

The Co-design and Proof of an Available File System

Mahsa Najafzadeh

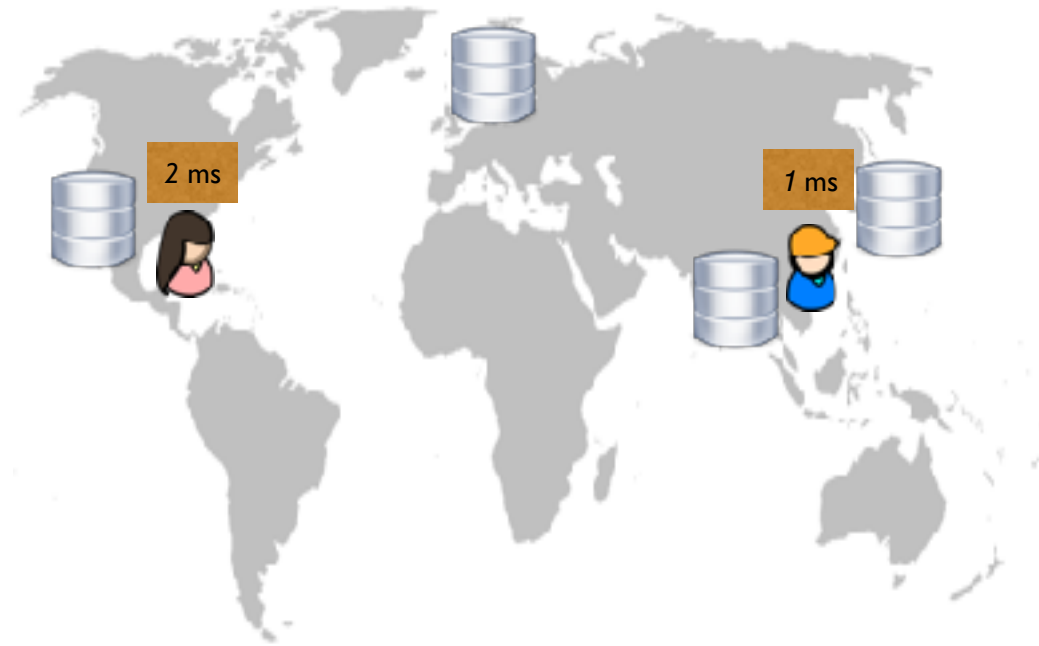
Marc Shapiro

INRIA & Université Pierre et Marie Curie

Replicated File System

File-replication

- Low latency
- High availability
- Fault tolerance



Requirement:

Maintain the file system application invariants

CISE Analysis

I. Static analysis tool: verifies integrity invariant of an application, above a weakly-consistent database

[Gotsman et al. POPL 2016 'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems]

CISE Rules to Prove Application is Correct

Commutativity:

Concurrent operations commute (convergence)

Effector Safety:

Every effect in isolation execution maintains the invariant (sequential safety)

Stability:

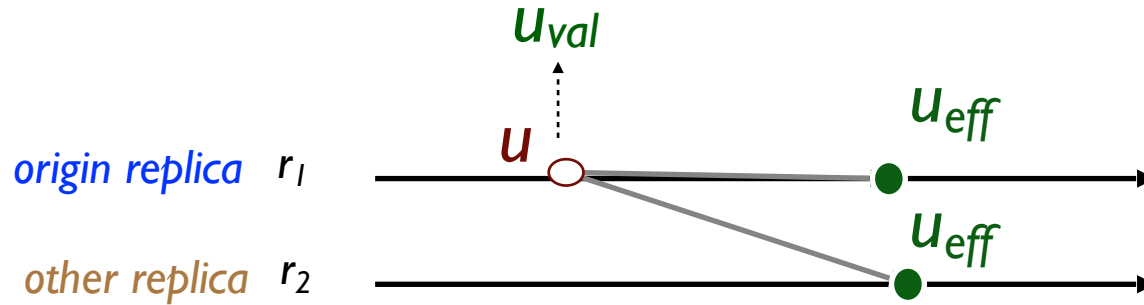
Preconditions are stable under concurrency (concurrent safety)

If satisfied: the invariant is guaranteed in every possible execution

System Model

- Full replication + Any number of replicas
- Each replica is sequential
 - Generator + Effector Operation model
- Causal + Exactly once delivery

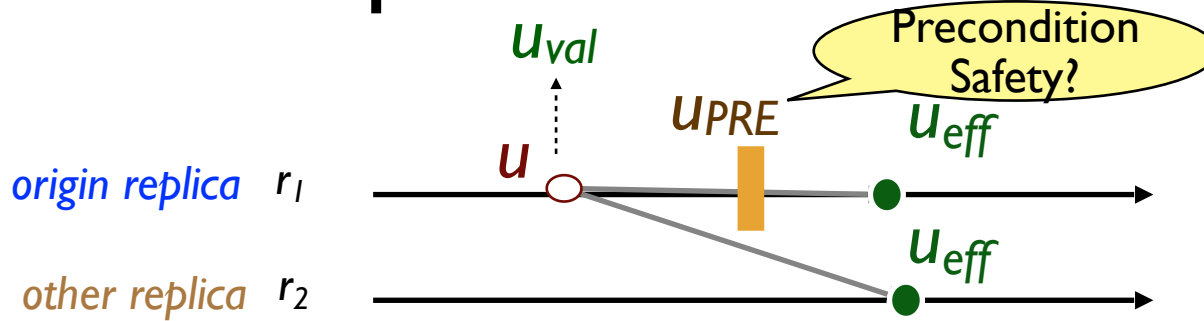
Operation Model



Generator (@origin): Read state from one copy and map operation u to :

- Return $\text{value}(u_{val})$

Operation Model



Generator (@origin): Read state from one copy and map operation u to :

- Effector (u_{eff}): State transformation applied at every replica

Concurrency Control

Tokens \approx concurrency control abstractions

Tokens = $\{\tau, \dots\}$

Conflict relation $\bowtie \subseteq \text{Tokens} \times \text{Tokens}$

Example - mutual exclusion tokens:

Tokens = $\{\tau\}$; $\tau \bowtie \tau$

An operation's generator may acquire a set of tokens

Operations associated with conflicting tokens cannot be concurrent

Sequential Specification of the File System

A directory: a map of *name* to file system *object* (INode)

Dir: Name \rightarrow INode

INode: Dir | File

Operations:

mkdir, addFile, rmFile, mvFile, updateFile, rmdir, mvDir.

Relations

- Parent relation: $(A \downarrow B)$: A is parent of B
- Ancestor relation $(root \downarrow^+ A)$: $root$ is an ancestor of A
- Least Common Ancestor of nodes A and B ($LCA(A,B)$)

Correctness Criteria

Convergent: do replicas that delivered the same updates have the same state?

Safe: are invariants preserved?

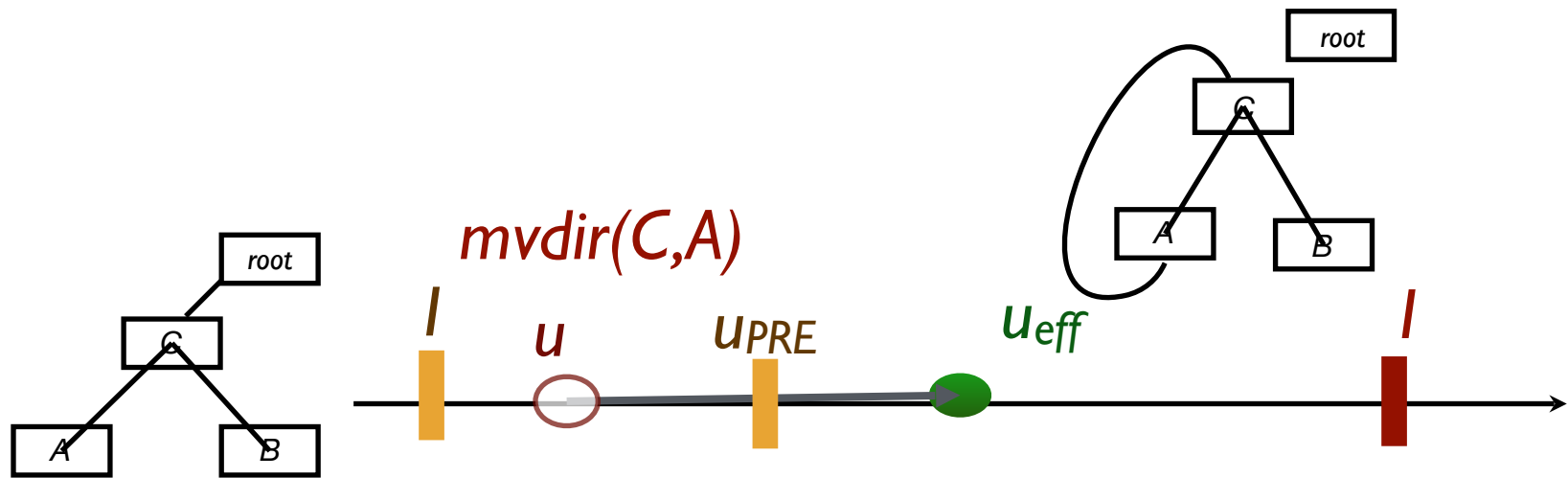
- **Sequential:** single operation in isolation maintains invariant
- **Concurrent execution** maintains invariant

Tree Invariant

- Has a known *Root*.
- *Root* is an ancestor of every inode in the tree (reachability).
- Every inode, which has a name has exactly one parent, except the *root*
- No cycle in the directory structure.
- Name of each inode is unique.
- No directory is a parent of itself.

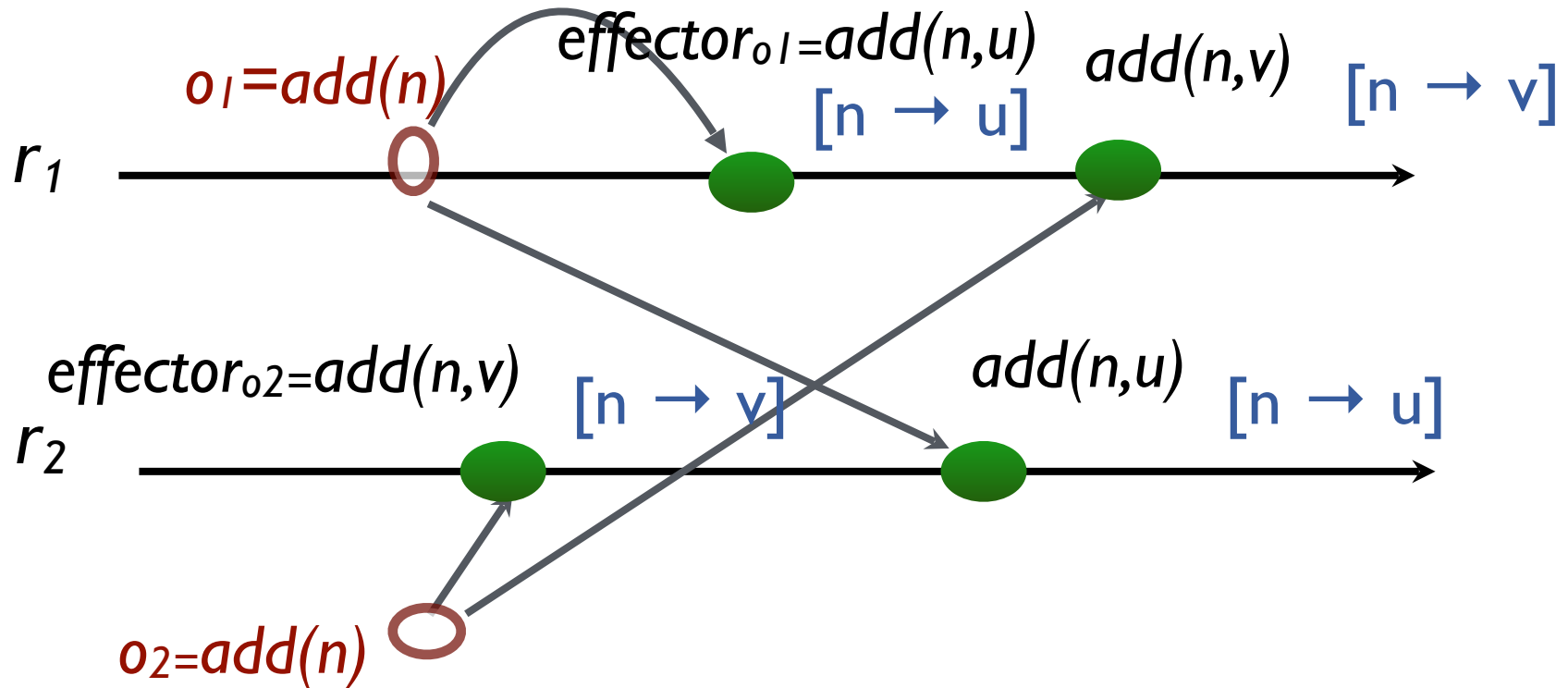
Effector Safety:

