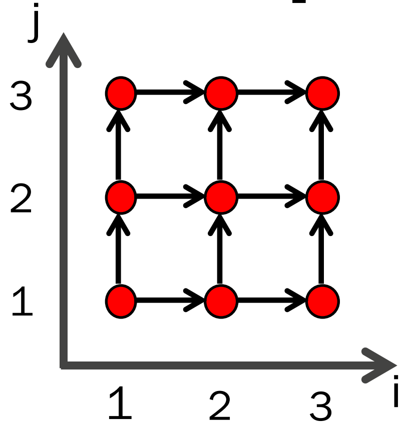
Application Programmer

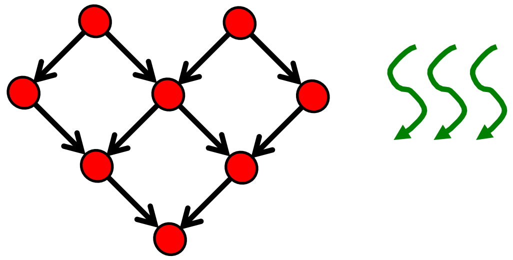
Compiler
Parallel Task Library

Runtime Scheduler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





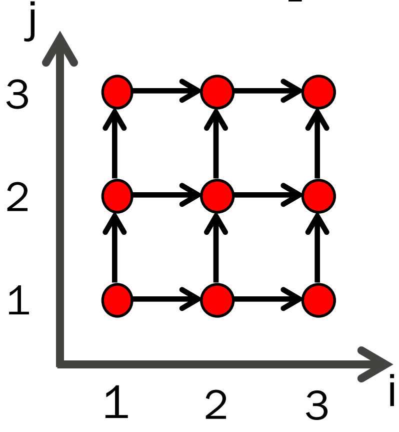
Application Programmer

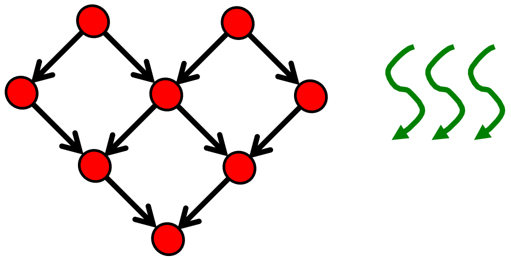
Compiler
Parallel Task Library
Parallel Programming Model

Runtime Scheduler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





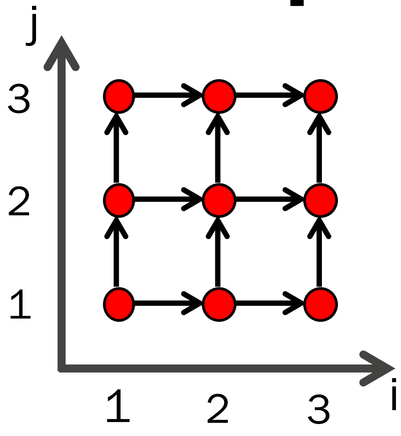
Application Programmer

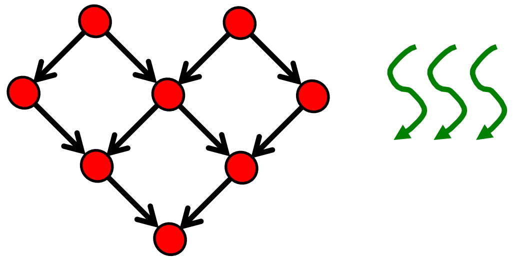
Compiler
Parallel Task Library
Parallel Programming Model
⋮

Runtime Scheduler

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

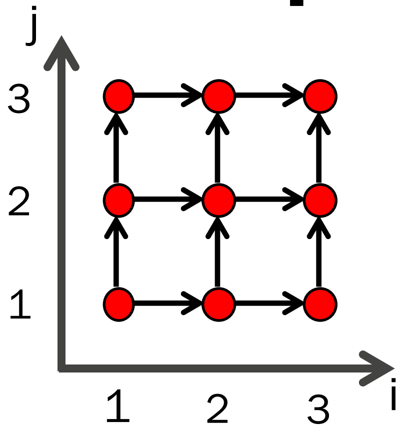
Compiler
Parallel Task Library
Parallel Programming Model
⋮

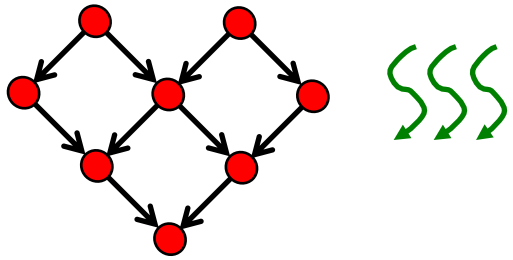
Runtime Scheduler

σ_0

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

Compiler
Parallel Task Library
Parallel Programming Model
⋮

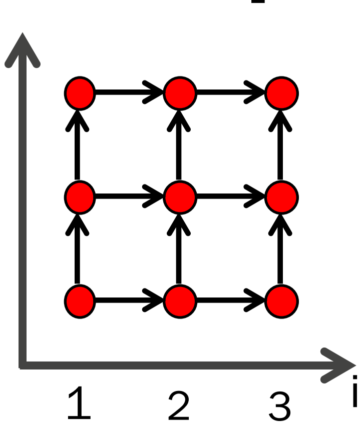
Runtime Scheduler

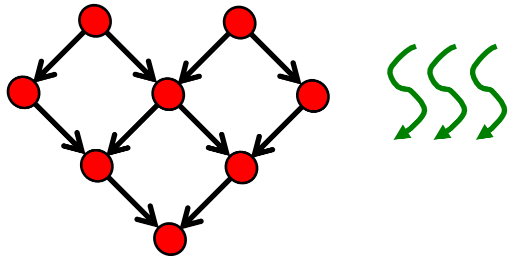
G_0

σ_0

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

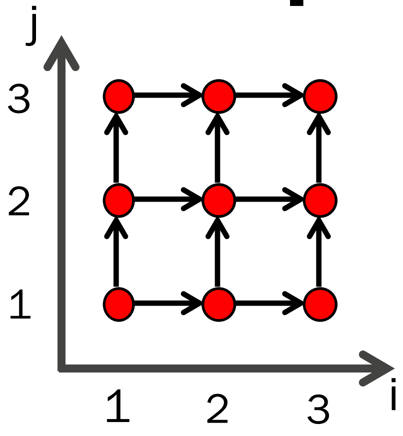
Compiler
Parallel Task Library
Parallel Programming Model
⋮

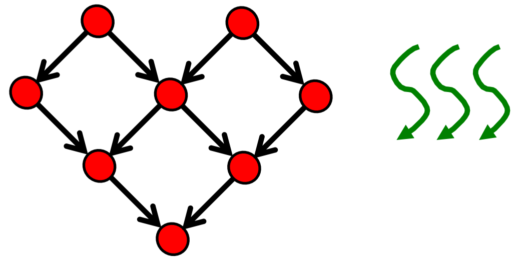
Runtime Scheduler

$$\begin{tikzcd} G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \\ \sigma_0 \arrow[u, dashed] & \end{tikzcd}$$

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

Compiler
Parallel Task Library
Parallel Programming Model
⋮

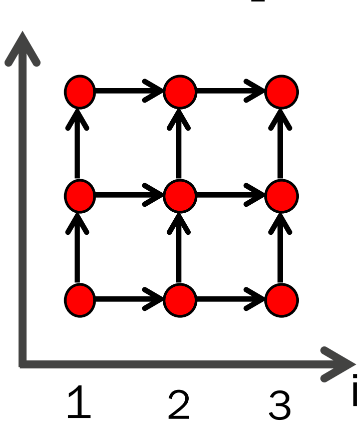
Runtime Scheduler

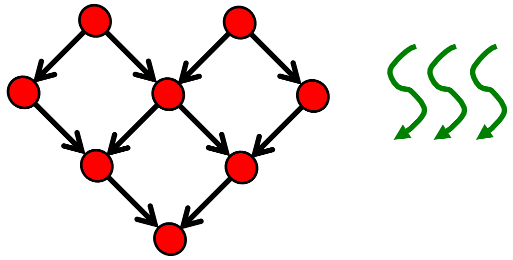
Execute in Parallel

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "w_0"] & \boxed{G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & {}} \\
\sigma_0 \arrow[u, dashed]
\end{tikzcd}$$

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

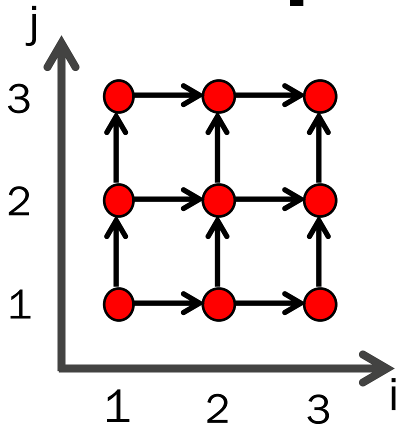
Compiler
Parallel Task Library
Parallel Programming Model
⋮

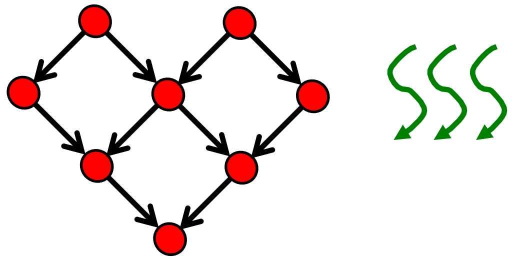
Runtime Scheduler

$$\begin{tikzcd} G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \ldots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\ \sigma_0 \arrow[u, dashed] \end{tikzcd}$$

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

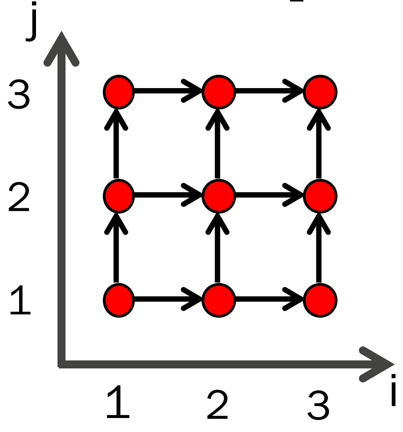
Compiler
Parallel Task Library
Parallel Programming Model
⋮

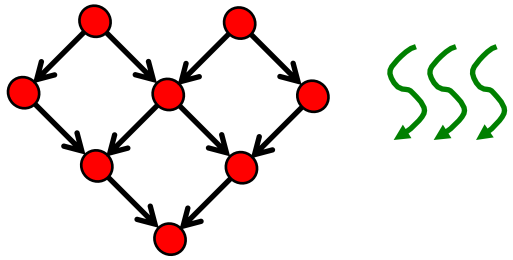
Runtime Scheduler

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \dots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] & & &
\end{tikzcd}$$

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

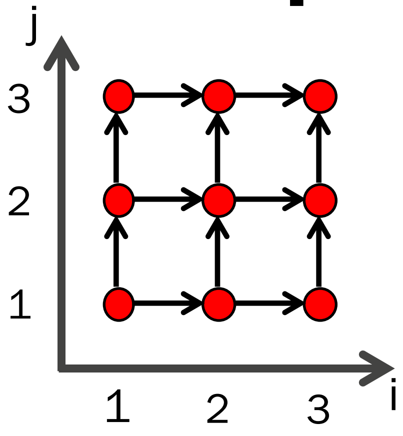
Compiler
Parallel Task Library
Parallel Programming Model
⋮

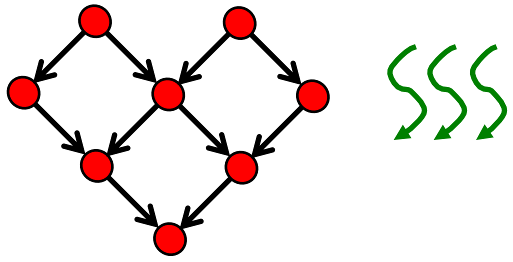
Runtime Scheduler

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \dots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] \arrow[r, "w_1"] & \sigma_2 \arrow[u, dashed] \arrow[r, "w_2"] & \dots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed]
\end{tikzcd}$$

Dependence Graph Scheduling

```
A[N,N] = ...

for i in 1:N
  for j in 1:N
    A[i,j] =
      A[i-1,j] + A[i,j-1]
```





