Self-Stabilization in Tree-Structured P2P Service Discovery Systems

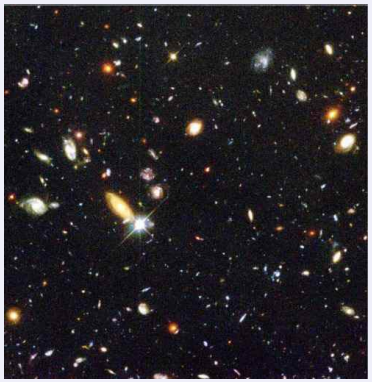
Cédric Tedeschi[1]
Eddy Caron[2], Ajoy K. Datta[3], and Franck Petit[4]

(1) INRIA Sophia Antipolis Méditerranée
(2) LIP Lab., University of Lyon — CNRS — ENS Lyon — UCB Lyon — INRIA
(3) University of Nevada Las Vegas
(4) MIS Lab., University of Picardie Jules Verne

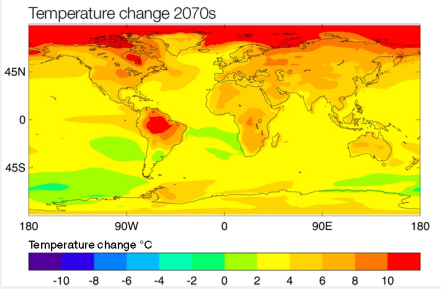