Example= move requires precondition



- *do not move directory under self*

Commutativity Rule: Counter-Example



Concurrent adding nodes under the same name to a directory are not commutative

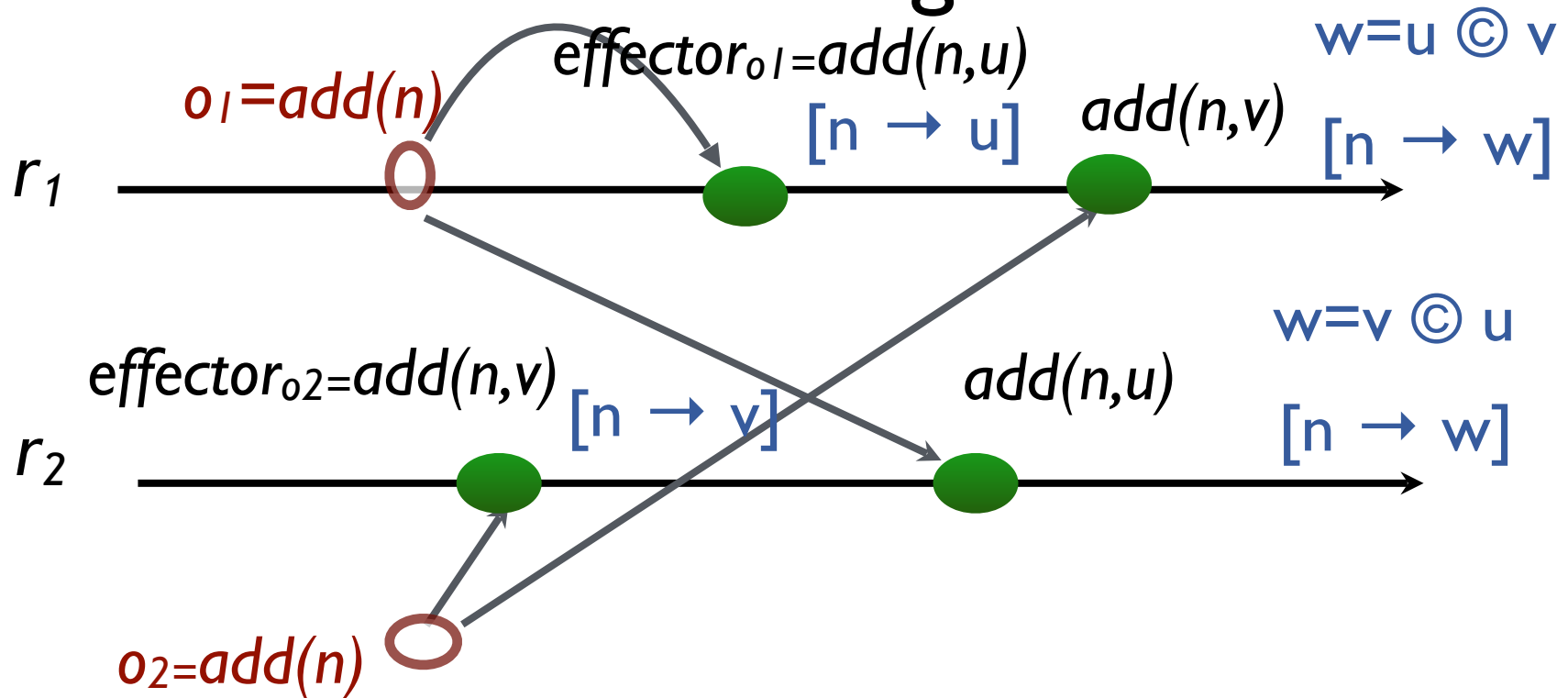
Concurrent Specification of the File System

Use replicated data types [Shapiro⁺ 2011]

Inodes implemented as CRDTs.

- Name Conflicts
 - Merge directories
 - Rename files
- Update/Remove Conflicts
 - Add-wins directory

Commutativity Rule: Co-design



Concurrently adding two directories under the same name to the same parent directory merge these two directories

A Commutative and Available File System

Name Conflicts

- Merge directories
- Rename files

Update/Remove Conflicts

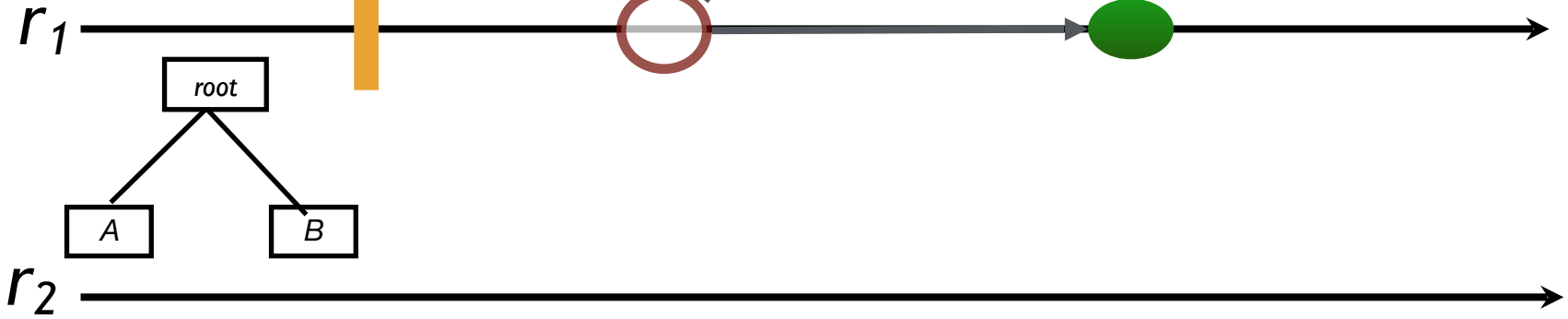
- add-wins directory

Stability Analysis: counter-example

B is NOT ancestor of A

$mvDir_{PRE}: \neg (B \downarrow^+ A)$

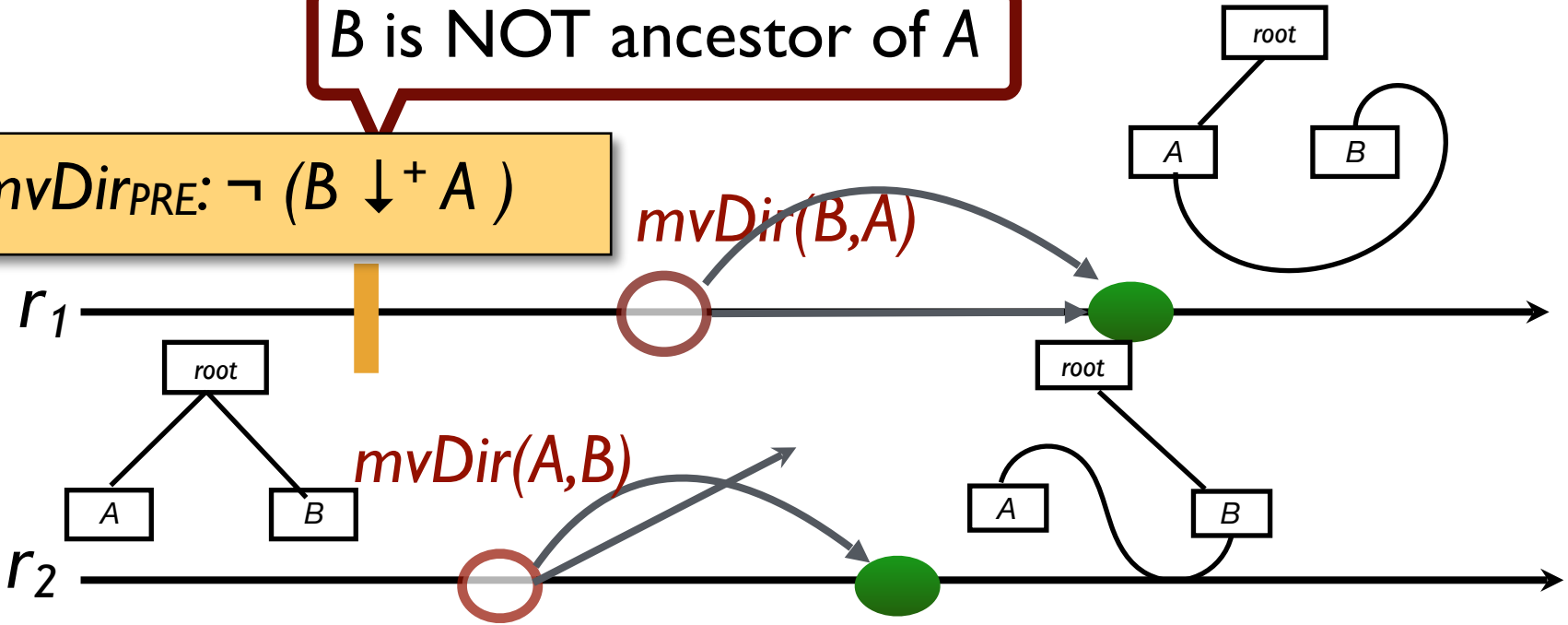
$mvDir(B,A)$



Stability Analysis: counter-example

B is NOT ancestor of A

$mvDir_{PRE}: \neg (B \downarrow^+ A)$

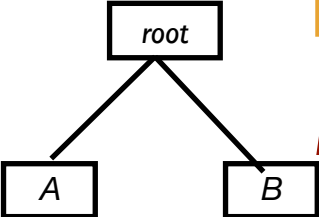
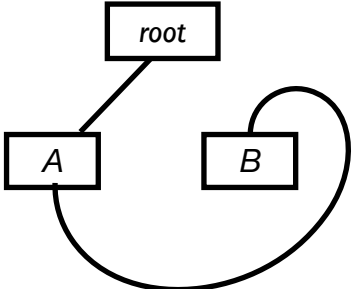


Stability Analysis: counter-example

B is NOT ancestor of A

$mvDir_{PRE}: \neg (B \downarrow^+ A)$

$mvDir(B,A)$



$mvDir(A,B)$

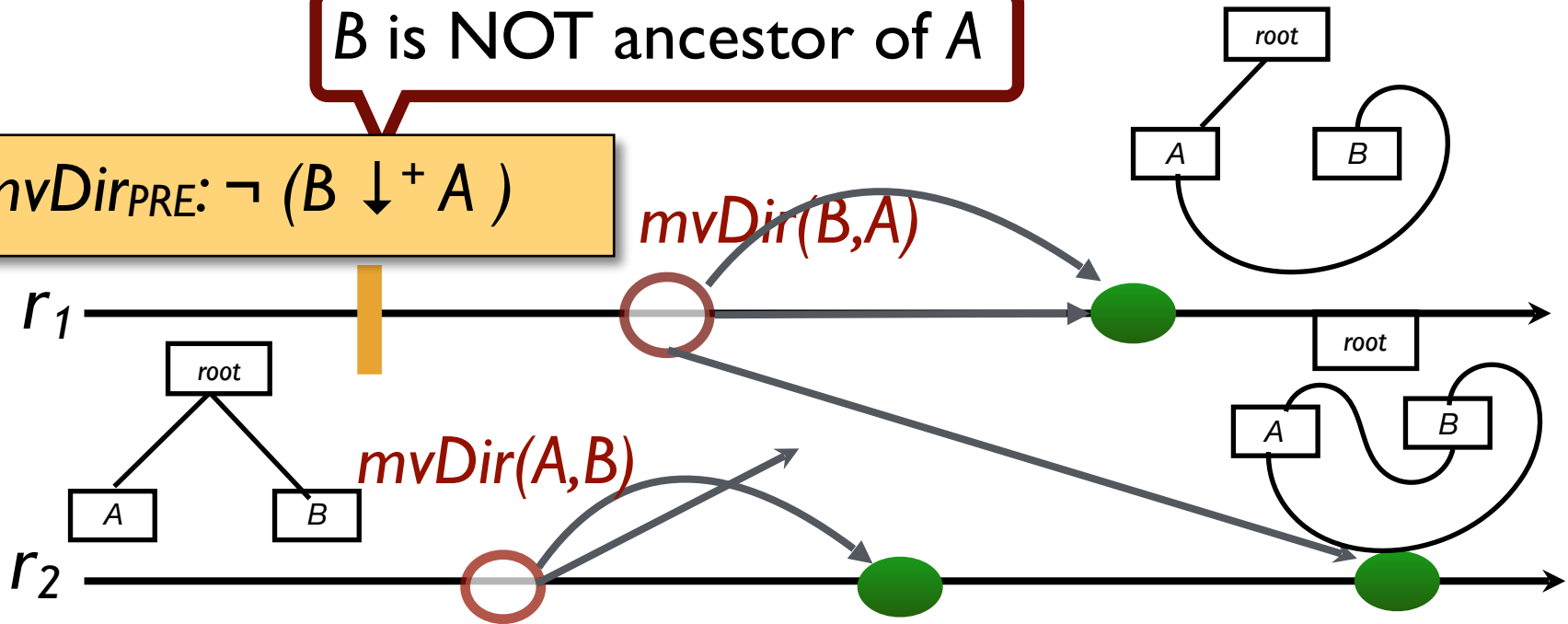


$mvDir_{PRE}: \neg (B \downarrow^+ A)$

Stability Analysis: counter-example+co-design

B is NOT ancestor of A

$mvDir_{PRE}: \neg (B \downarrow^+ A)$



- Weaken the specification, e.g., GeoFs
- Add some concurrency control, to avoid $mvDir \parallel mvDir$