Application Programmer

Compiler
Parallel Task Library
Parallel Programming Model
⋮

Runtime Scheduler

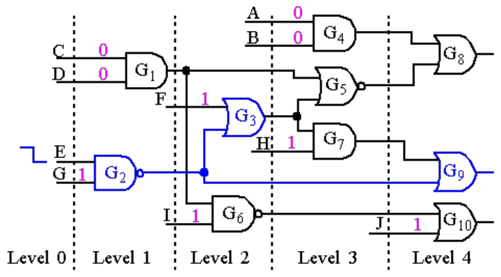
Scheduler World

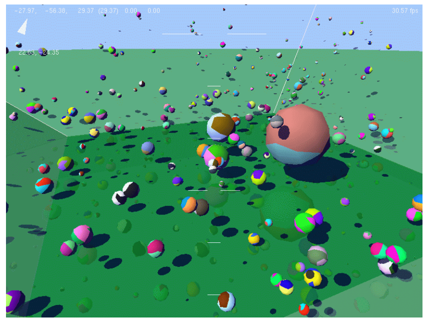
Program World

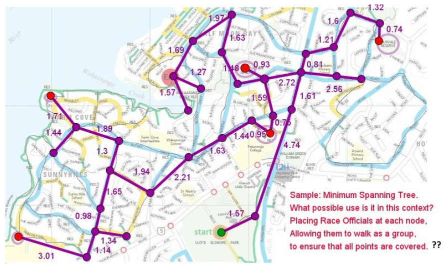
$$\begin{tikzcd} G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \dots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\ \sigma_0 \arrow[r, "w_0"] \arrow[u, dashed] & \sigma_1 \arrow[r, "w_1"] \arrow[u, dashed] & \sigma_2 \arrow[r, "w_2"] \arrow[u, dashed] & \dots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed] \end{tikzcd}$$

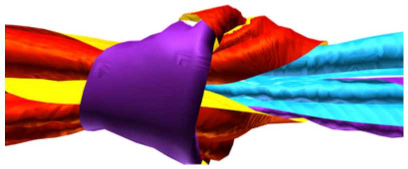
Limitations of Dependence Graphs

- Not always sufficient
  - Discrete event simulation
  - Billiard ball simulation
  - Kruskal’s algorithm for MSTs
  - Asynchronous Variational Integrators
  - …
- Ordered algorithms
  - Have tasks with algorithm-specific order in which tasks must appear to execute
- Can use speculation
  - But overheads can be high for ordered algorithms

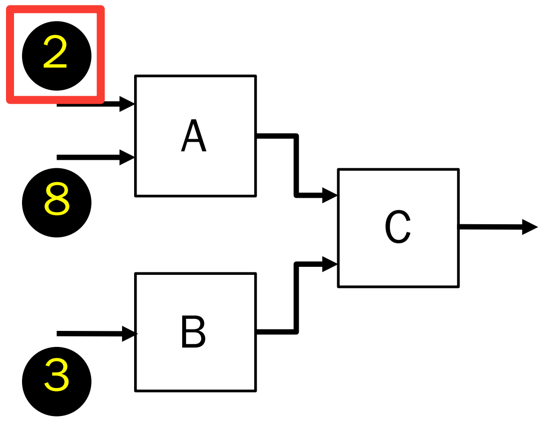




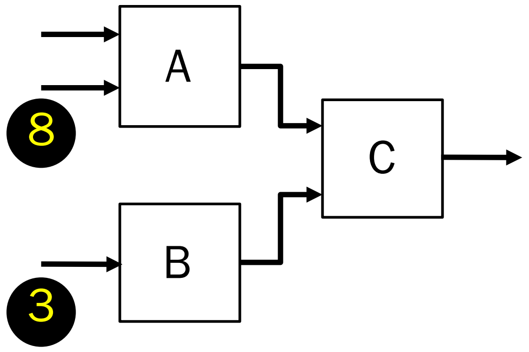

Sample: Minimum Spanning Tree.
What possible use is it in this context?
Placing Race Officials at each node,
Allowing them to walk as a group,
to ensure that all points are covered. ??



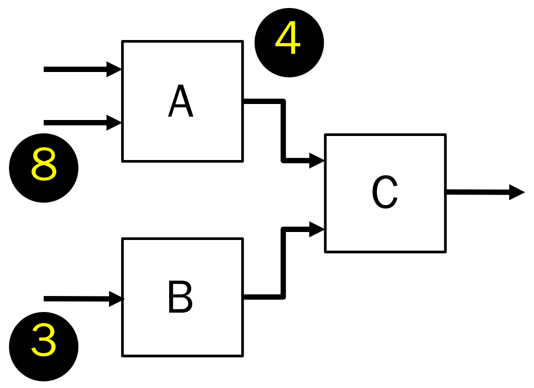
Discrete Event Simulation



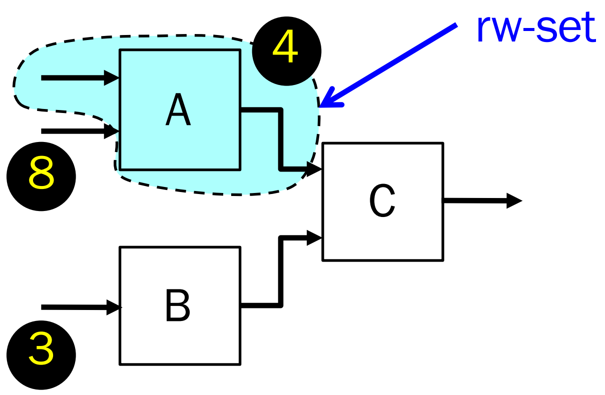
Discrete Event Simulation



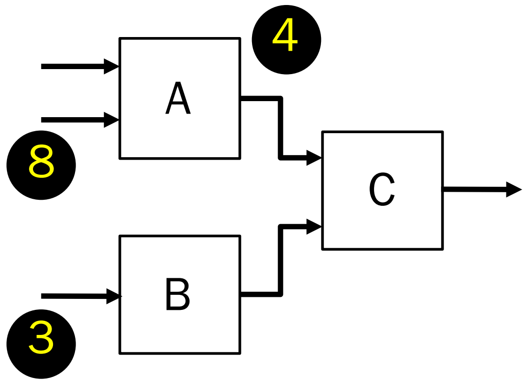
Discrete Event Simulation



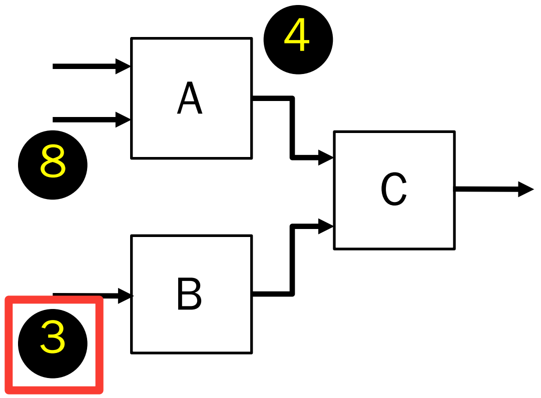
Discrete Event Simulation



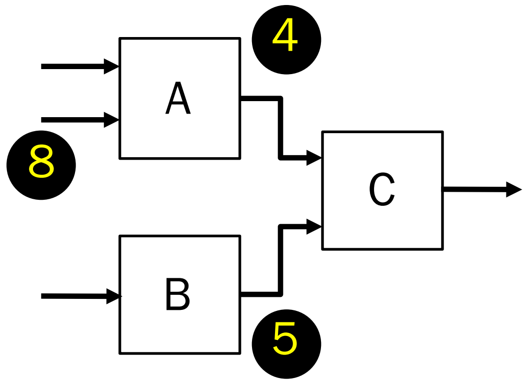
Discrete Event Simulation



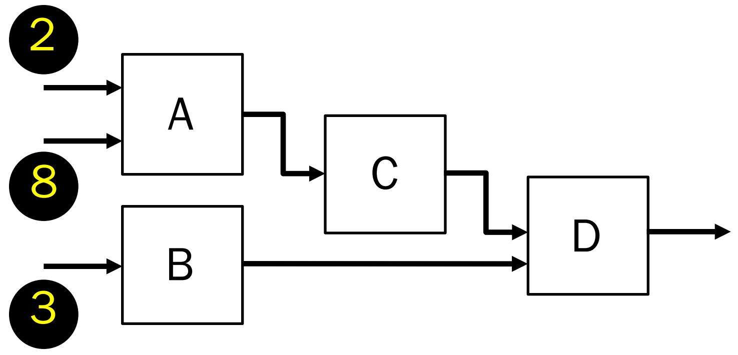
Discrete Event Simulation



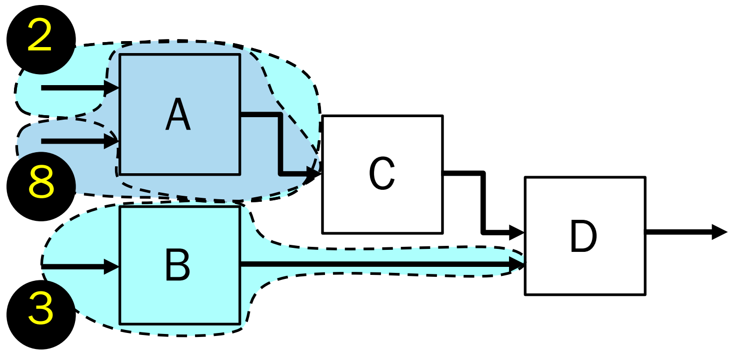
Discrete Event Simulation



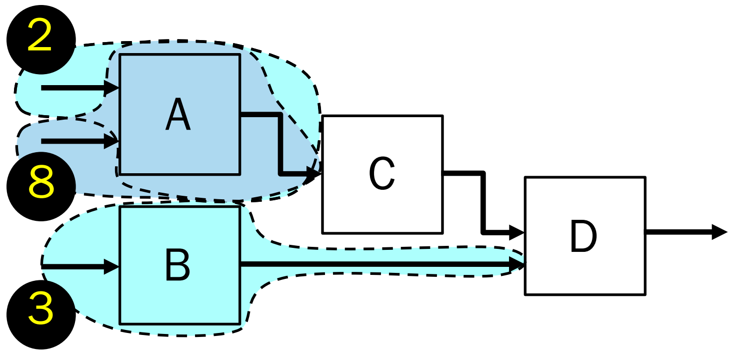
Discrete Event Simulation



Discrete Event Simulation

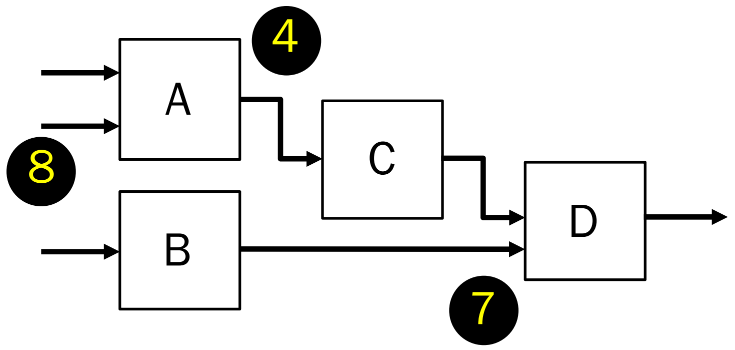


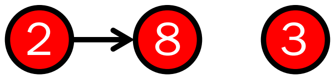
Discrete Event Simulation



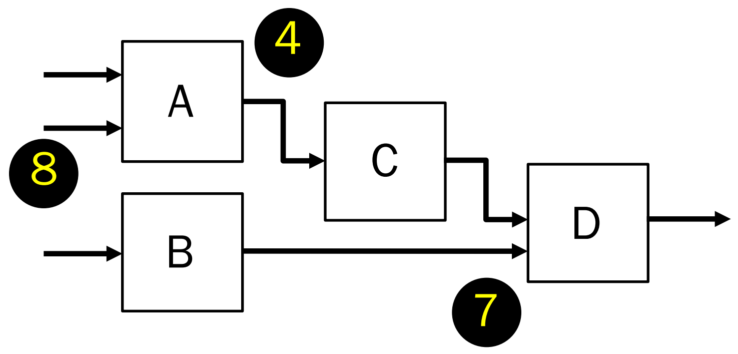
$$\begin{tikzcd} 2 \arrow[r] & 8 & 3 \end{tikzcd}$$