Workshop APRETAF
Jan. 22nd

Context

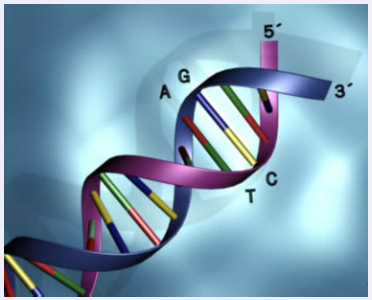
Computing Needs



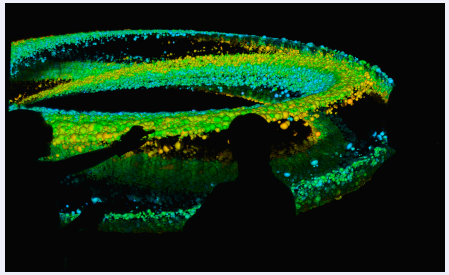
Cosmology



Climate prediction



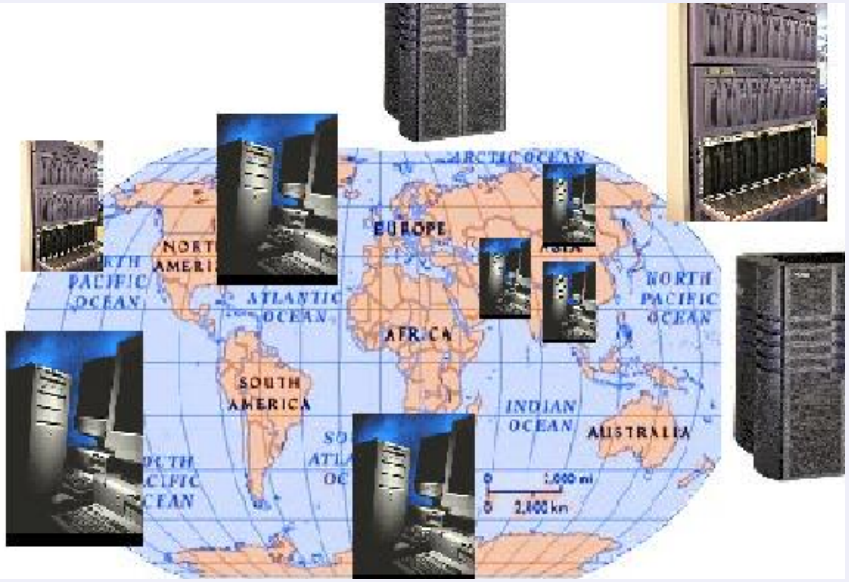