Fully Asynchronous File System

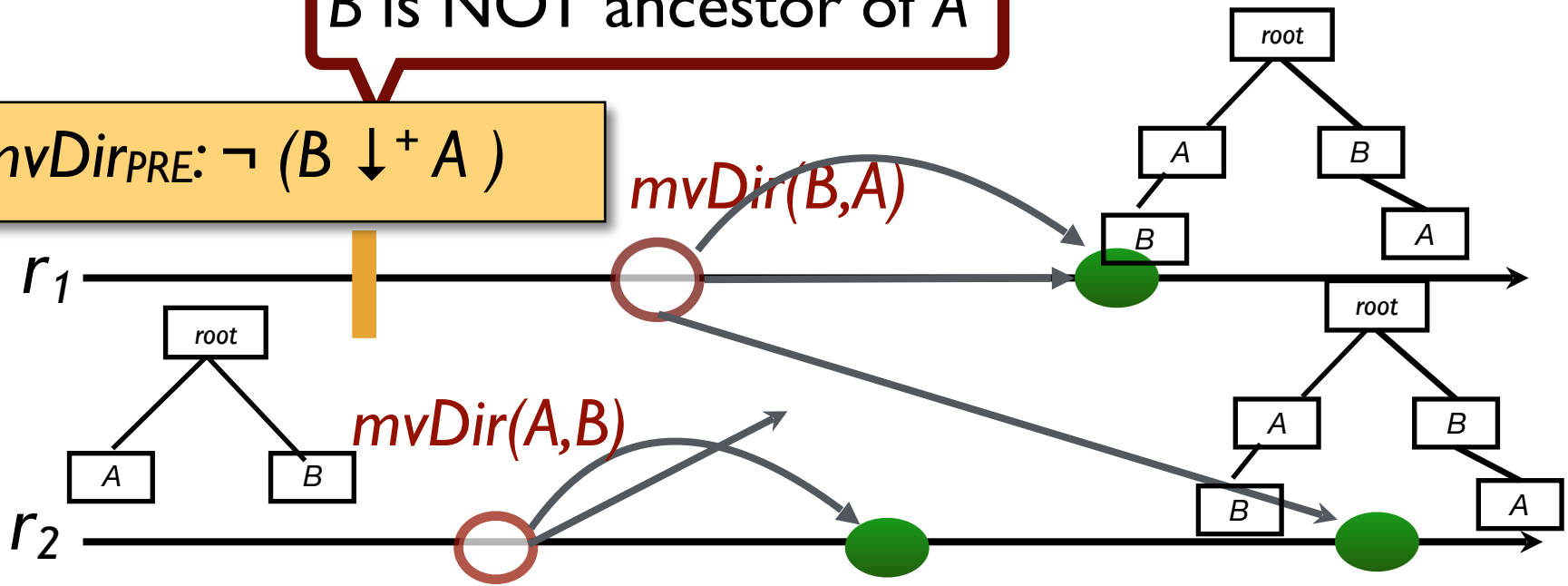
Allow concurrent moves.

Duplicate all the directories in the cycle(anomalous).

Stability Analysis: co-design

B is NOT ancestor of A

$mvDir_{PRE}: \neg (B \downarrow^+ A)$






Mostly Asynchronous File System

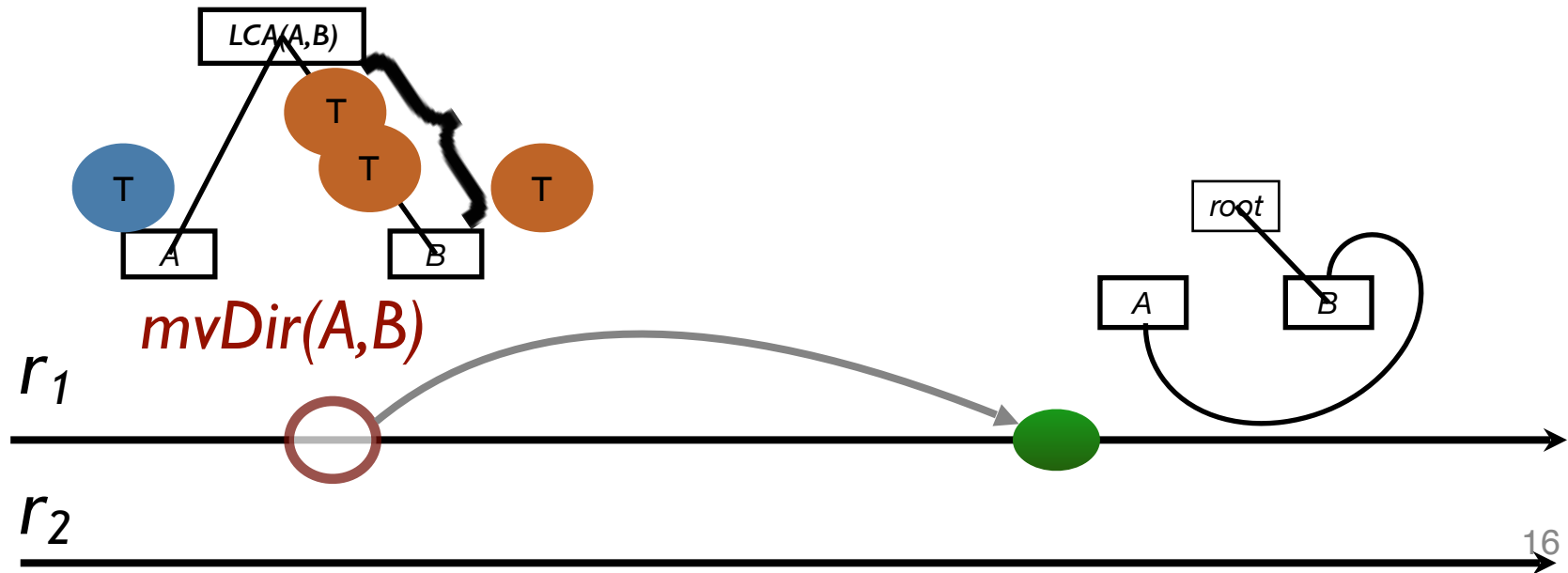
- Make move (partially) synchronous
- Add tokens, avoid $mvDir \parallel mvDir$
- A mutually exclusive token for each directory $d \in Dir$:

$$(\tau_{(d)} \bowtie \tau_{(d)})$$

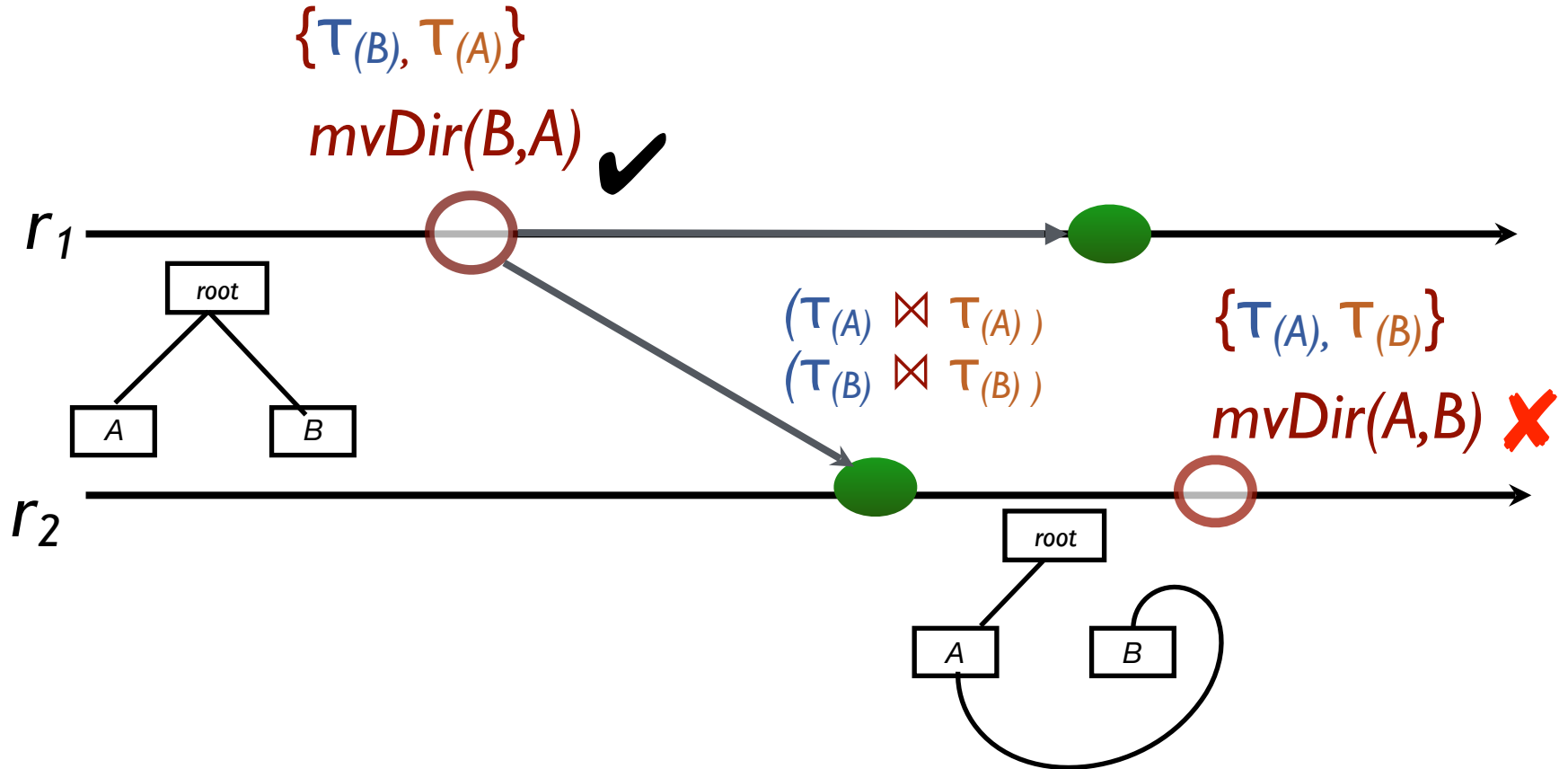
Specification of Move Tokens

Tokens for $mvDir(A,B)$:

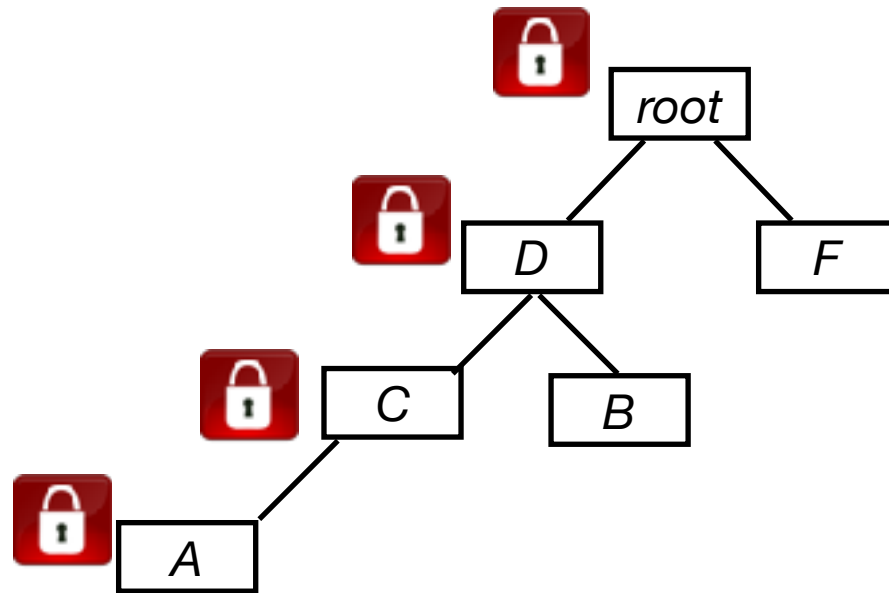
-  Token over Source directory A
-  Token over Destination directory B
-  Tokens over Ancestors up to LCA