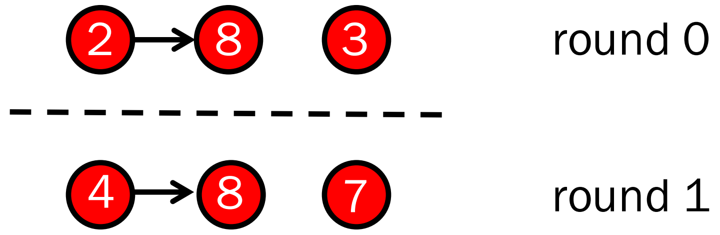
Discrete Event Simulation



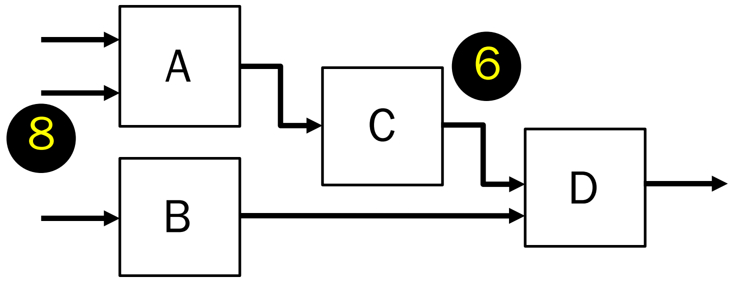


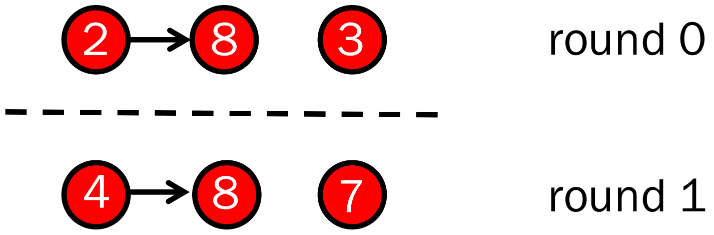
Discrete Event Simulation



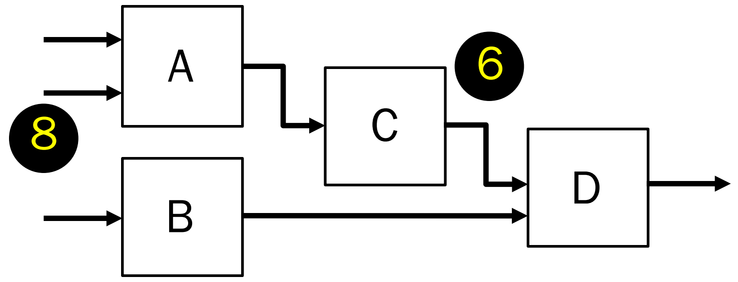


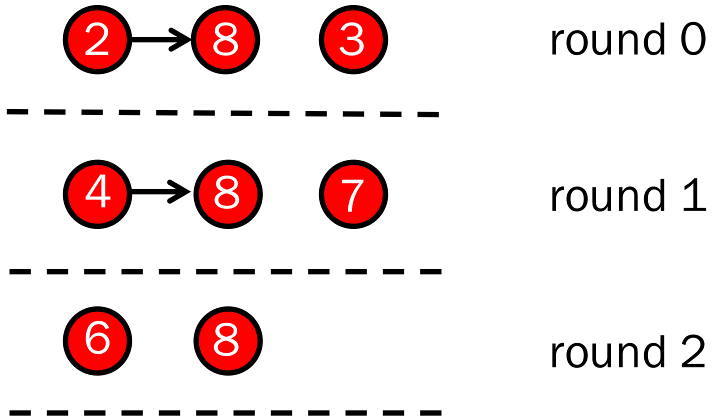
Discrete Event Simulation



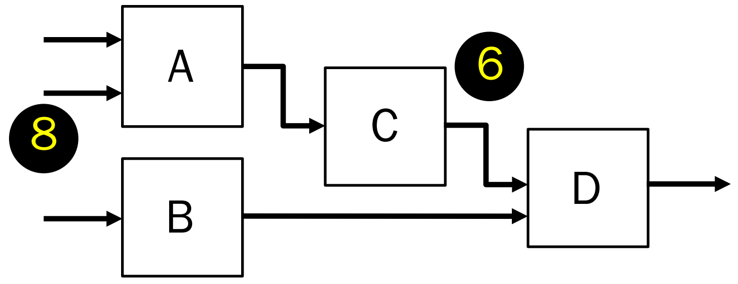


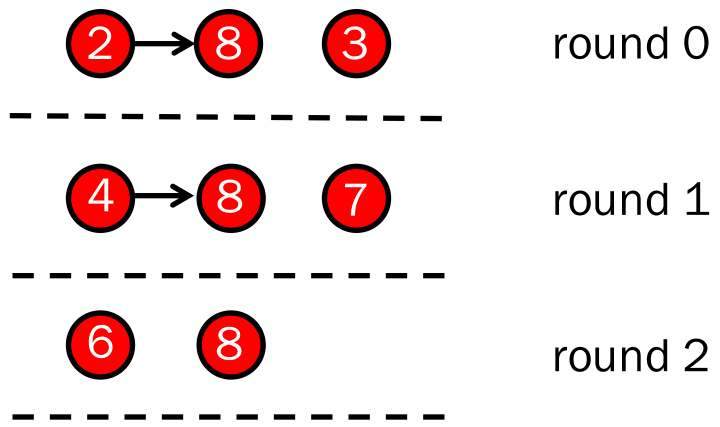
Discrete Event Simulation





Discrete Event Simulation





- Inadequacy of dependence graphs
  - Tasks are created dynamically
  - Not all sources are safe to execute
  - Task execution may require updating DAG

Dependence Graph

- A dependence graph is:
  - A DAG ***G*** for the program state ***σ***
  - An update rule ***U*** to produce the next ***G*** after executing task ***w***
    - Remove source

Execute sources

Dependence Graph

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \ldots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] \arrow[r, "w_1"] & \sigma_2 \arrow[u, dashed] \arrow[r, "w_2"] & \ldots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed]
\end{tikzcd}$$

Kinetic Dependence Graph

- A kinetic dependence graph is:
  - A DAG ***G*** for the program state ***σ***
  - A safe-source test ***P***
  - An update rule ***U*** to produce the next ***G*** after executing task ***w***
    - Remove source, update other tasks' dependencies, …

Execute sources

Dependence Graph

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "w_0"] & G_1 \arrow[r, Rightarrow, "w_1"] & G_2 \arrow[r, Rightarrow, "w_2"] & \ldots \arrow[r, Rightarrow, "w_{n-1}"] & G_n \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] \arrow[r, "w_1"] & \sigma_2 \arrow[u, dashed] \arrow[r, "w_2"] & \ldots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed]
\end{tikzcd}$$

Kinetic Dependence Graph

- A kinetic dependence graph is:
  - A DAG ***G*** for the program state ***σ***
  - A safe-source test ***P***
  - An update rule ***U*** to produce the next ***G*** after executing task ***w***
    - Remove source, update other tasks’ dependencies, …

Kinetic Dependence Graph

$$\begin{tikzcd}
\langle \sigma_0, G_0 \rangle \arrow[r, Rightarrow, "w_0"] & \langle \sigma_1, G_1 \rangle \arrow[r, Rightarrow, "w_1"] & \langle \sigma_2, G_2 \rangle \arrow[r, Rightarrow, "w_2"] & \cdots \arrow[r, Rightarrow, "w_{n-1}"] & \langle \sigma_n, G_n \rangle \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] \arrow[r, "w_1"] & \sigma_2 \arrow[u, dashed] \arrow[r, "w_2"] & \cdots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed]
\end{tikzcd}$$

Kinetic Dependence Graph

- A kinetic dependence graph is:
  - A DAG ***G*** for the program state ***σ***
  - A safe-source test ***P***
  - An update rule ***U*** to produce the next ***G*** after executing task ***w***
    - Remove source, update other tasks' dependencies, …

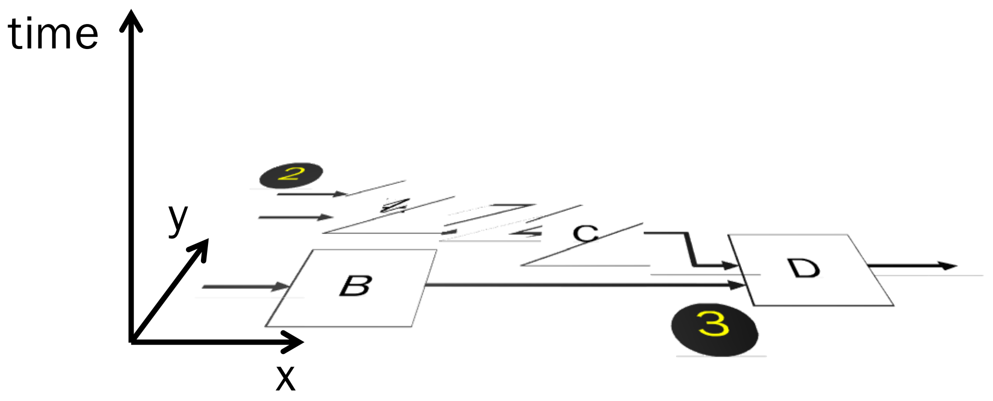
Execute safe sources

Kinetic Dependence Graph

$$\begin{tikzcd}
\langle \sigma_0, G_0 \rangle \arrow[r, Rightarrow, "w_0"] & \boxed{\langle \sigma_1, G_1 \rangle} \arrow[r, Rightarrow, "w_1"] & \langle \sigma_2, G_2 \rangle \arrow[r, Rightarrow, "w_2"] & \dots \arrow[r, Rightarrow, "w_{n-1}"] & \langle \sigma_n, G_n \rangle \\
\sigma_0 \arrow[u, dashed] \arrow[r, "w_0"] & \sigma_1 \arrow[u, dashed] \arrow[r, "w_1"] & \sigma_2 \arrow[u, dashed] \arrow[r, "w_2"] & \dots \arrow[r, "w_{n-1}"] & \sigma_n \arrow[u, dashed]
\end{tikzcd}$$

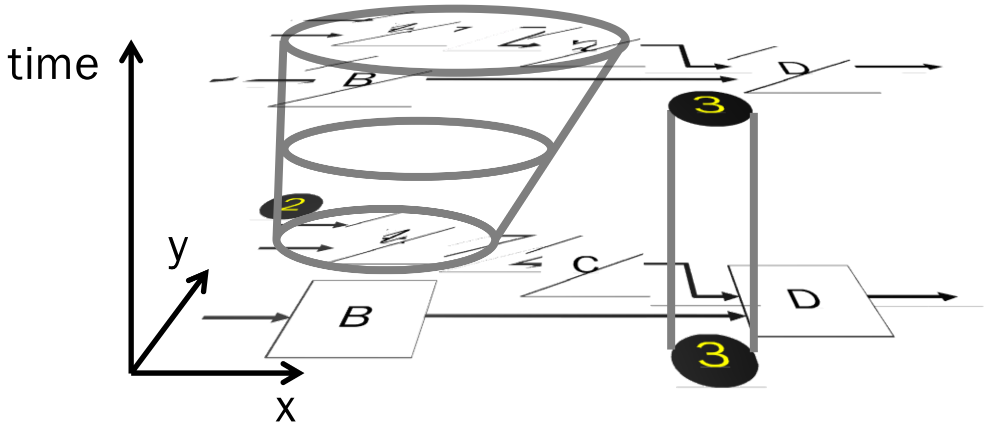
Safe Sources

- Application-specific cone of influence



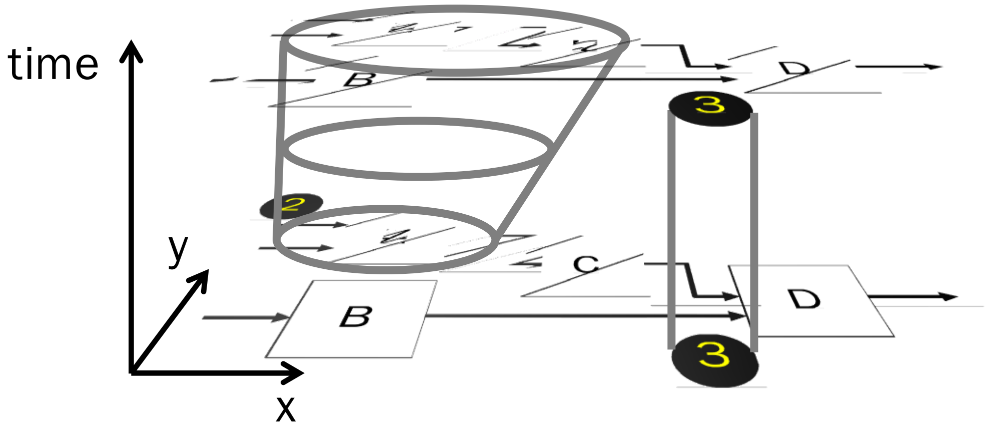
Safe Sources

- Application-specific cone of influence



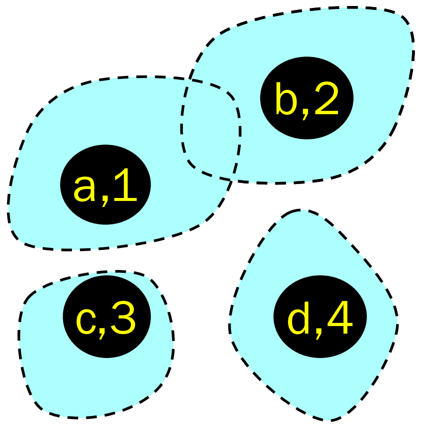
Safe Sources

- Application-specific cone of influence



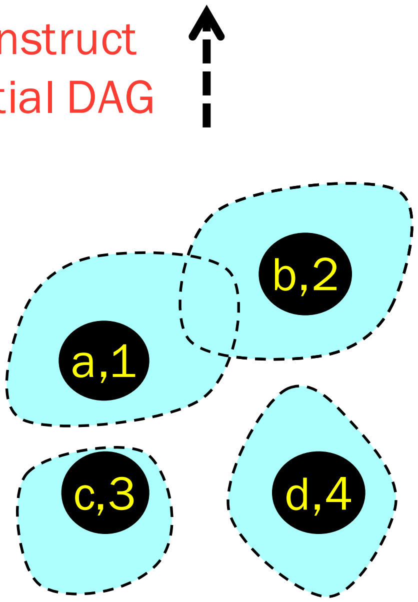
- In some algorithms, all sources are safe
  - Stable source algorithm