Genomics



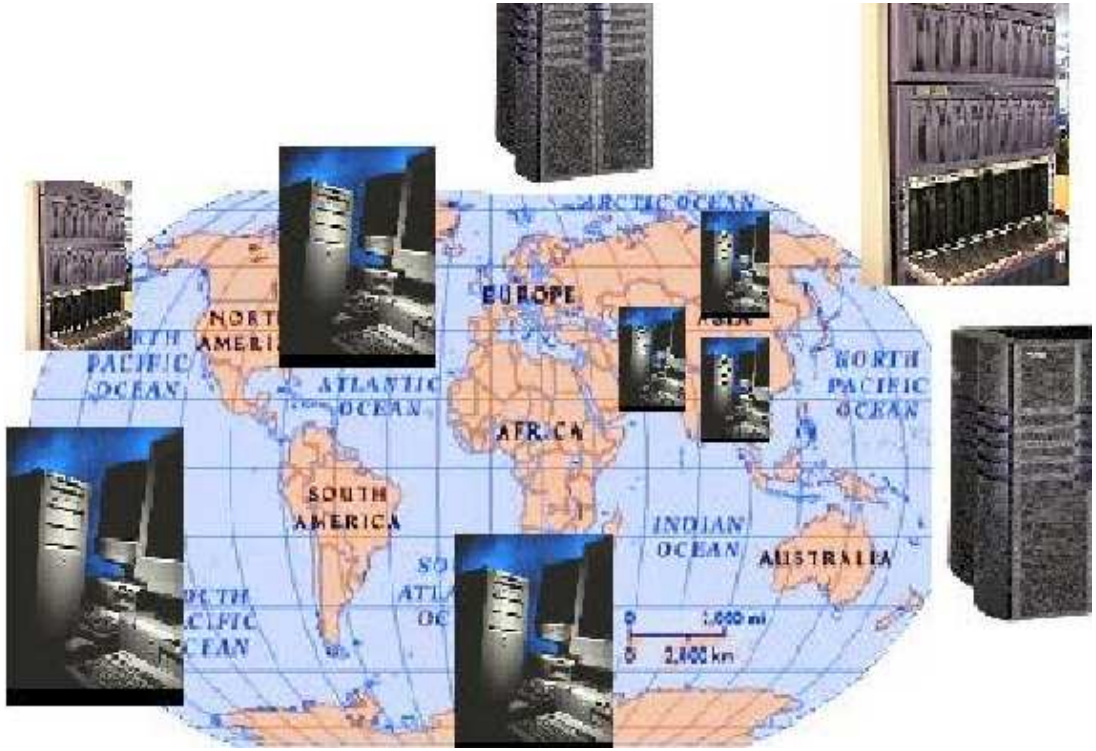
Nuclear security

Computing Power

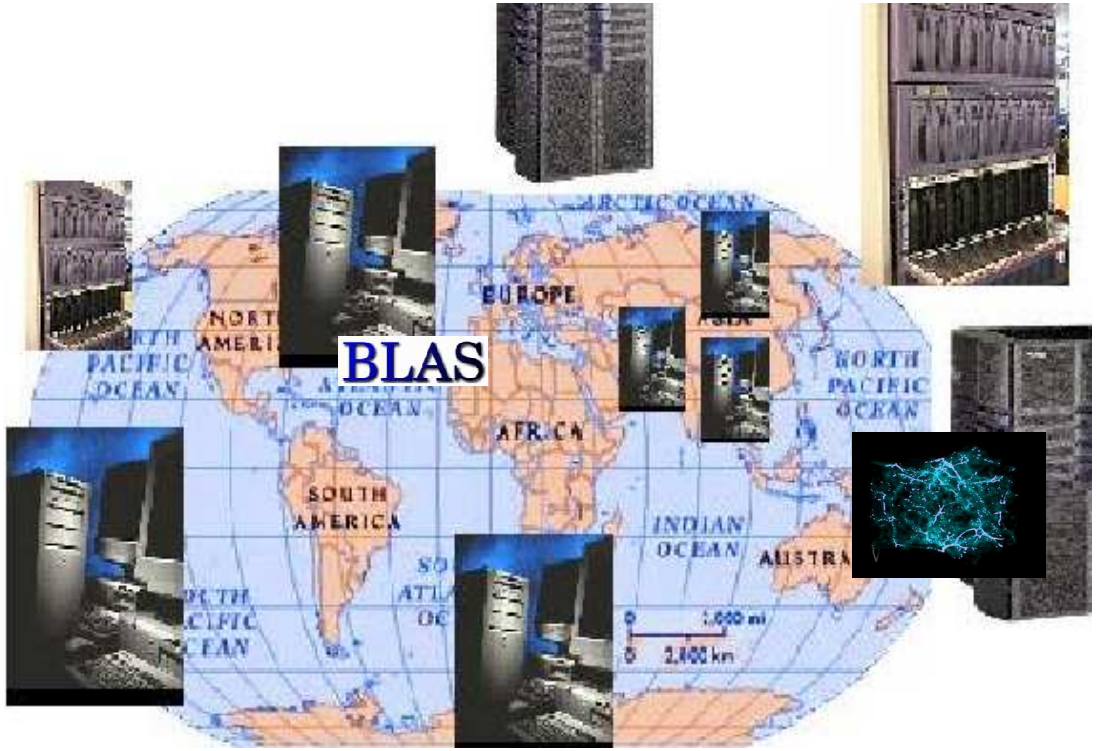
Computational Grids



Context

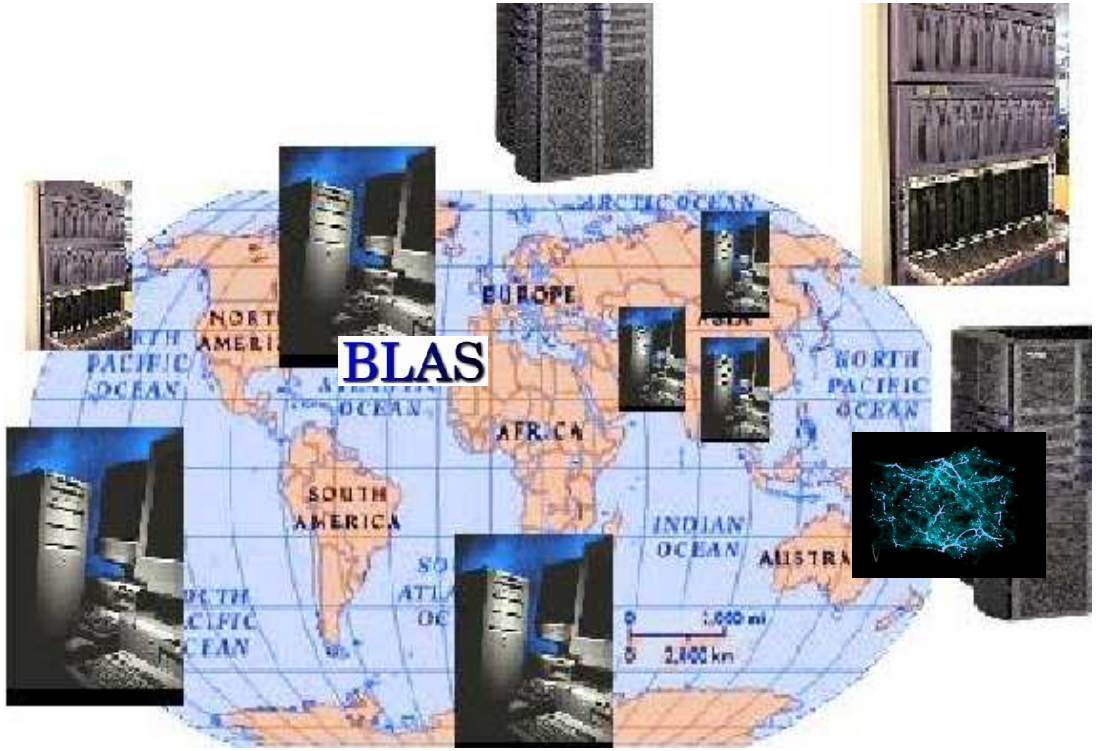

ARCTIC OCEAN