Stability Analysis



GeoFS

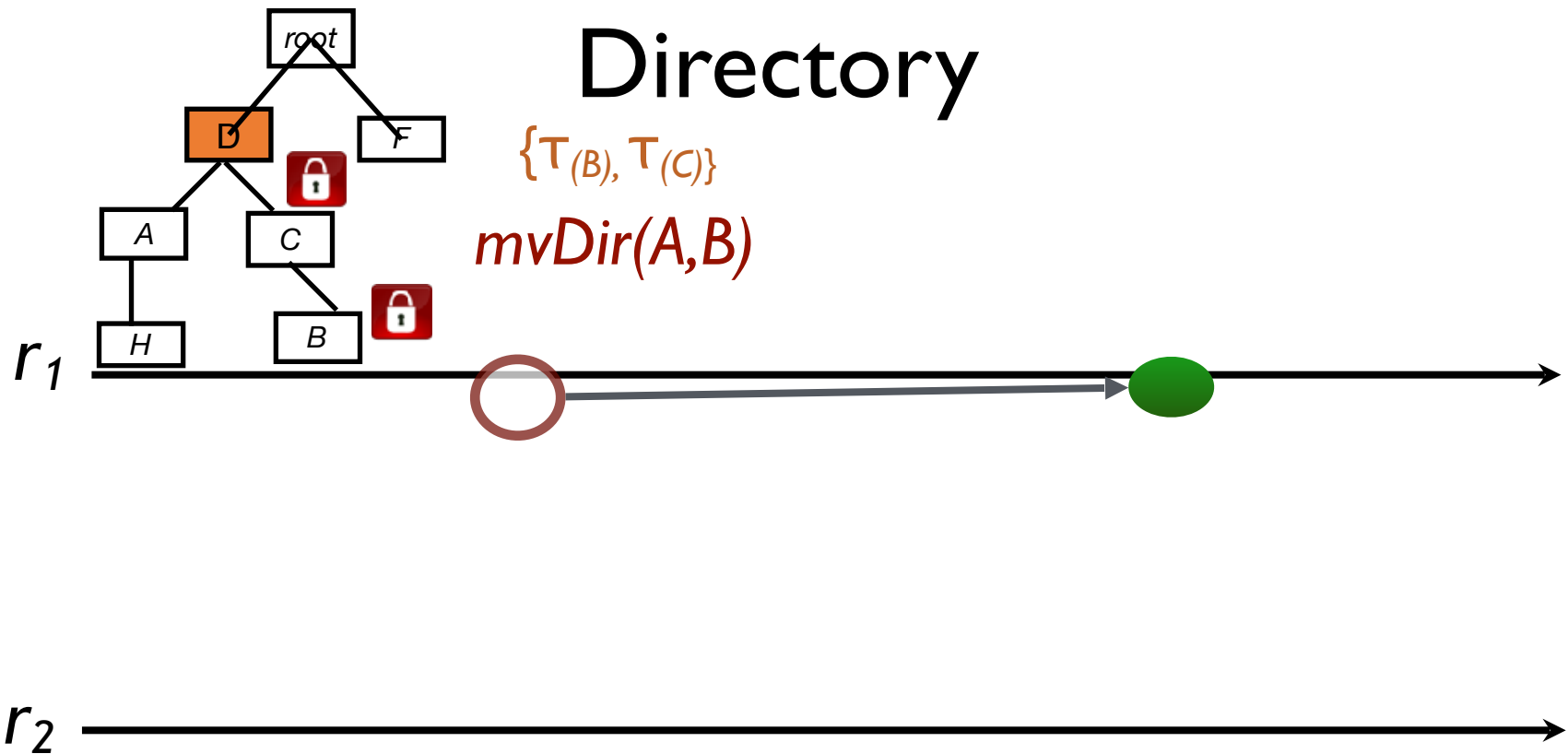


*concurrent mvDir(F,D)
AND mvDir(A,B)
is not possible*

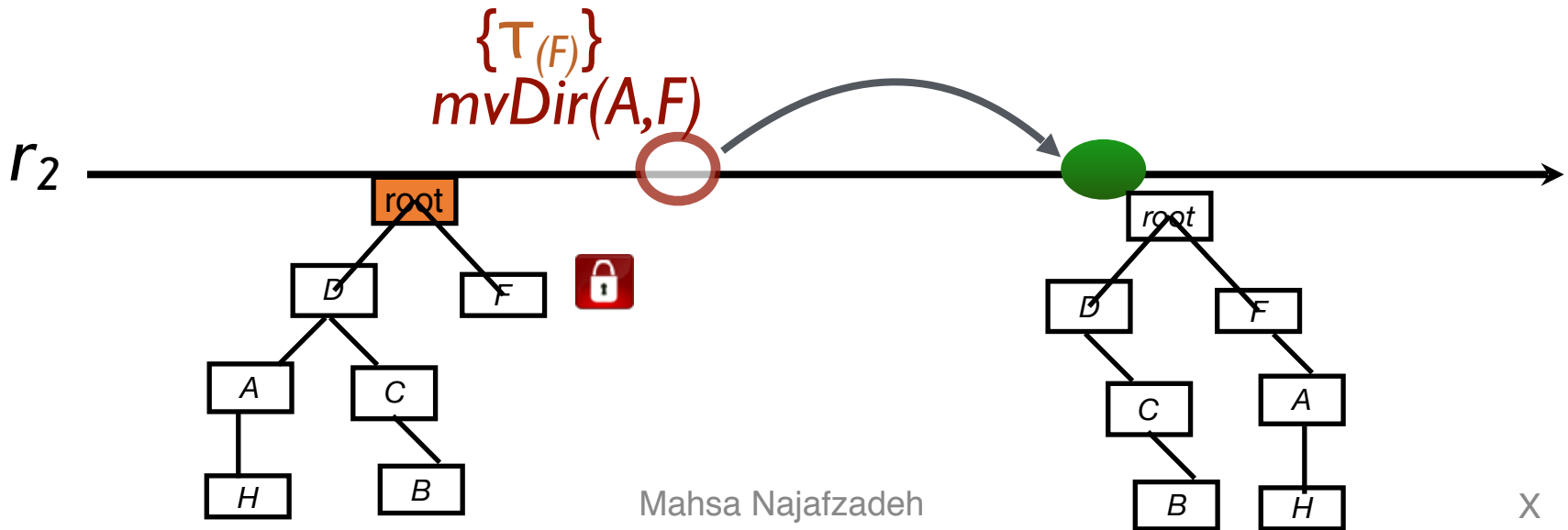
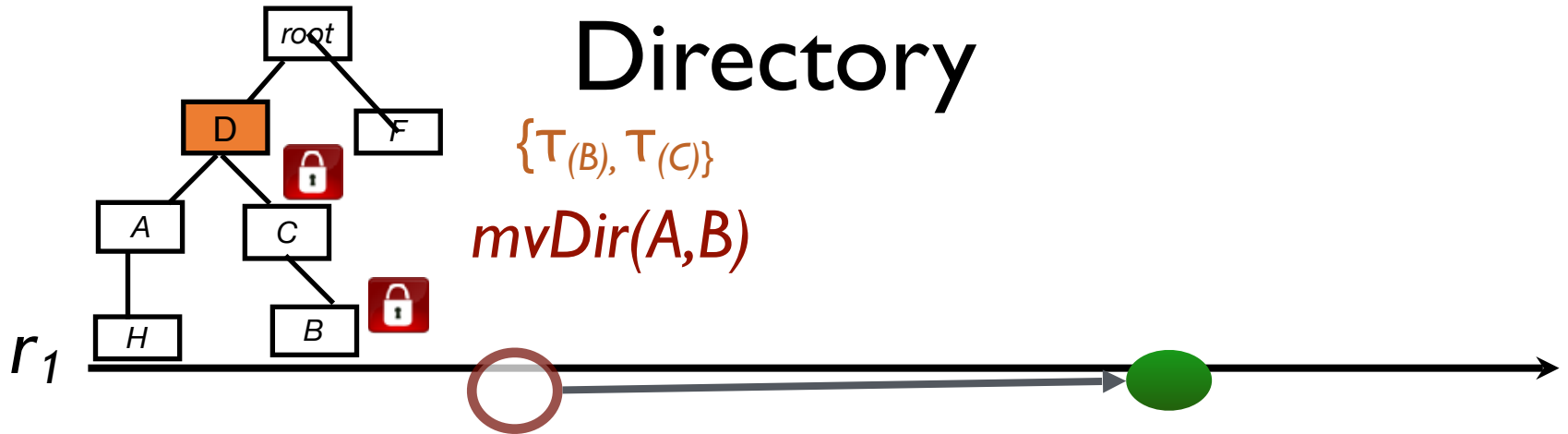
To move B to A: lock path to root

$$\mathbf{T}(A), \{ \mathbf{T}(e) \mid e \in \text{Node } \text{root} \downarrow^+ e \wedge e \downarrow^+ A \}$$