General KDG Executor

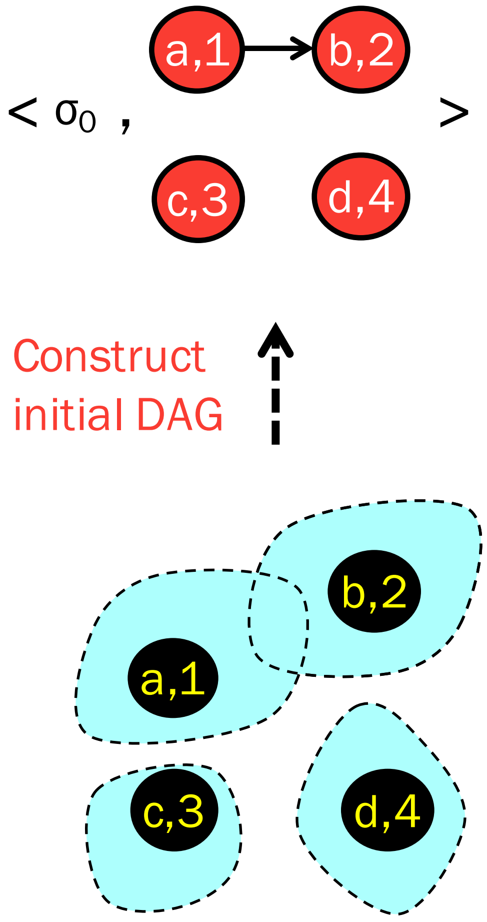


General KDG Executor

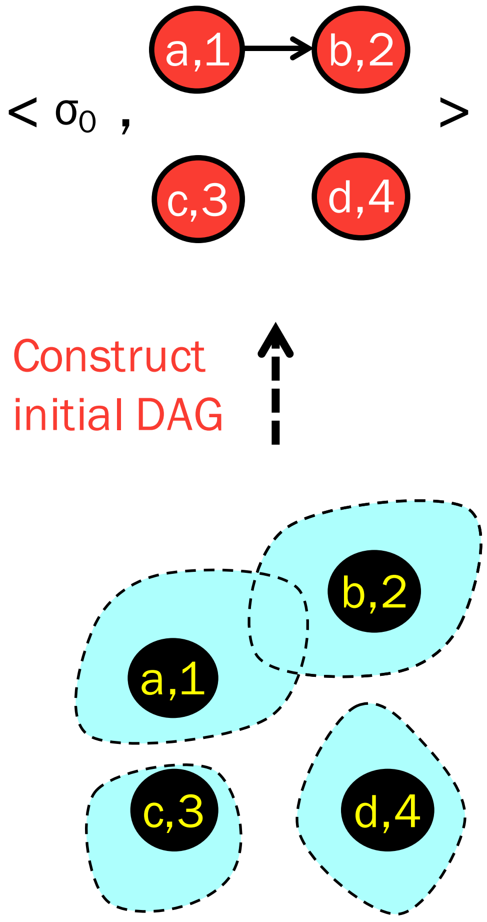
1. Construct initial DAG



General KDG Executor



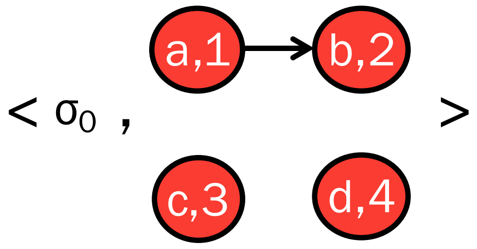
General KDG Executor



1. Construct initial DAG

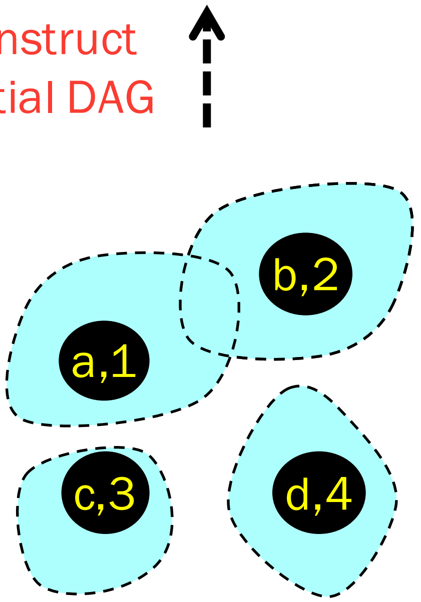
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

General KDG Executor



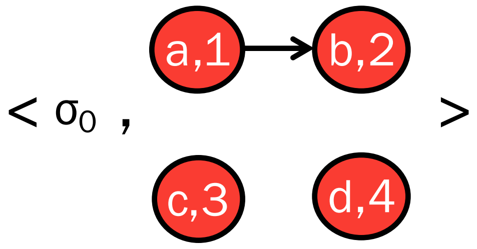
a,1

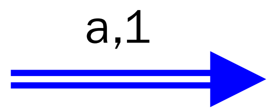
1. Construct initial DAG



2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

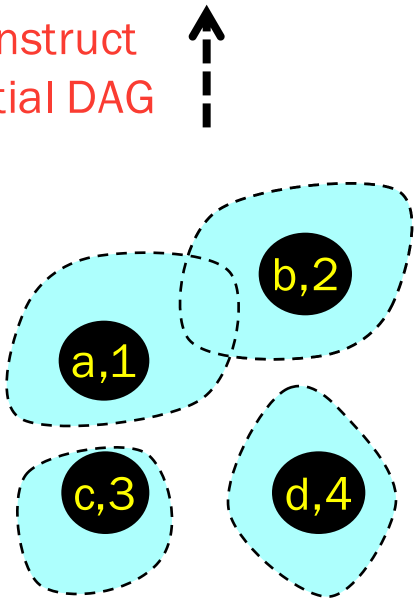
General KDG Executor



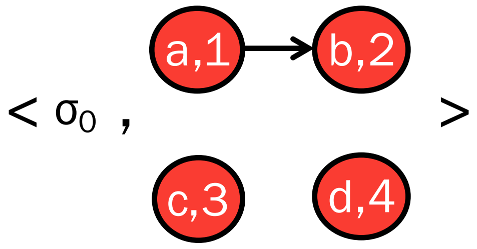


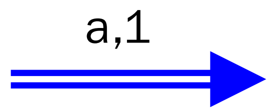
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

1. Construct initial DAG



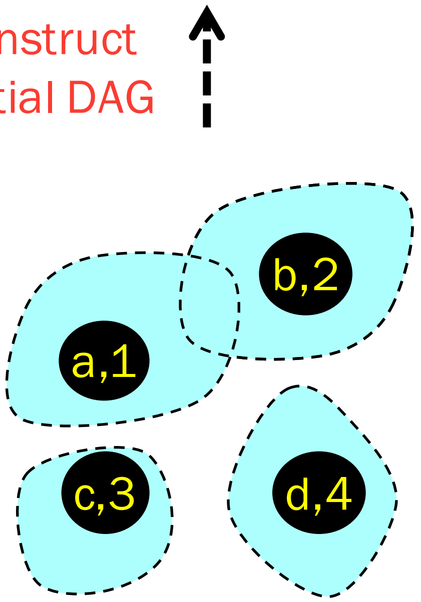
General KDG Executor

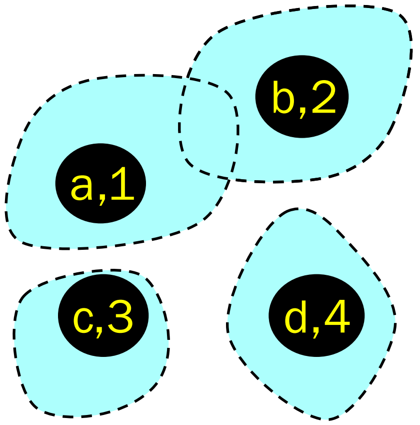




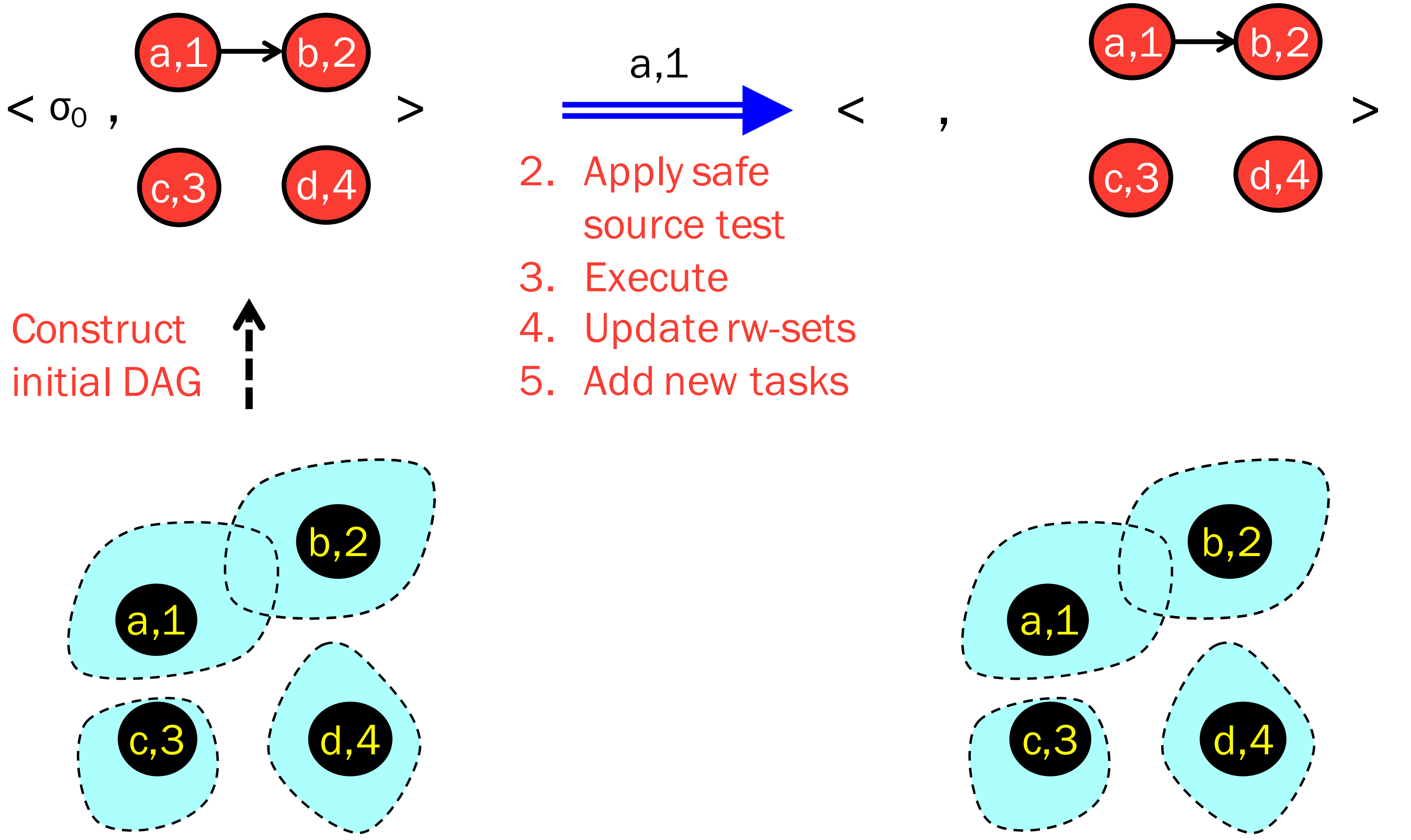
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