EUROPE
NORTH PACIFIC OCEAN
ATLANTIC OCEAN
AFRICA
SOUTH AMERICA
INDIAN OCEAN
AUSTRALIA

Context

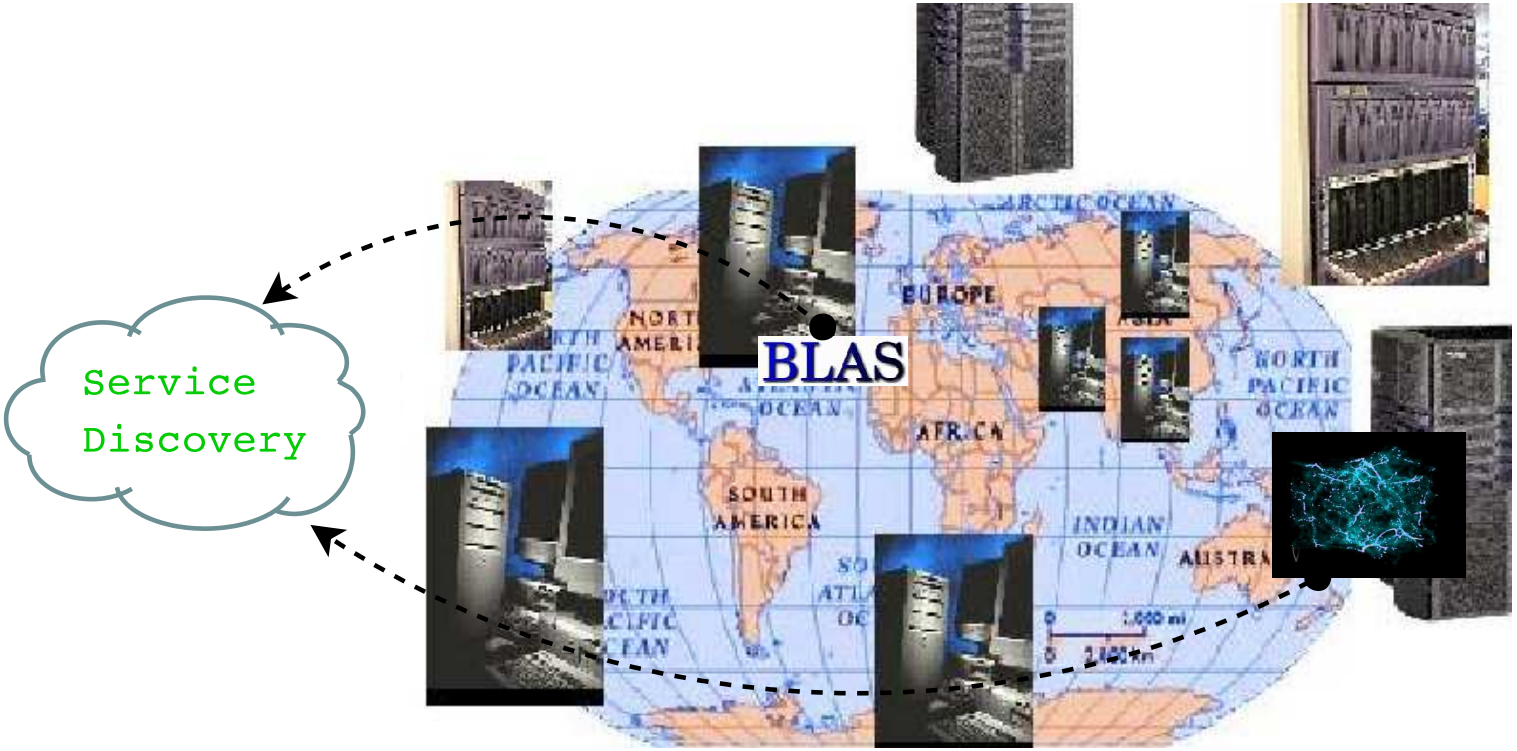

BLAS
EUROPE
AFRICA
SOUTH AMERICA
INDIAN OCEAN
NORTH PACIFIC OCEAN