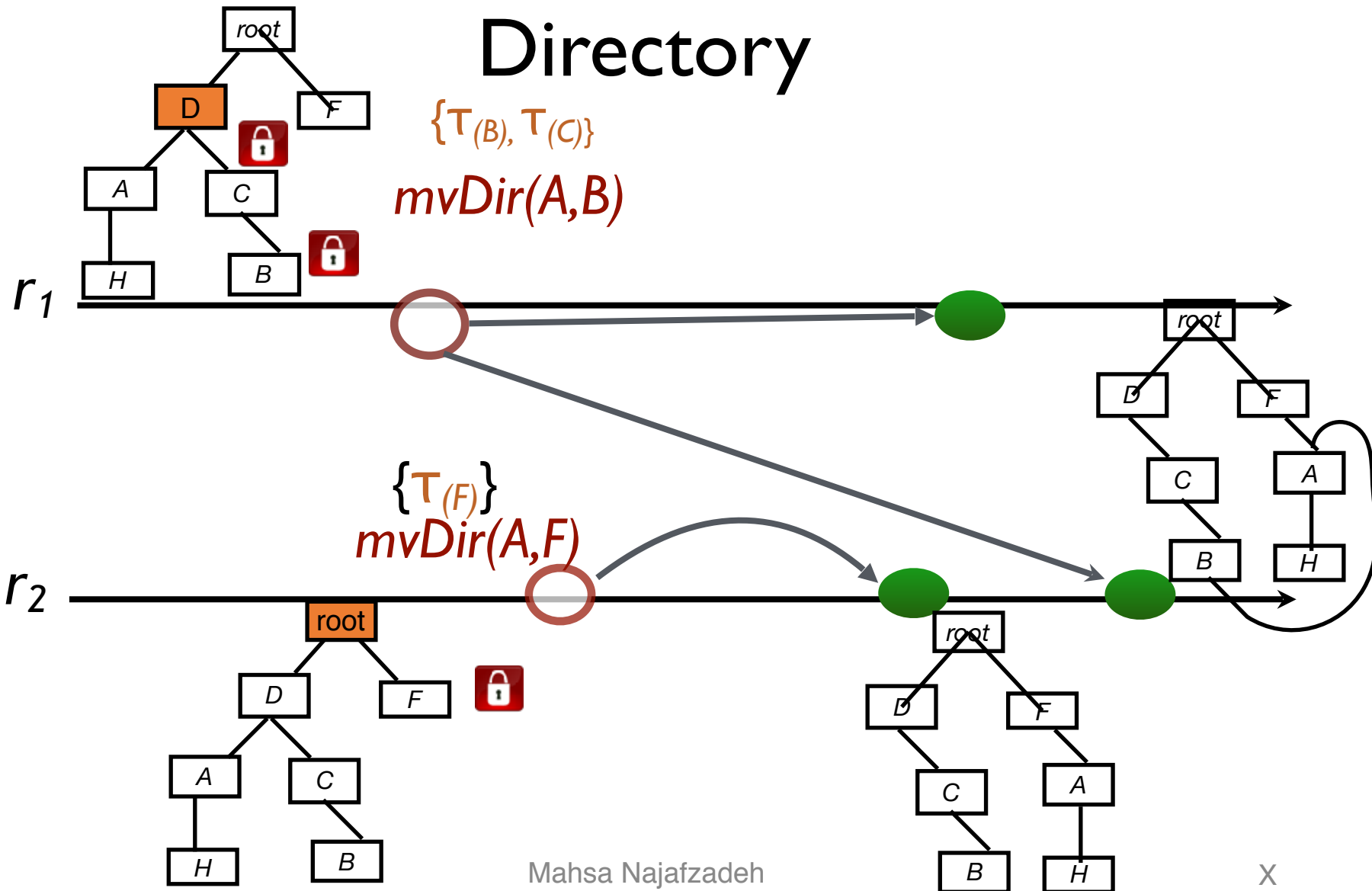
Removing Token Over Source Directory



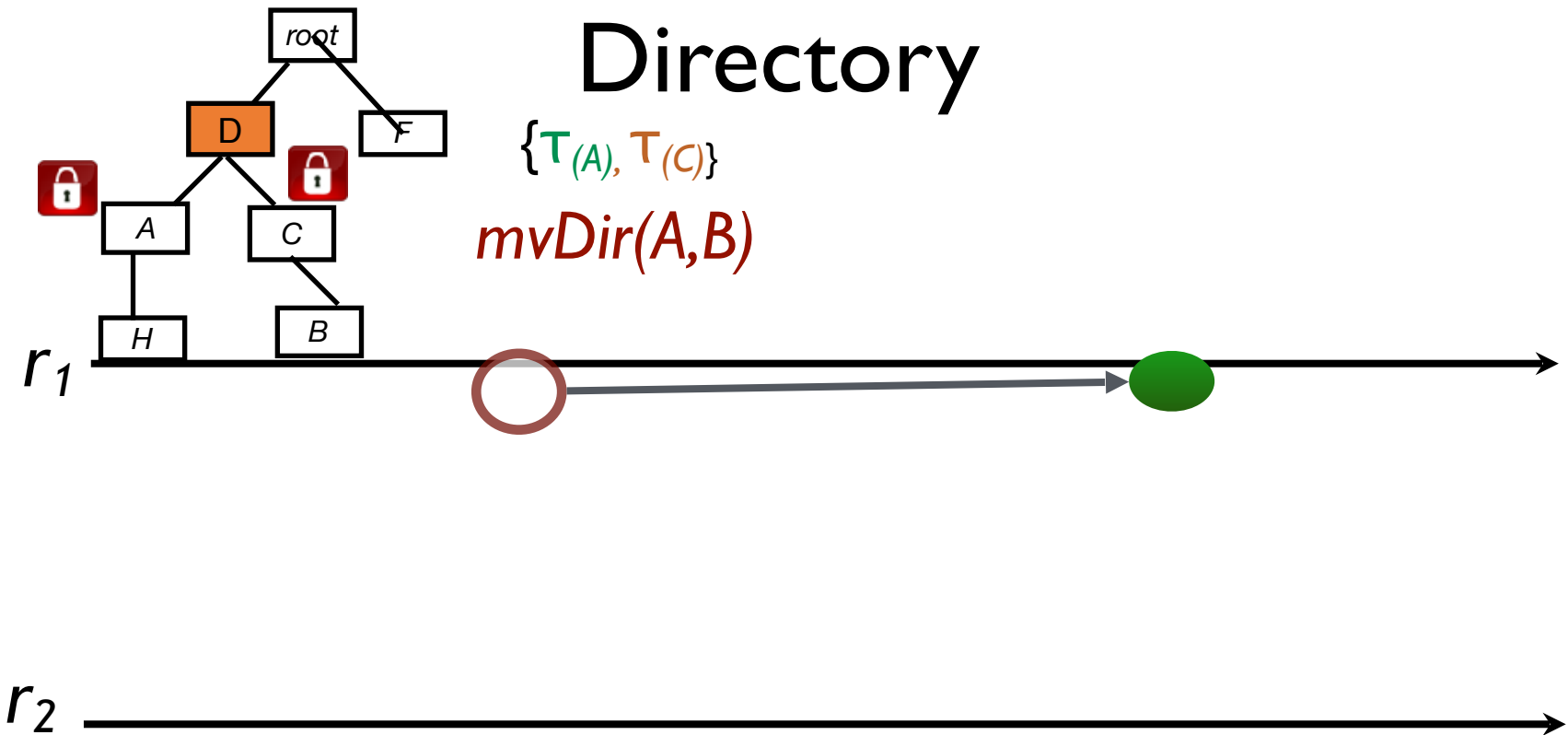
Removing Token Over Source Directory



Removing Token Over Source Directory

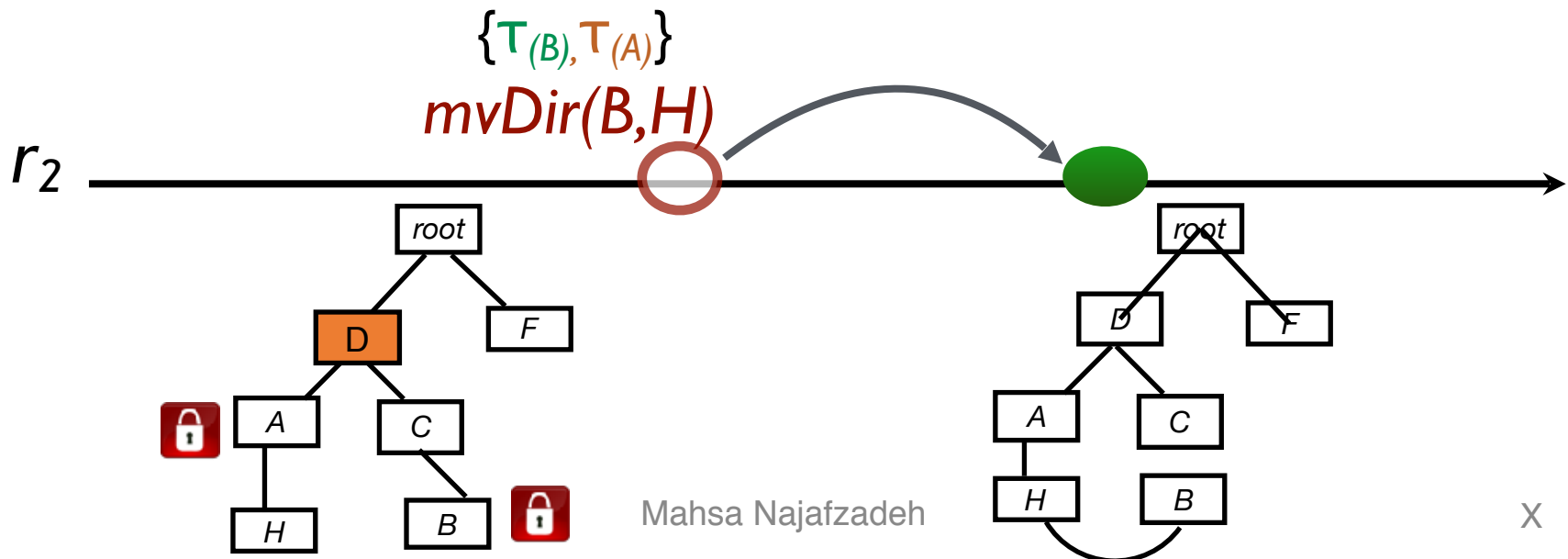
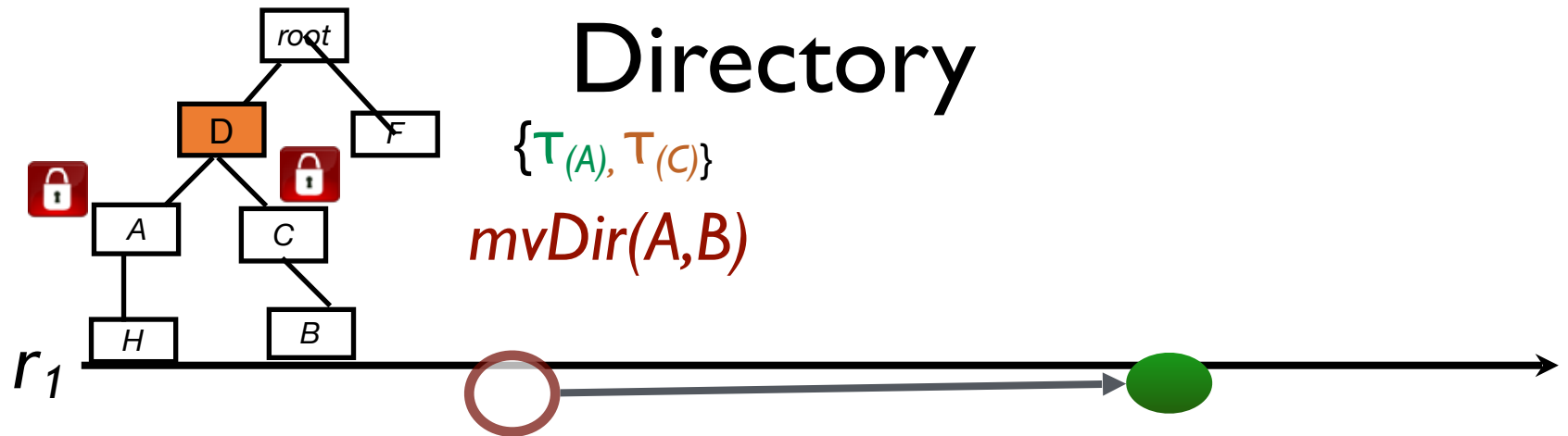