1. Construct initial DAG



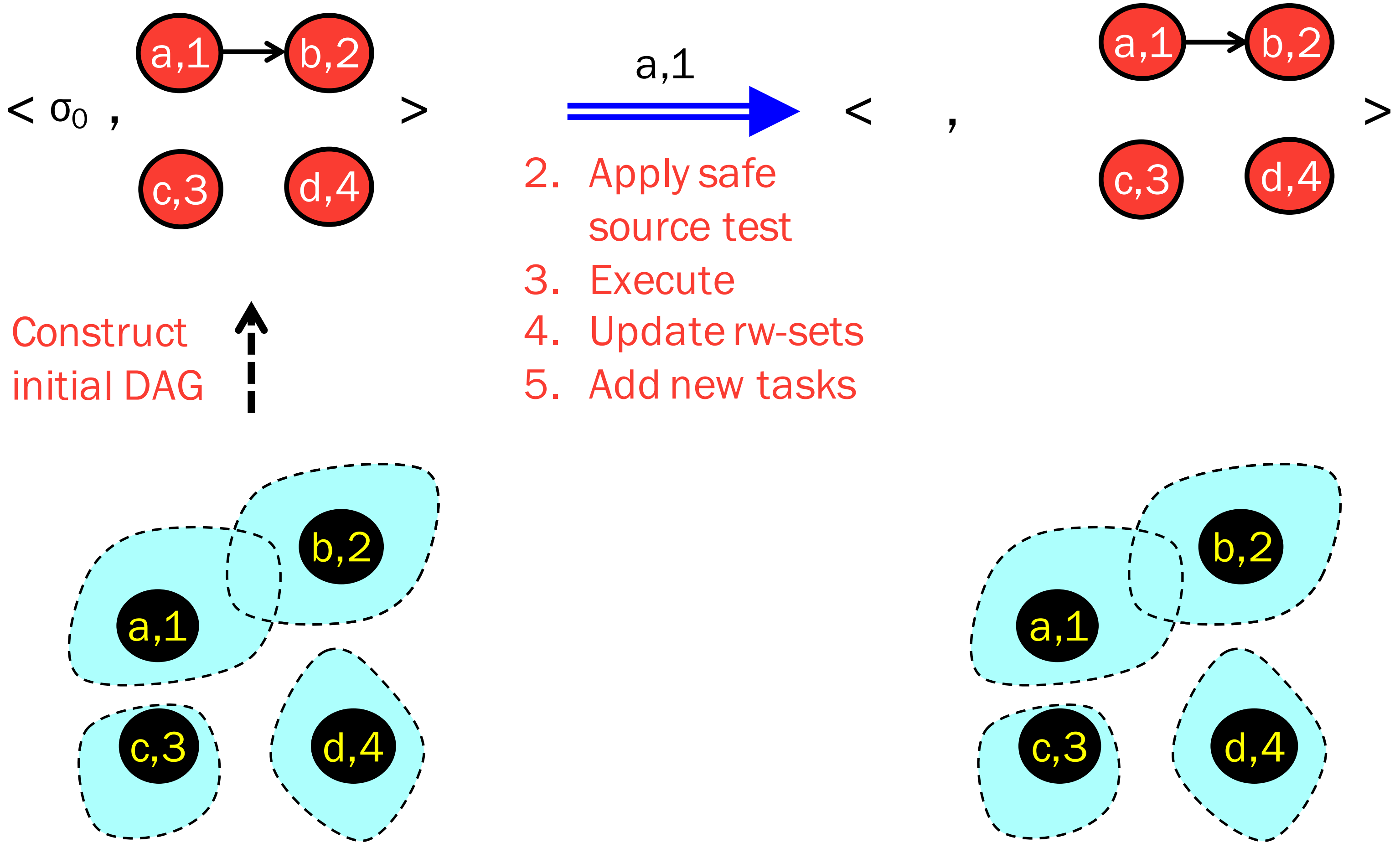


General KDG Executor



1. Construct initial DAG
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

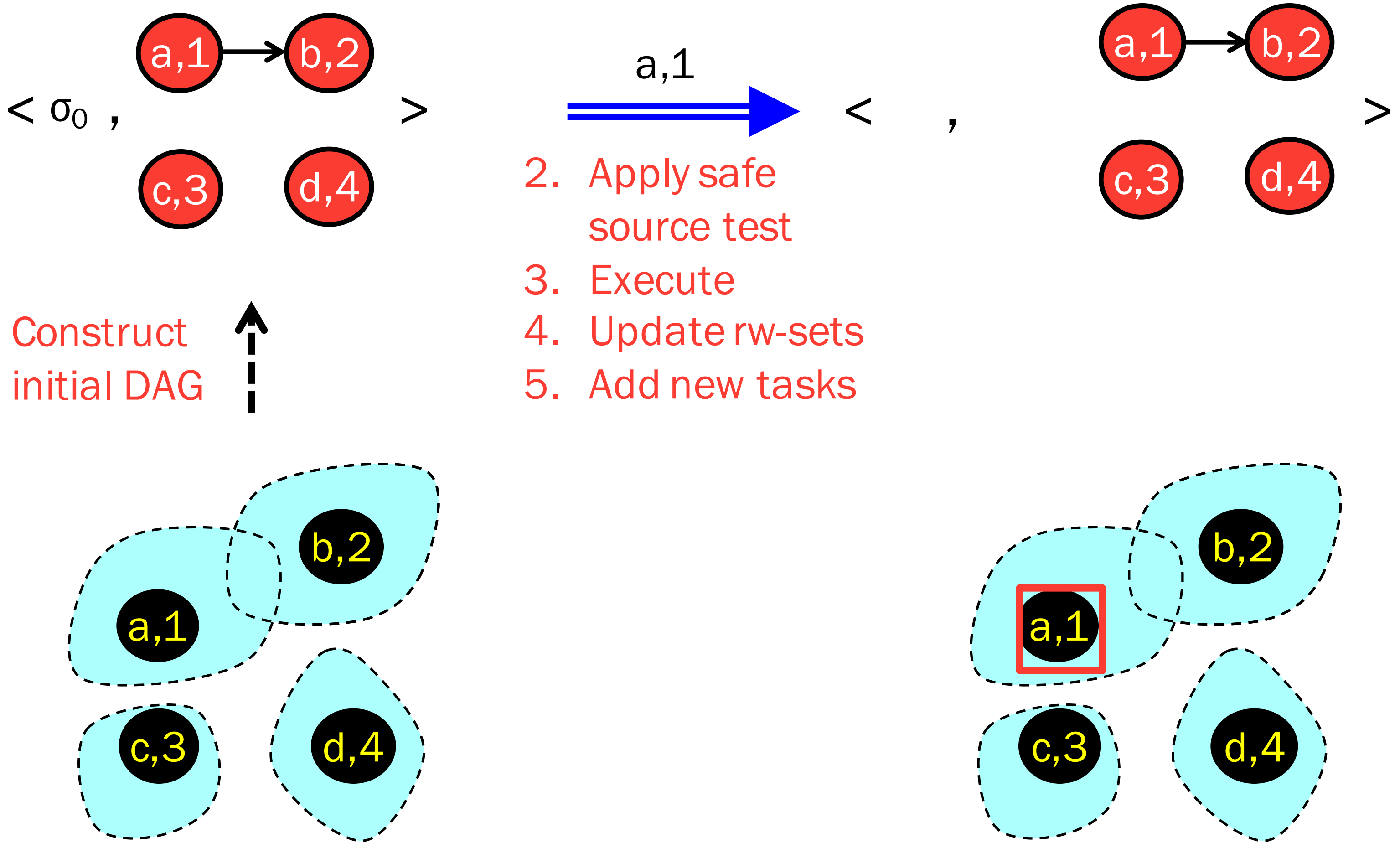
General KDG Executor



1. Construct initial DAG

2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

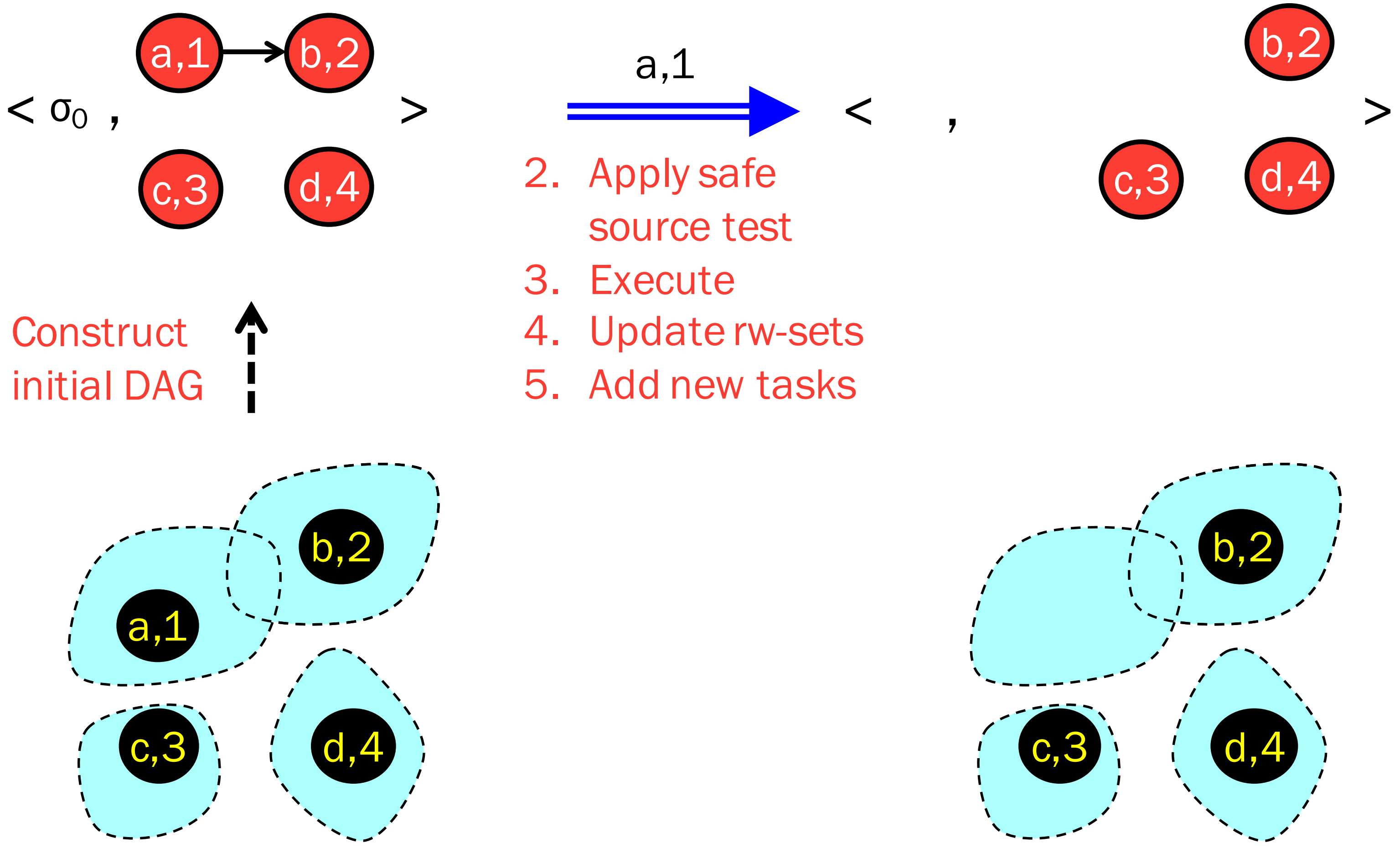
General KDG Executor



1. Construct initial DAG

2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

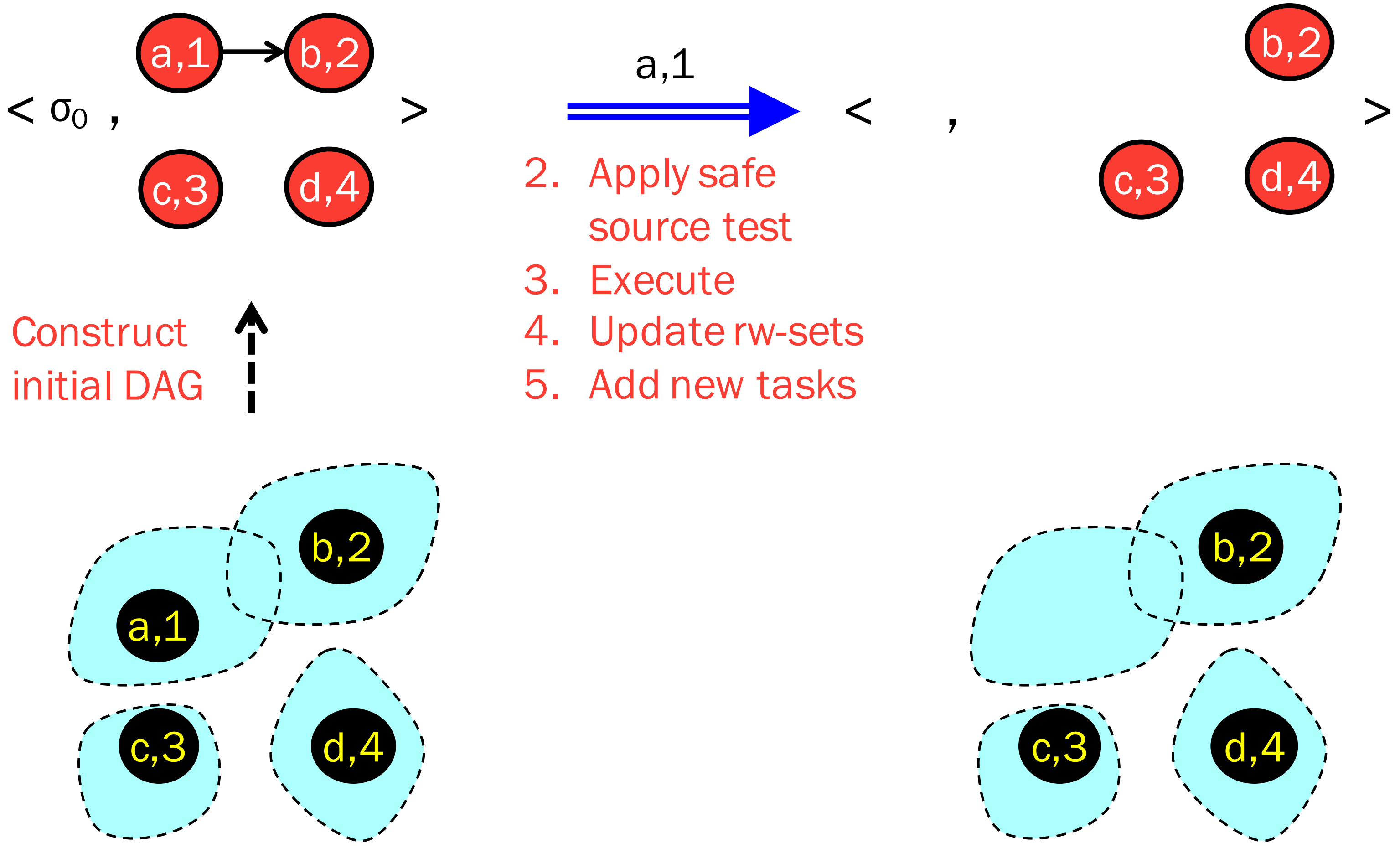
General KDG Executor



1. Construct initial DAG

2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

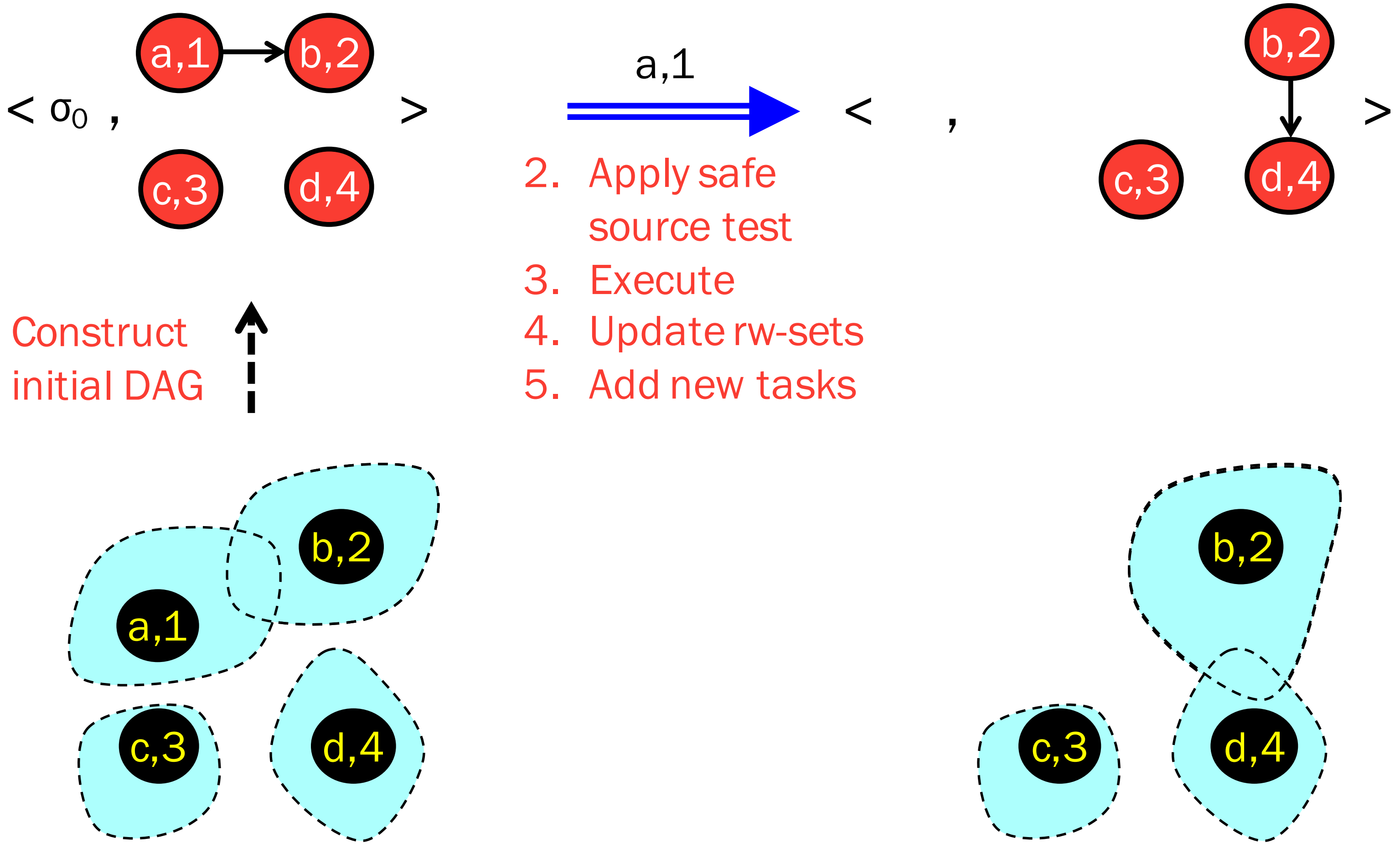
General KDG Executor



1. Construct initial DAG

2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

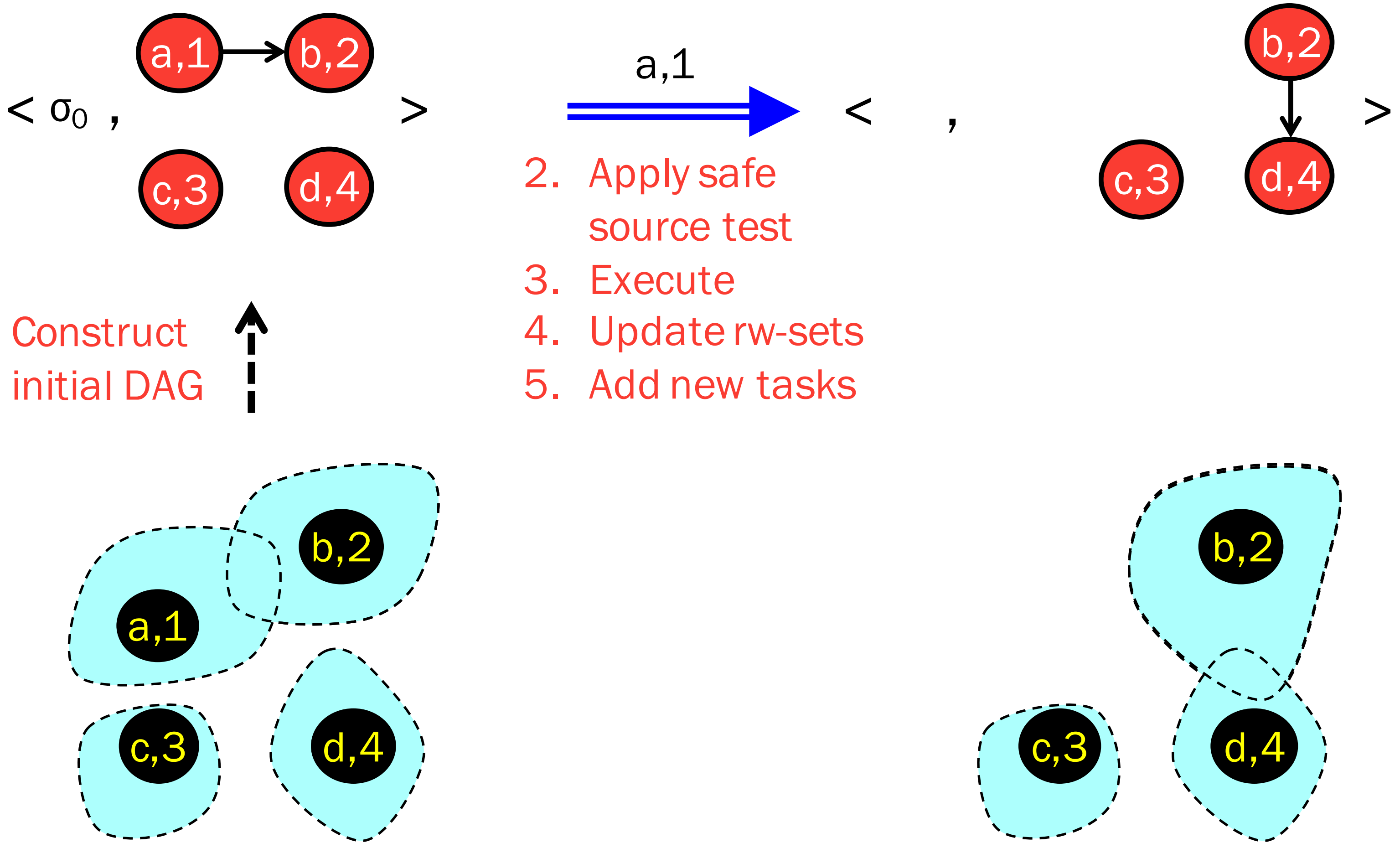
General KDG Executor



1. Construct initial DAG

2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

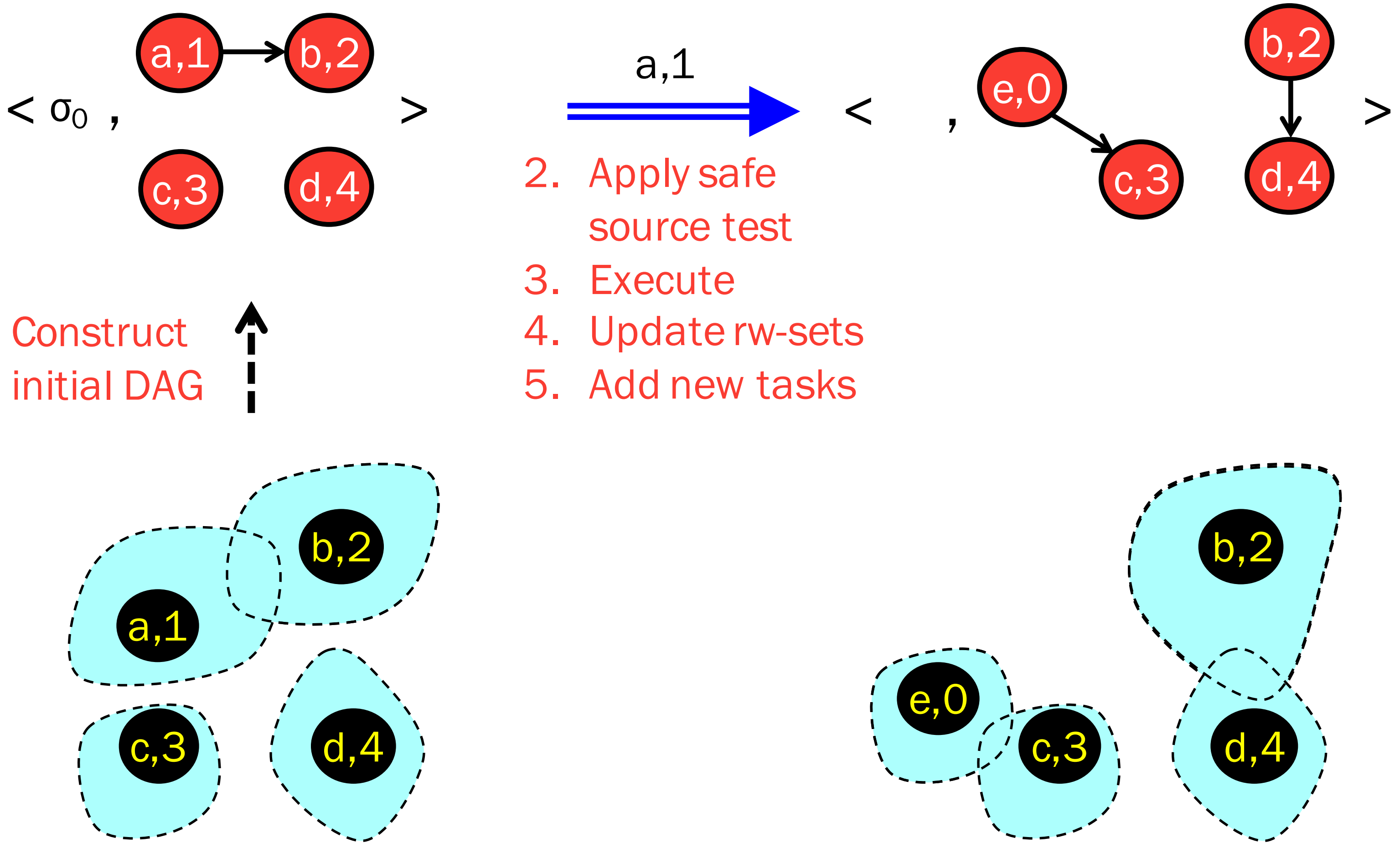
General KDG Executor



1. Construct initial DAG

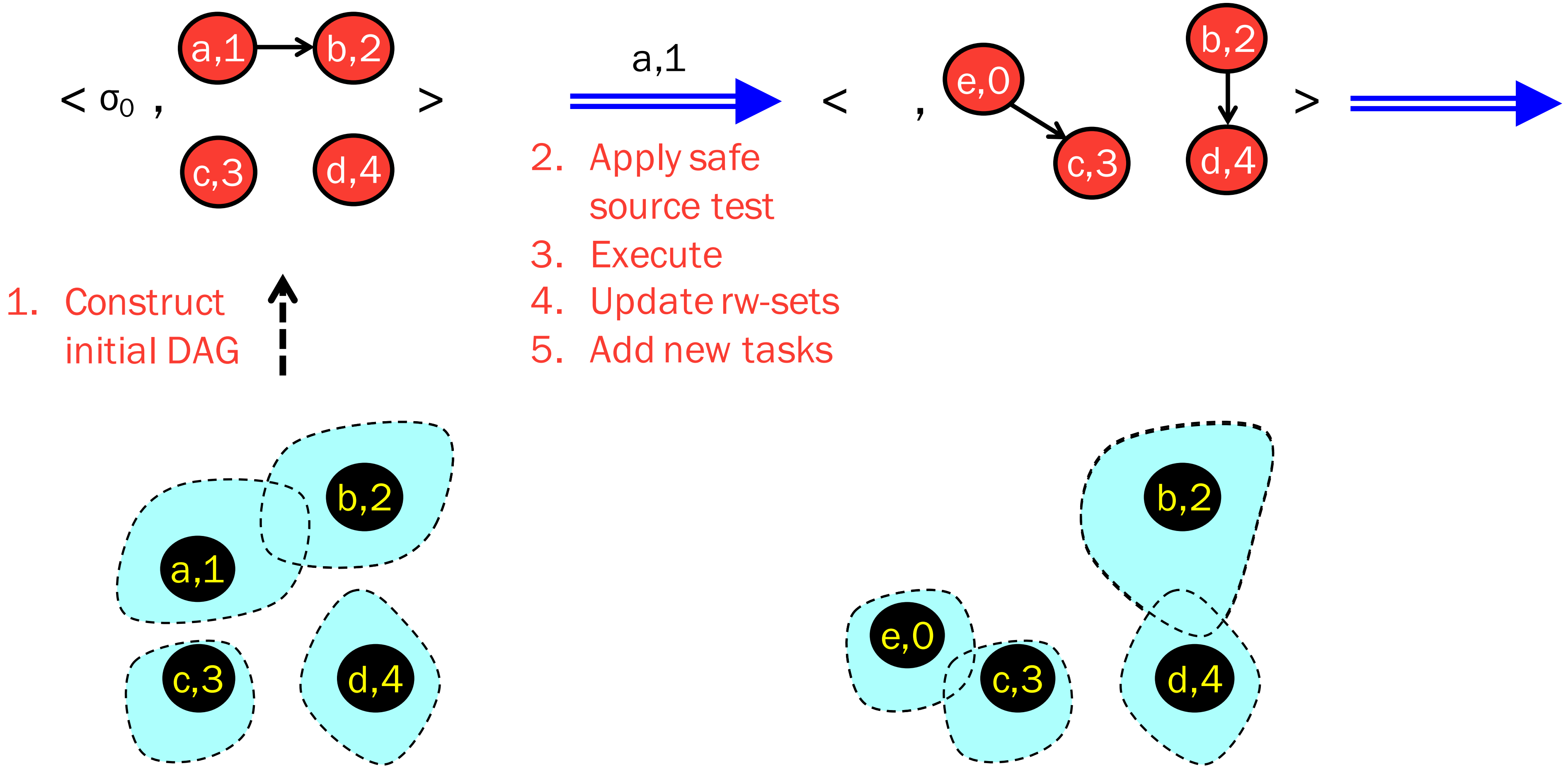
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

General KDG Executor



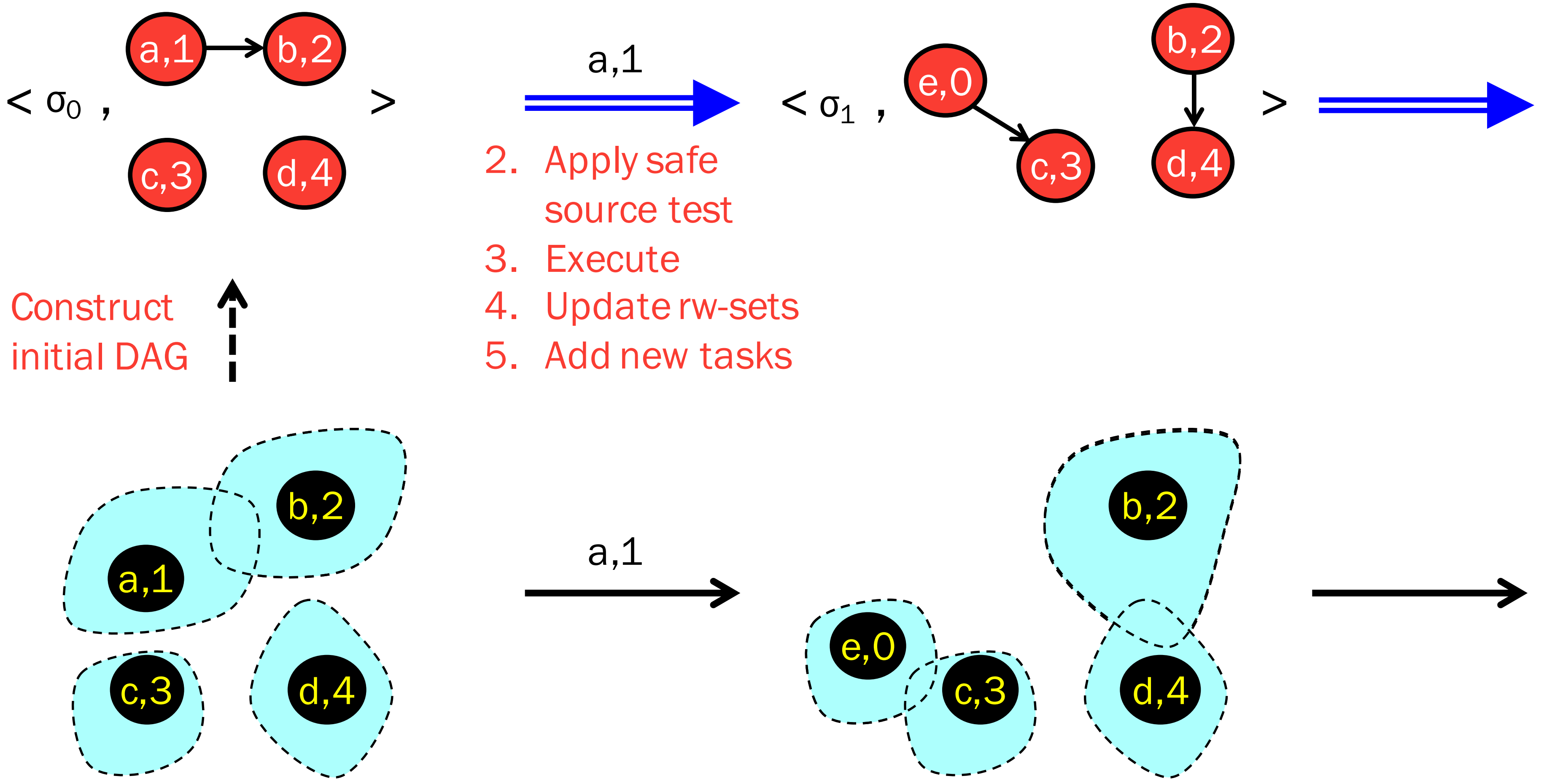
1. Construct initial DAG
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

General KDG Executor



1. Construct initial DAG
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

General KDG Executor



1. Construct initial DAG
2. Apply safe source test
3. Execute
4. Update rw-sets
5. Add new tasks

KDG Specializations

Opportunities

- Each step needs a barrier
- Task's rw-set potentially updated several times prior to execution

KDG Runtime

KDG Specializations

Opportunities

- Each step needs a barrier
- Task’s rw-set potentially updated several times prior to execution

Program Properties

KDG Runtime

Application Programmer

KDG Specializations

Opportunities

- Each step needs a barrier
- Task's rw-set potentially updated several times prior to execution

KDG Runtime

Program Properties

- Sources
  - In general: safe source test could inspect entire state
  - Local safe source test
  - Stable source (sources are safe)
- RW-sets
  - In general: task could change other tasks' RW-sets
  - Non-increasing RW-set
  - Structure-based RW-set
- New tasks
  - In general: task could create new tasks at any priority
  - Monotonic
  - No new tasks

Application Programmer

KDG Specializations

Opportunities

- Each step needs a barrier
- Task’s rw-set potentially updated several times prior to execution

Specializations

- Remove unnecessary barriers given program properties
- Construct KDG for prefix of tasks
- Use different DAG representations

KDG Runtime

Program Properties

- Sources
  - In general: safe source test could inspect entire state
  - Local safe source test
  - Stable source (sources are safe)
- RW-sets
  - In general: task could change other tasks’ RW-sets
  - Non-increasing RW-set
  - Structure-based RW-set
- New tasks
  - In general: task could create new tasks at any priority
  - Monotonic
  - No new tasks

Application Programmer

KDG Specializations

Opportunities

- Each step needs a barrier
- Task's rw-set potentially updated several times prior to execution

Specializations

- Remove unnecessary barriers given program properties
- Construct KDG for prefix of tasks
- Use different DAG representations

KDG Runtime

Program Properties

- Sources
  - In general: safe source test could inspect entire state
  - Local safe source test
  - Stable source (sources are safe)
- RW-sets
  - In general: task could change other tasks' RW-sets
  - Non-increasing RW-set
  - Structure-based RW-set
- New tasks
  - In general: task could create new tasks at any priority
  - Monotonic
  - No new tasks