Removing Token Over Destination



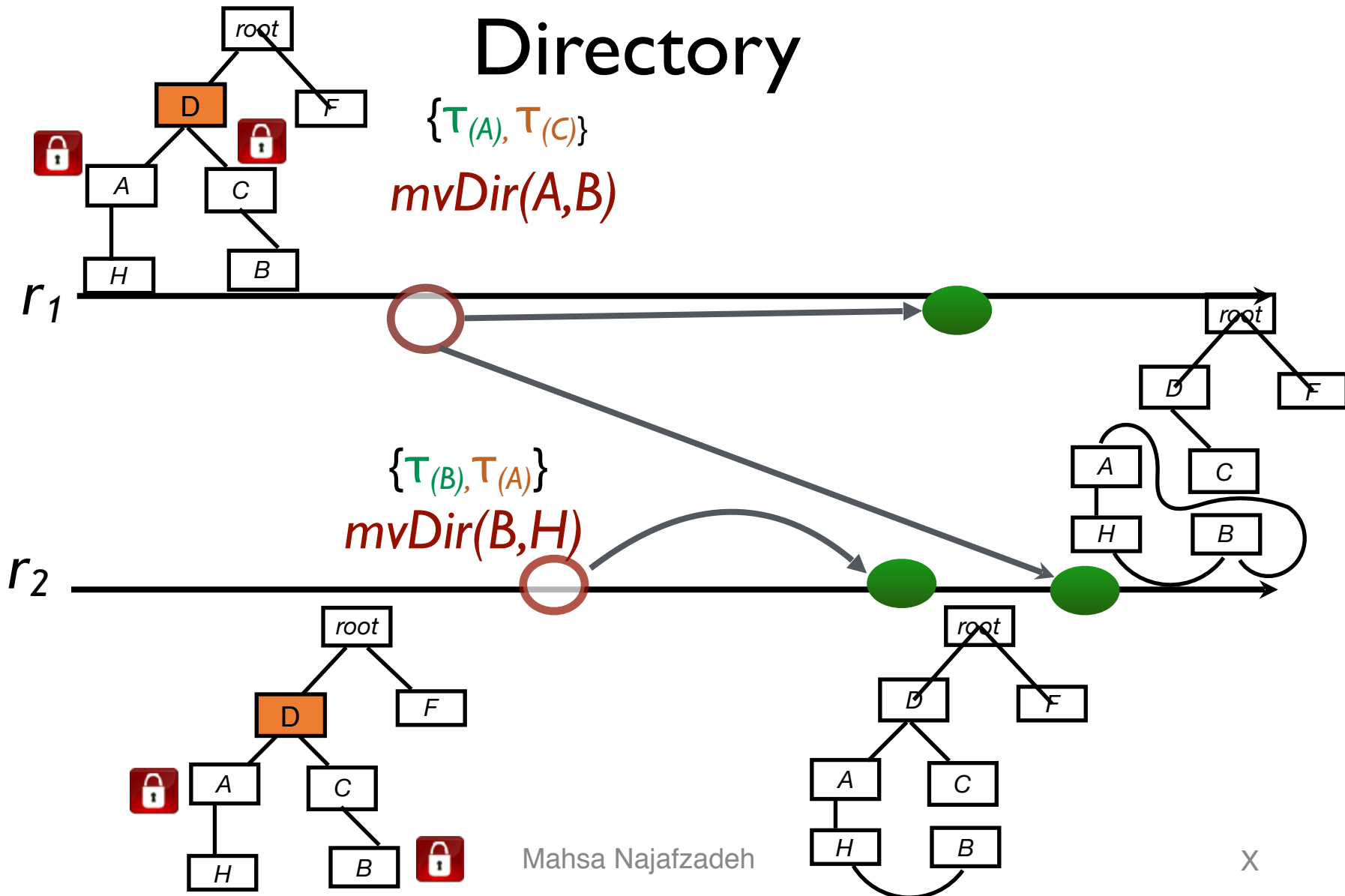
Removing Token Over Destination

Directory

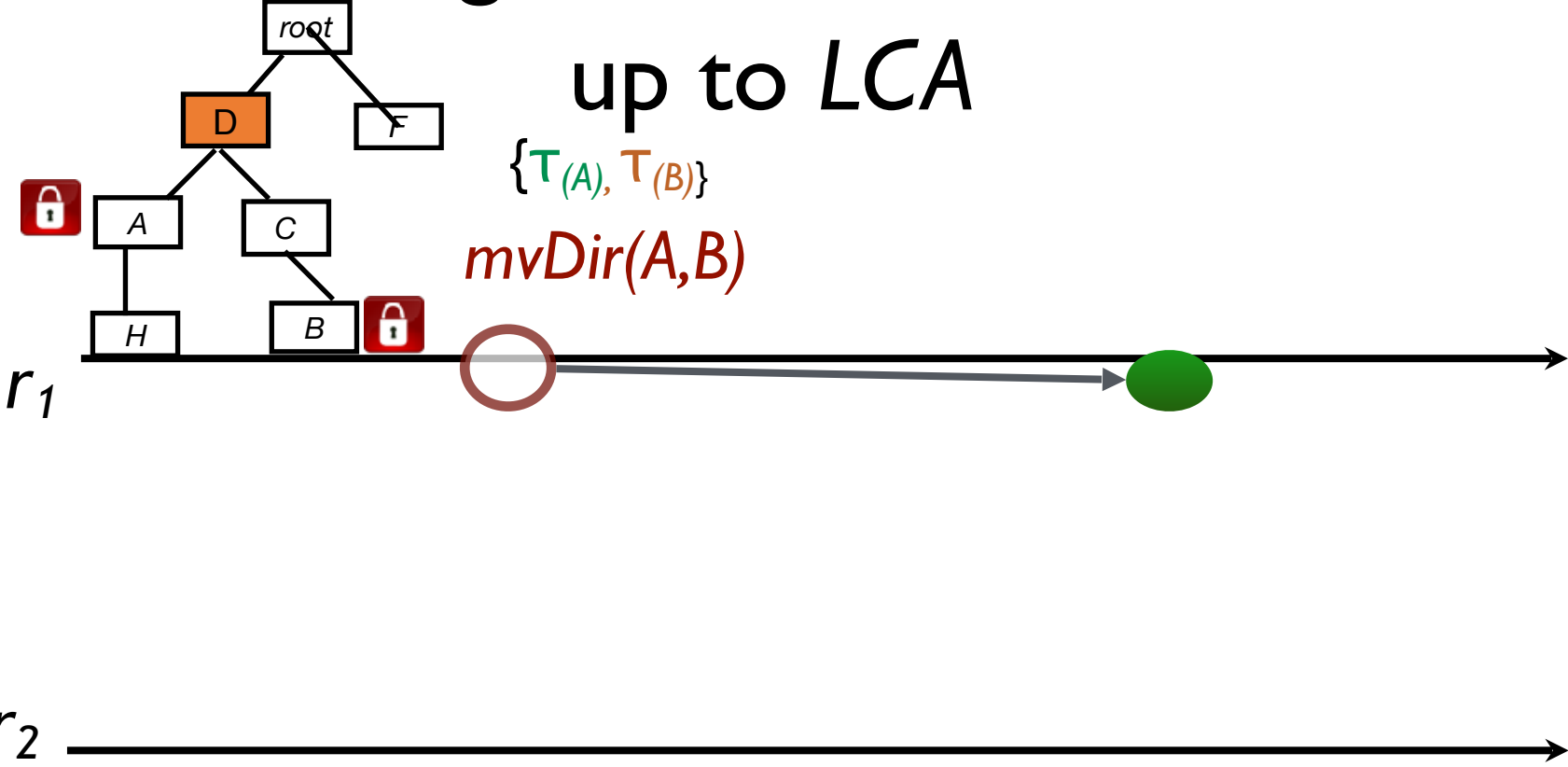


Removing Token Over Destination

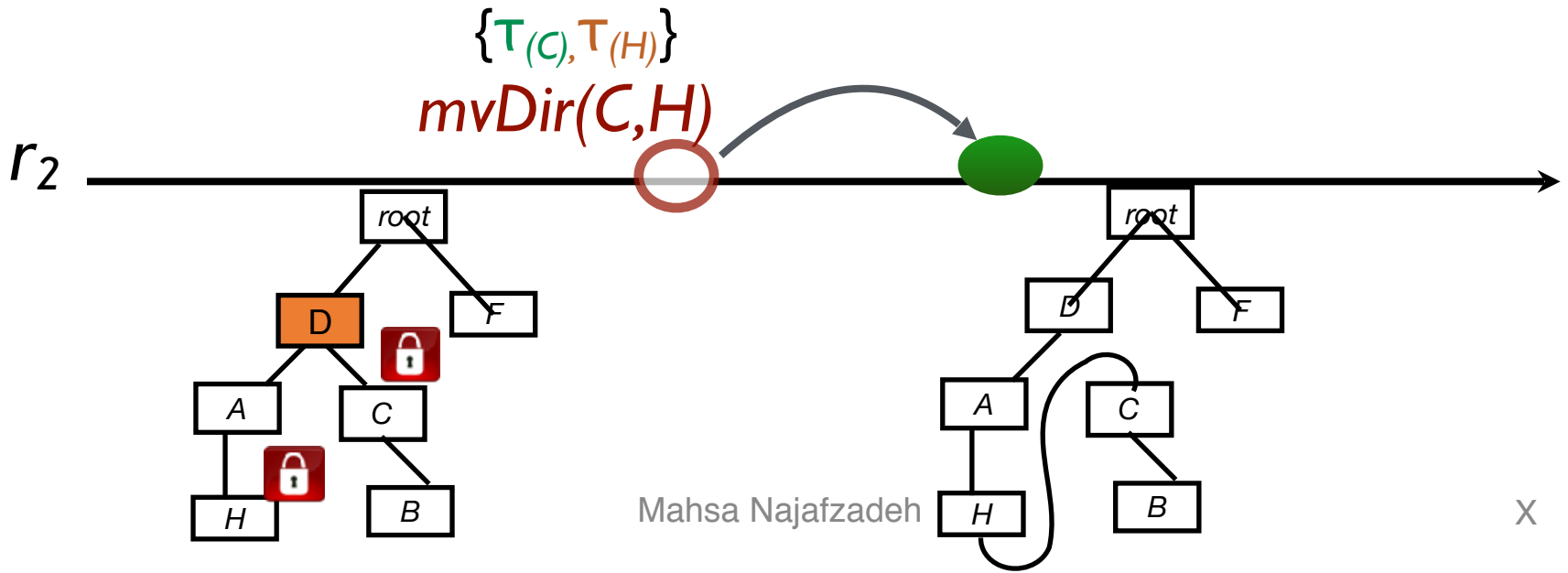
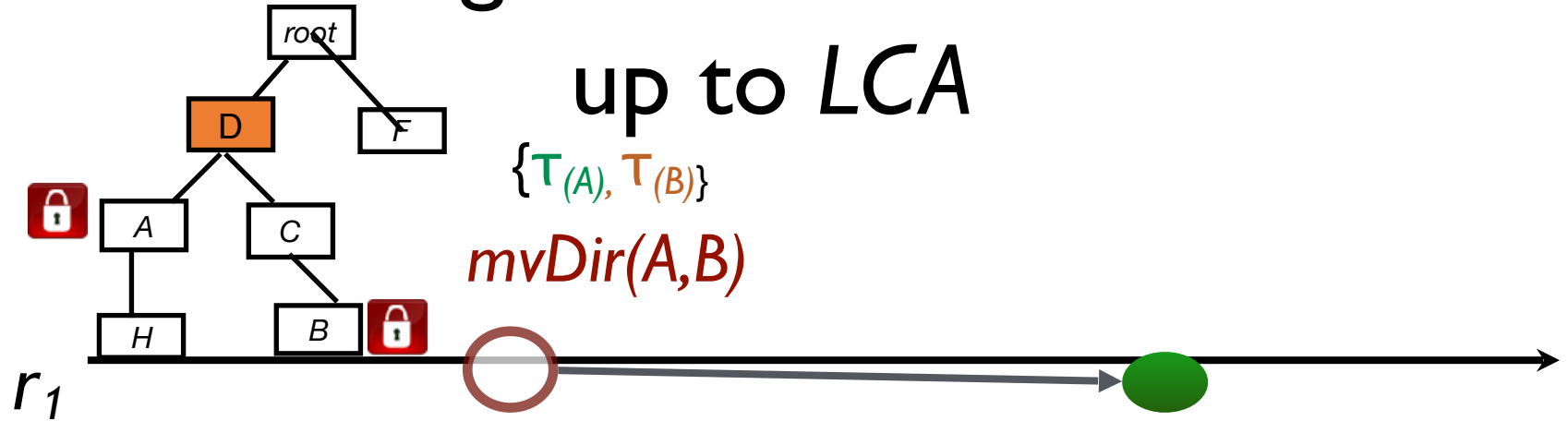
Directory



Removing Token Over Ancestors

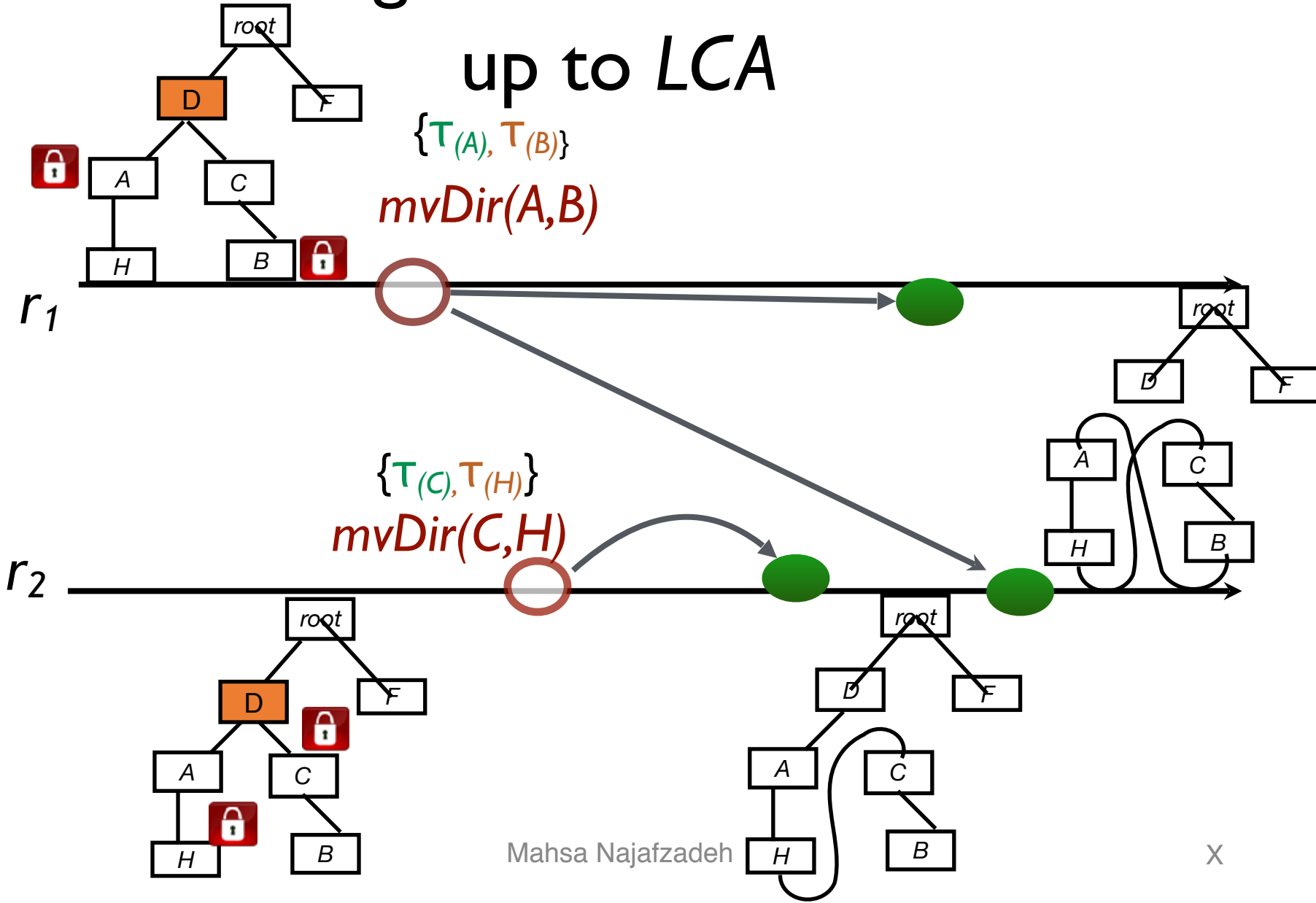


Removing Token Over Ancestors



Removing Token Over Ancestors

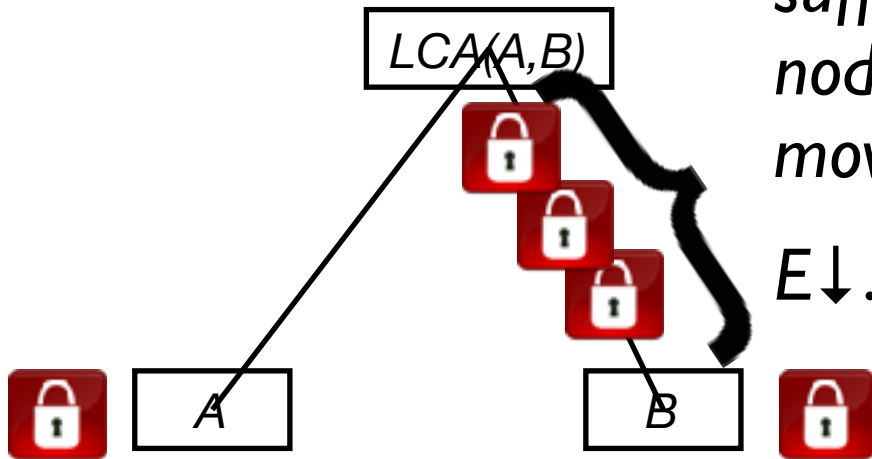
up to *LCA*



Intuition For Move Tokens

Assume that these tokens are not sufficient and we have loop over a node, called E , due to concurrent move operations:

$E \downarrow \dots B \downarrow A \dots \downarrow E$

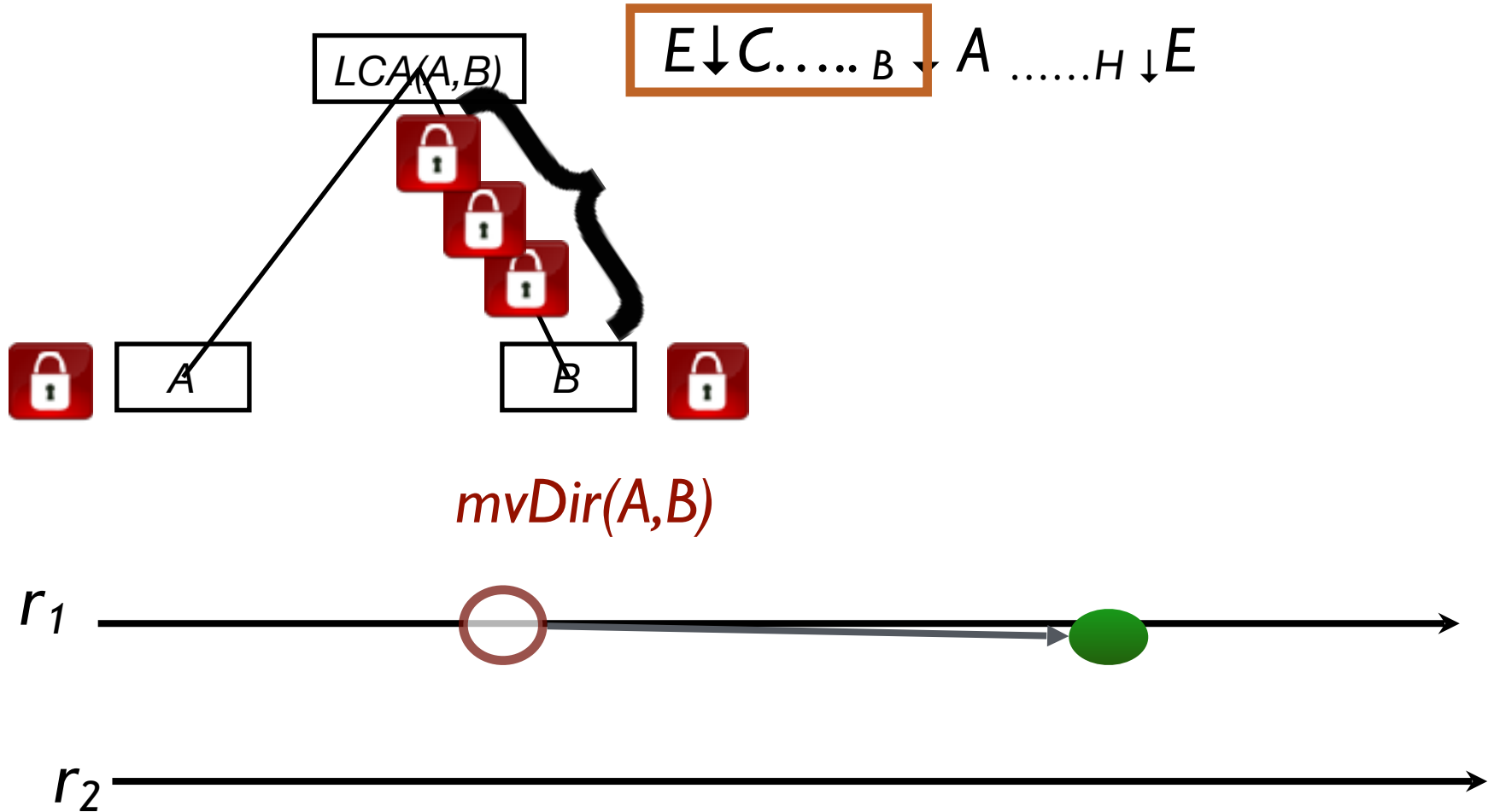


mvDir(A,B)



Intuition For Move Tokens

consider the left side of the loop



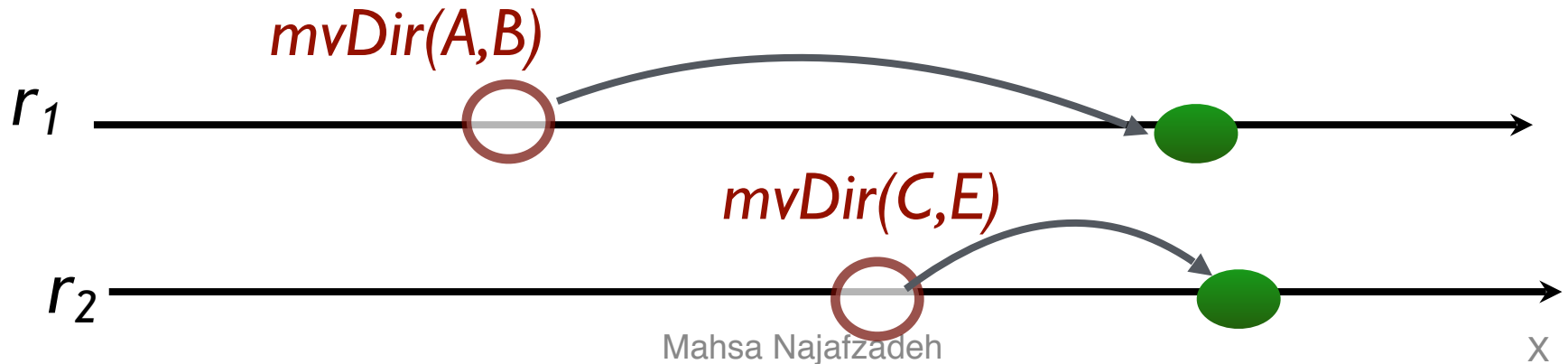
Intuition For Move Tokens

$$E \downarrow C \dots B \downarrow A \dots H \downarrow E$$

The left side implies that one of B's ancestors, called C, concurrently moves to E

mvDir(C,E):

Precondition: Directory E is not a descendent of C



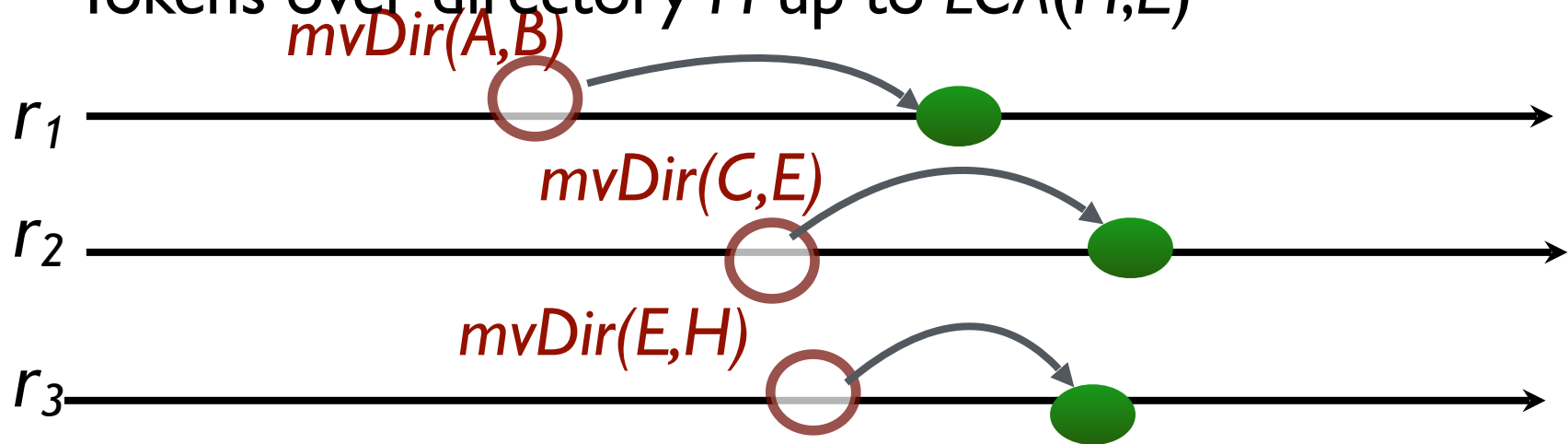
$E \downarrow C \dots B \downarrow A \dots H \downarrow E$

Now, consider the right side of loop

The right side implies that E concurrently moves to one of A's descendants, called H

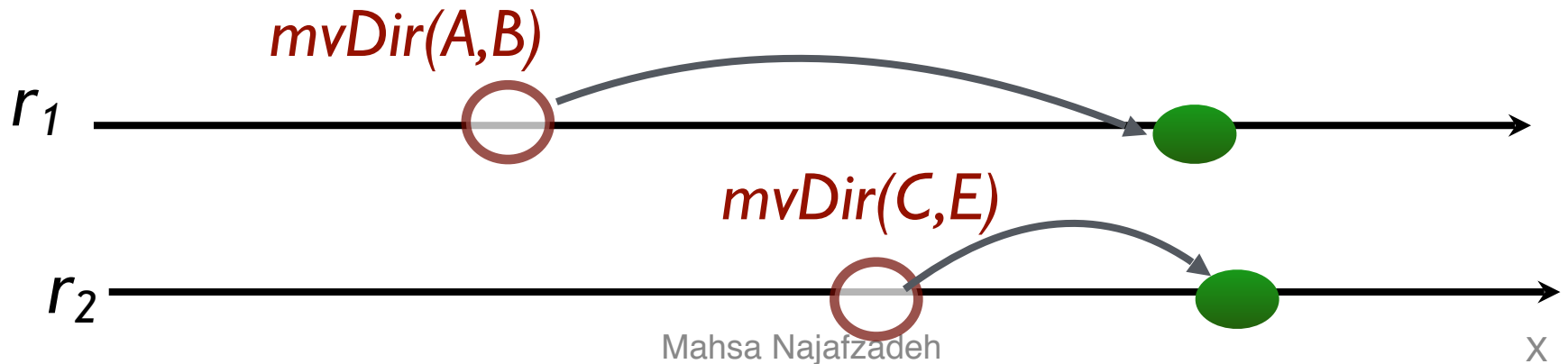
$mvDir(E, H)$

Tokens over directory H up to $LCA(H, E)$

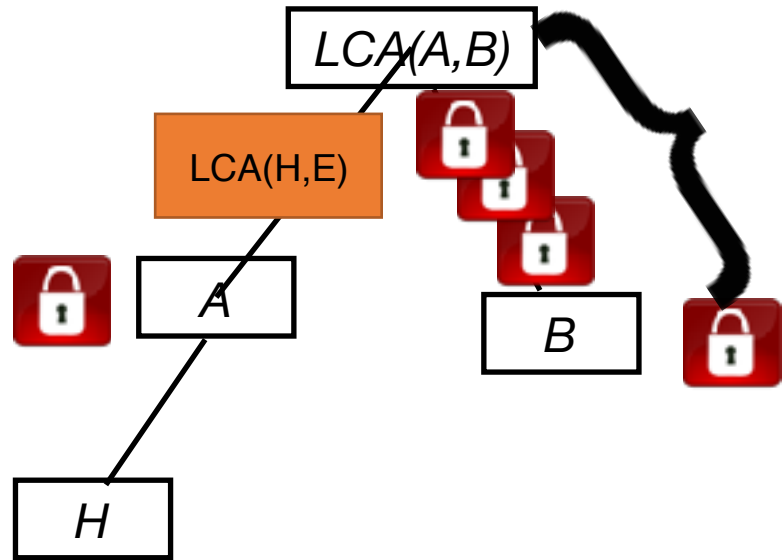


Intuition For Move Tokens

where is $LCA(H,E)$?