Application Programmer

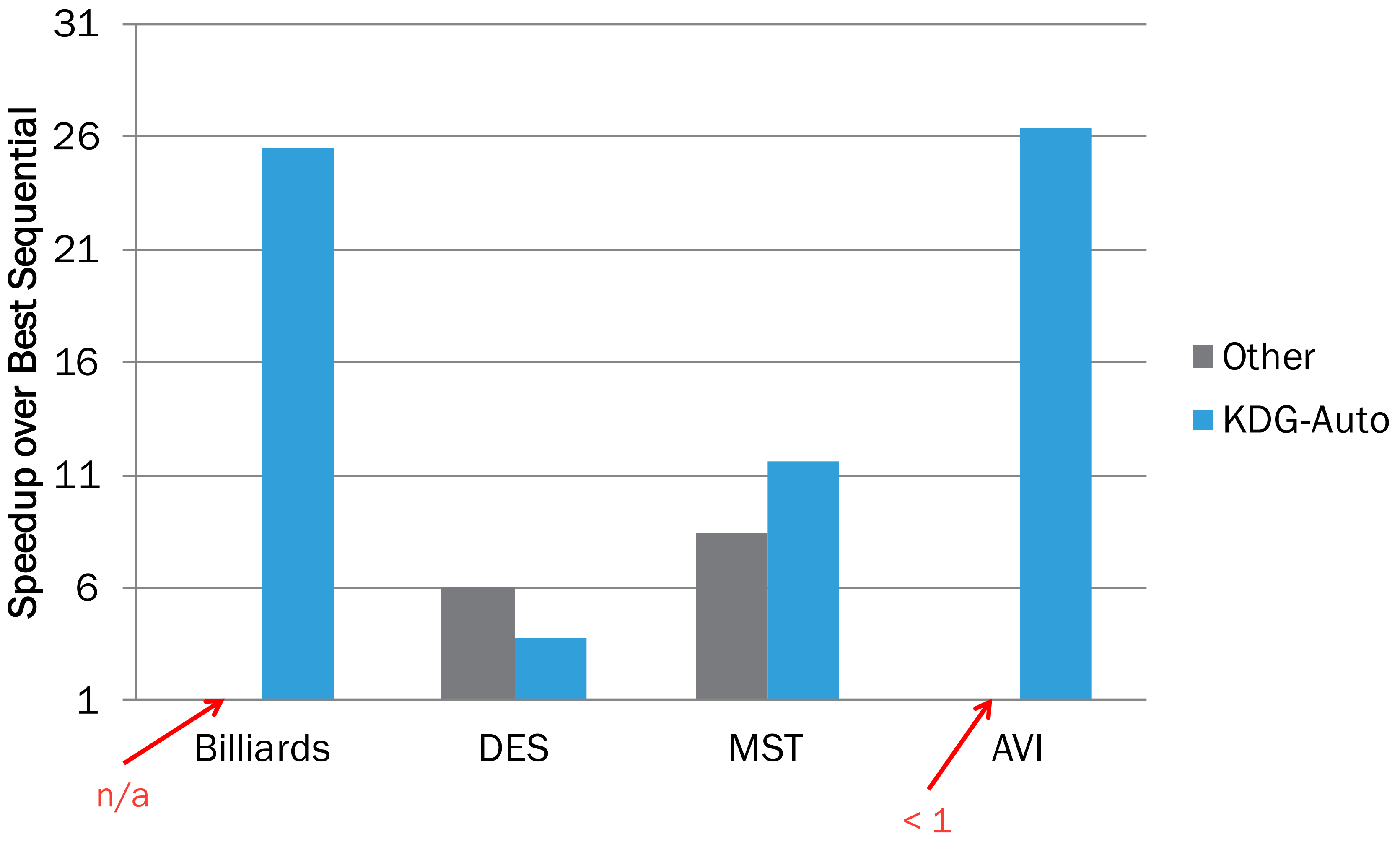
KDGs in Action

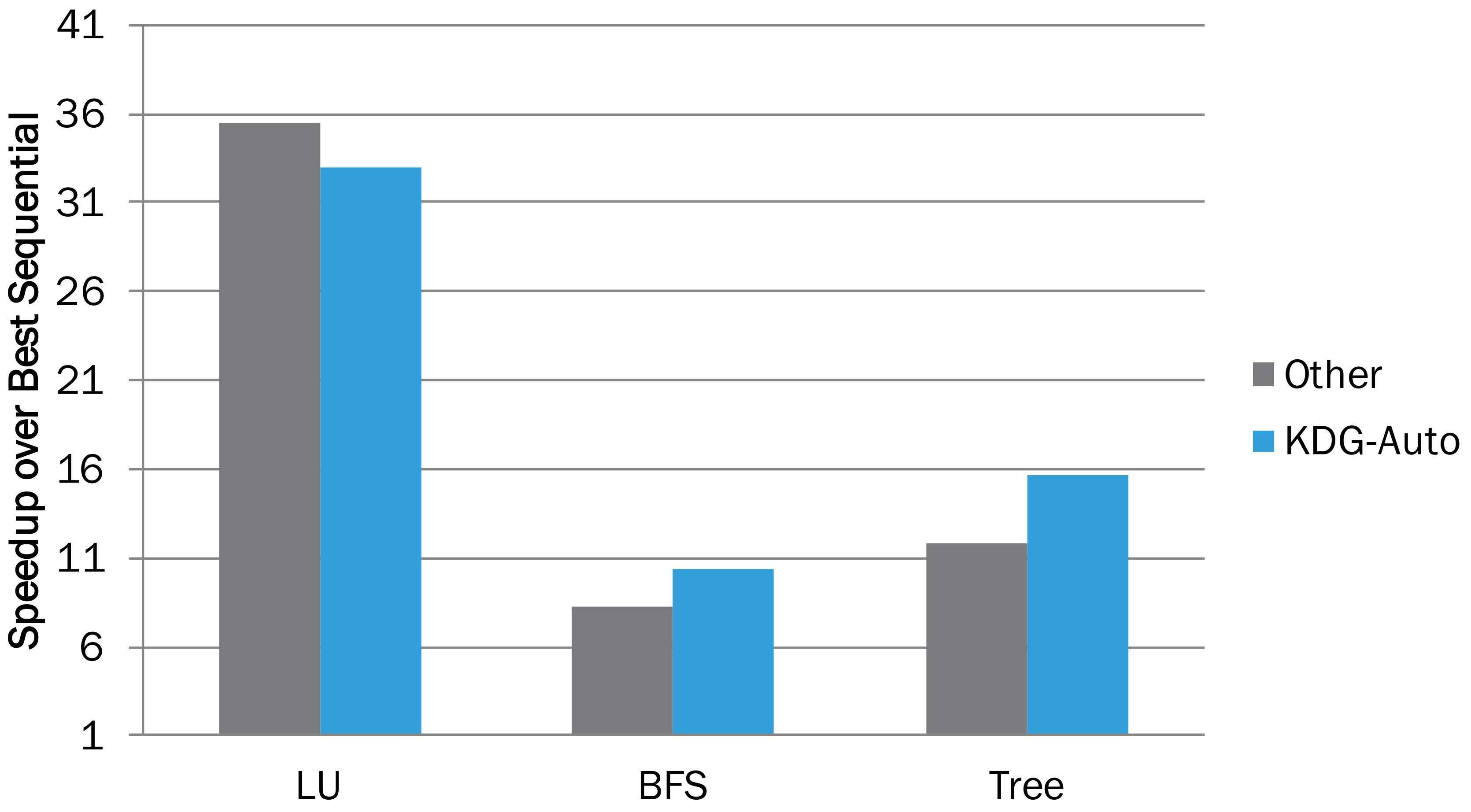
- Application programmer
  - Writes programs with ordered loops
    - Loop body must use provided data structure library
    - Loop properties provided via annotations
    - Loop body must read all elements before modifying any (cautious)
- KDG runtime (in Galois system)
  - Uses library API to track rw-sets at runtime
  - Selects executor based on annotations

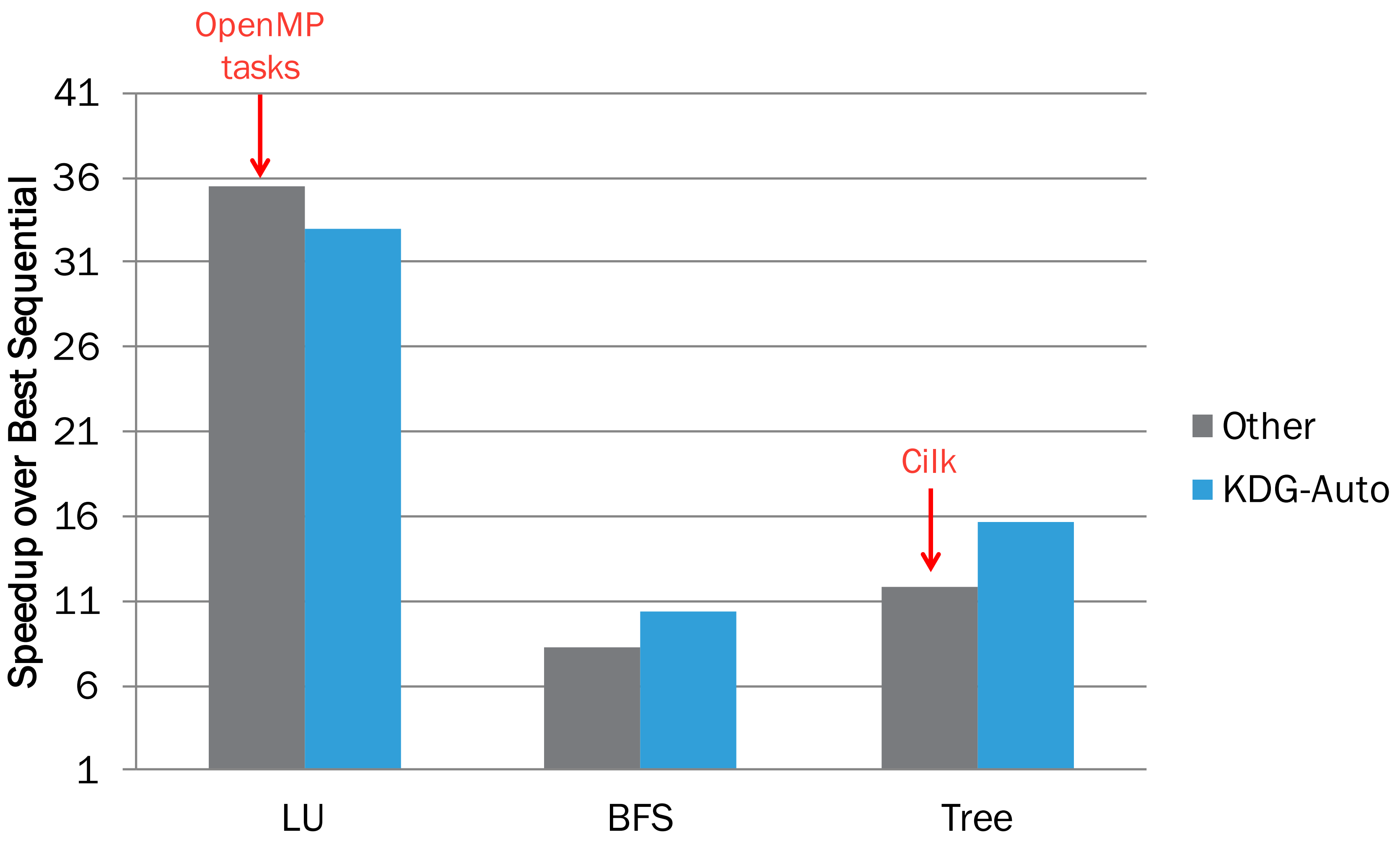
```
Graph g = ...
Set<Event> E = ...

@hasStructuredRWSets
@monotonic
foreach Event e in E orderedby e.t
  process(e, g)
  if *
    E.push(newE)
```

Evaluation

- 7 applications
  - From billiards simulation (hard-to-parallelize) to tree traversal (well supported by prior work)
- 2 types of programs
  - Other
    - 3rd-party handwritten, application-specific, OpenMP tasks, Cilk
  - KDG-Auto
    - Ordered loops + program property annotations
- 40-core, shared-memory machine


Speedup over Best Sequential
31
26
21
16
11
6
1
Billiards
DES
MST
AVI
n/a
< 1
■ Other
■ KDG-Auto


Speedup over Best Sequential
41
36
31
26
21
16
11
6
1
LU
BFS
Tree
Other
KDG-Auto



OpenMP tasks
Cilk
Speedup over Best Sequential
41
36
31
26
21
16
11
6
1
LU
BFS
Tree
Other
KDG-Auto

Summary

- Problem
  - Dependence graph scheduling is insufficient for many ordered programs
- Solution
  - Develop general KDG executor
  - Specialize executor according to small number of program properties

"It is a mistake to try to look too far ahead. The chain of destiny can only be grasped one link at a time."

-Winston Churchill

Kinetic Dependence Graphs

M. Amber Hassaan,

Donald Nguyen,

Keshav Pingali

The University of Texas at Austin