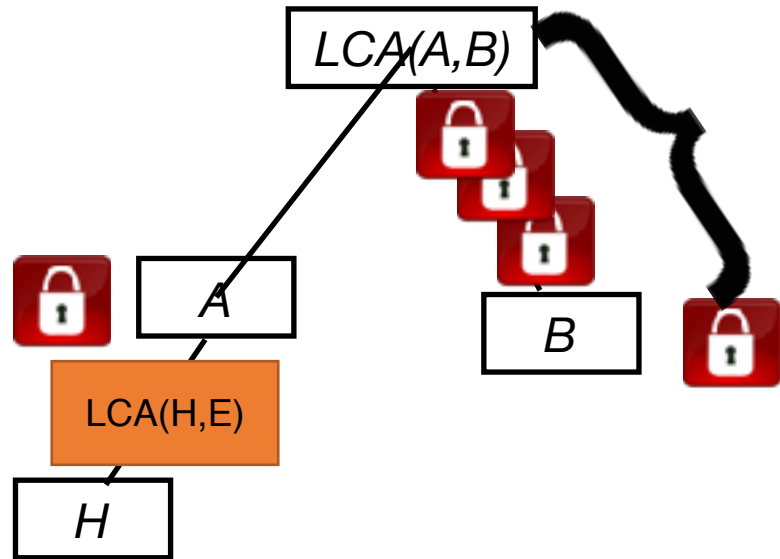


$E \downarrow C \dots B \downarrow A \dots H \downarrow E$



I) $LCA(H,E)$ is located between A and $LCA(A,B)$
in this case moving E to H requires token over A
that conflicts with tokens for moving A to B

$E \downarrow C \dots B \downarrow A \dots H \downarrow E$



2) $LCA(H,E)$ is located under A:

E is concurrently moved under A which is not possible because this move operation needs to acquire tokens conflicting with $mvDir(A,B)$

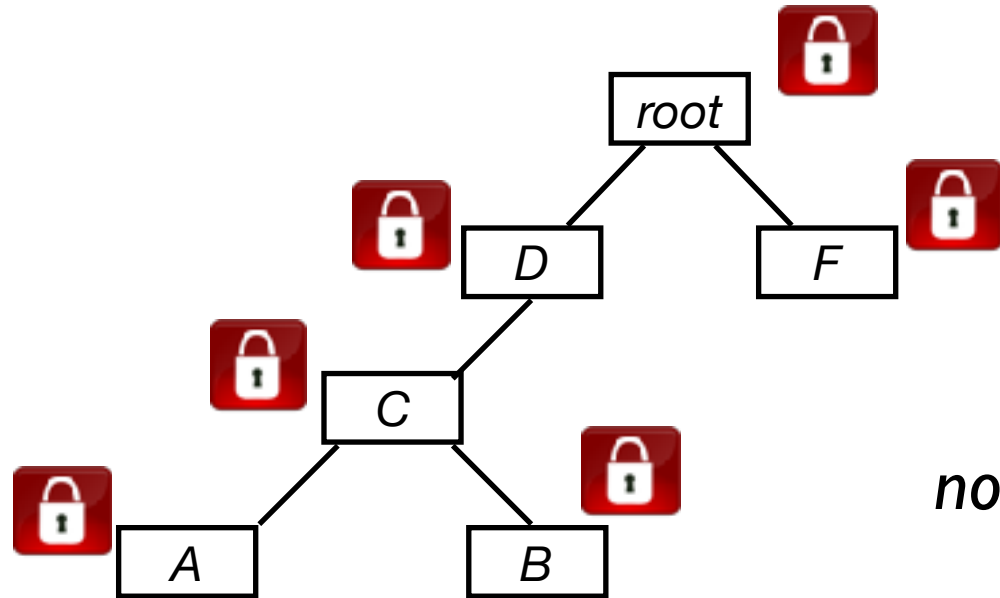
CISE Proof Tool's Result

Semantics	#OP	#Token	#Invariant	#Violation	Time (ms)
Sequential	7	7	1	0	1297
Fully Async	7	0	1	1	2350
Mostly Async	7	2	1	0	1570

Future Work

- Implement the file system semantics
And compare their actual performance under real workloads
- Reason about the operation executions in the presence of failure

Root Lock

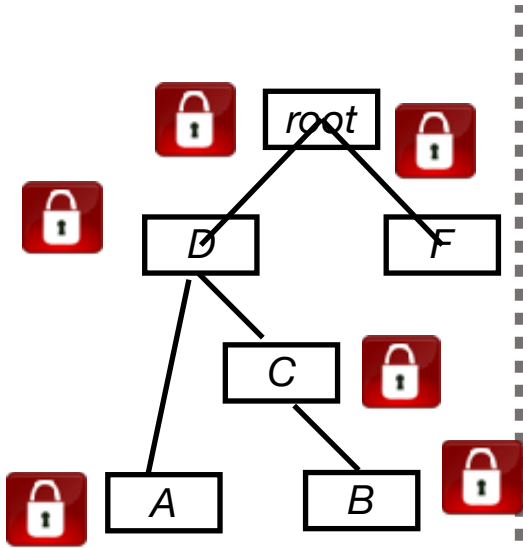


no concurrent moves

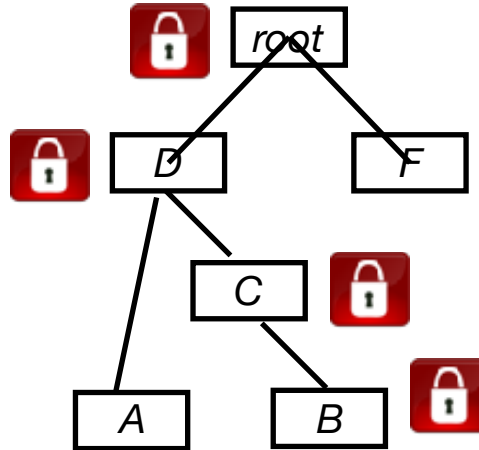
To move A to B: lock whole tree

$$\{ \mathcal{T}(e) \mid e \in \text{Node}, (\mathcal{T}(e) \bowtie \mathcal{T}_{d(e)}) \}$$

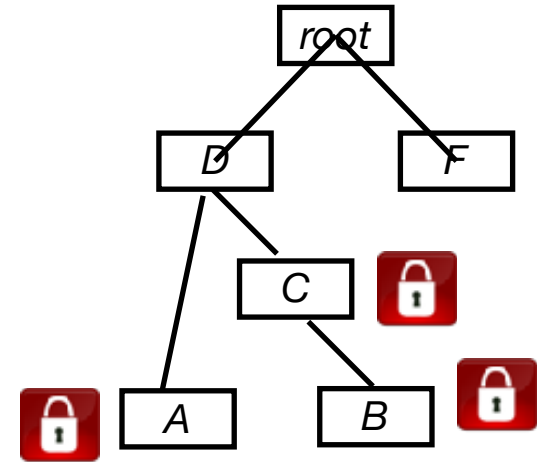
Move tokens



lock whole tree



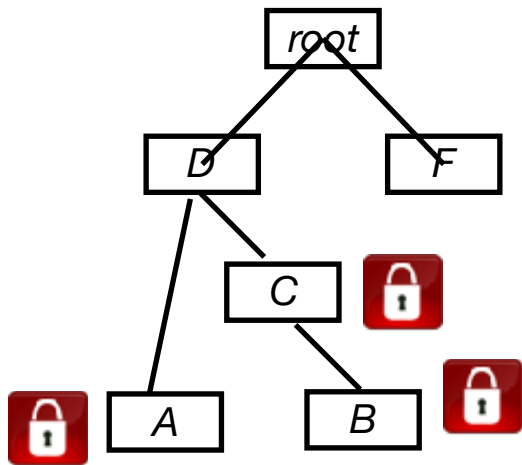
lock path to root



lock until LCA(B,A)

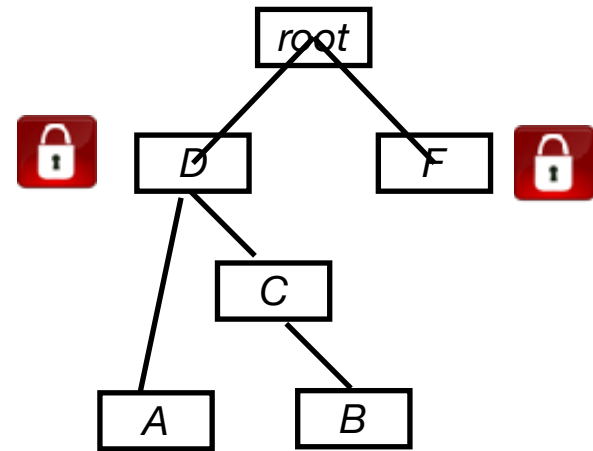
no concurrent moves

Concurrent Moves



To move A to B:

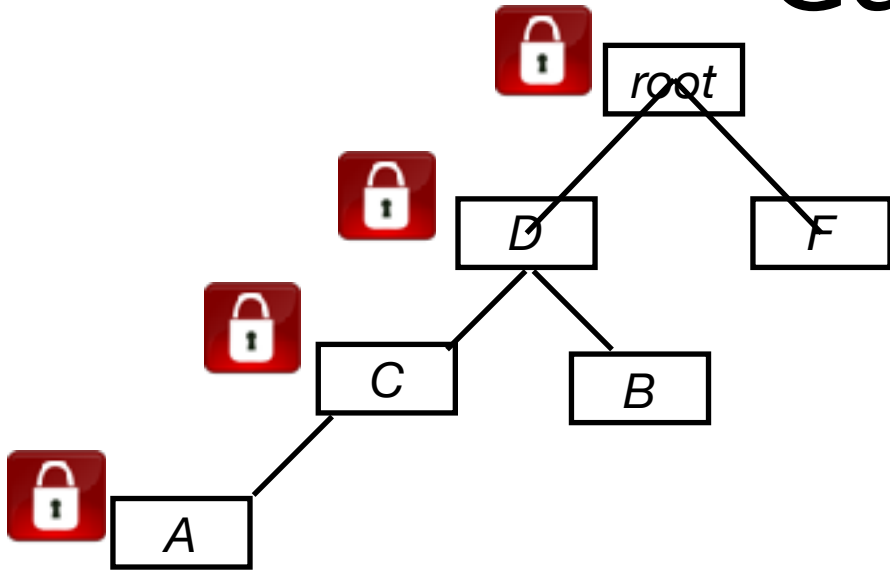
$$\tau_{s(A)}, \tau_{d(B)}, \tau_{d(C)}$$



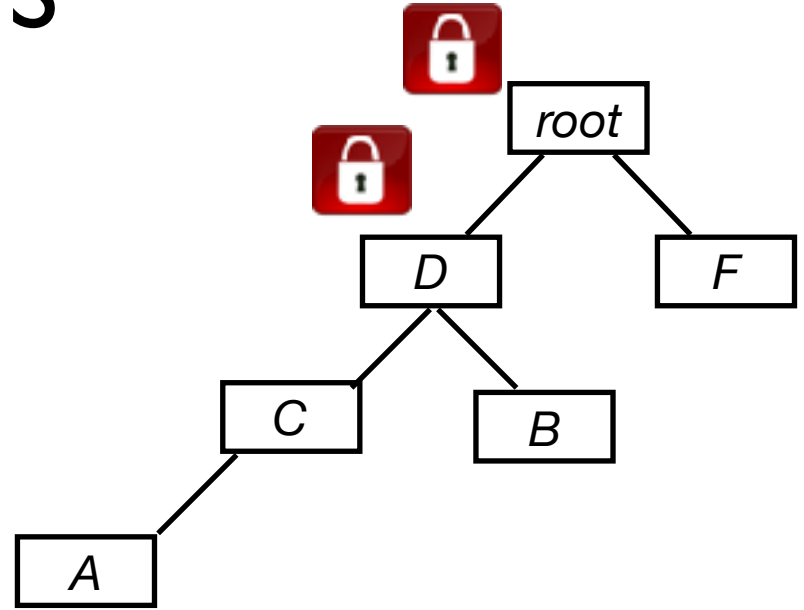
To move F to D:

$$\tau_{s(F)}, \tau_{d(D)}$$

GeoFS



To move B to A



To move F to D

lock path to root

Thesis Contributions

1. Static analysis tool for proving integrity invariants of applications
2. A case study of the application of our analysis tool for designing an efficient file system semantics
3. A set of useful invariant patterns + protocols

Efficiently Implementable Patterns of Invariants

- Some interesting classes of invariants

Relating consistency to invariants

- Which primitives guarantee which invariants