Context

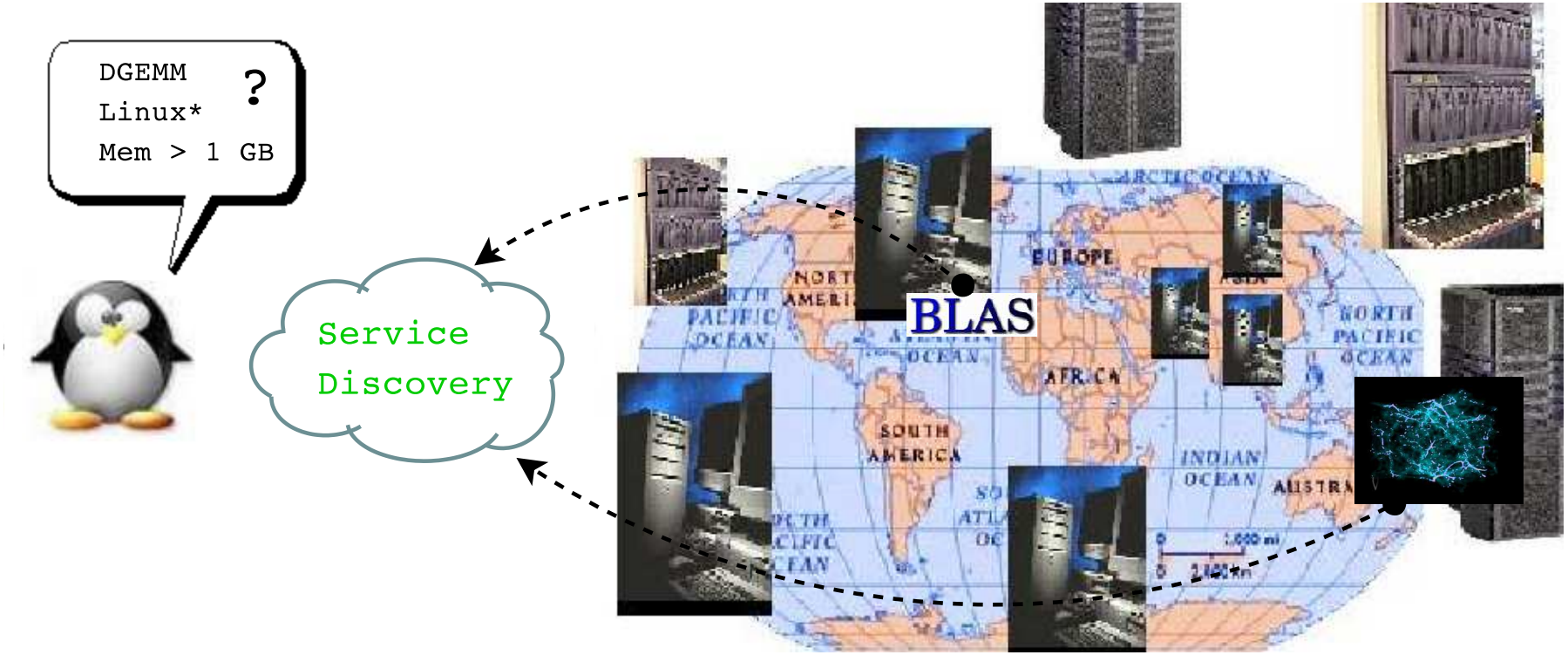
Service Discovery



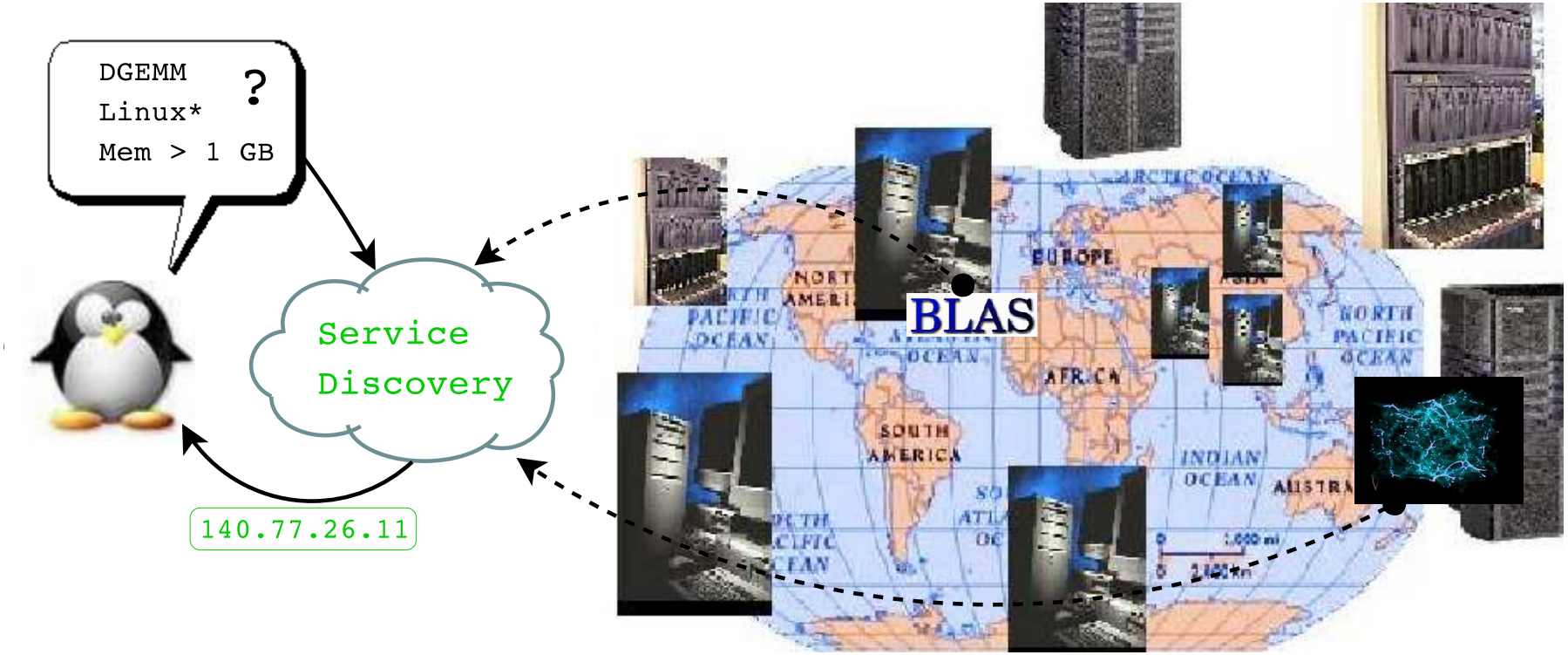
Context


Service
Discovery
BLAS
EUROPE
AFRICA
SOUTH
AMERICA
INDIAN
OCEAN
NORTH
PACIFIC
OCEAN

Context

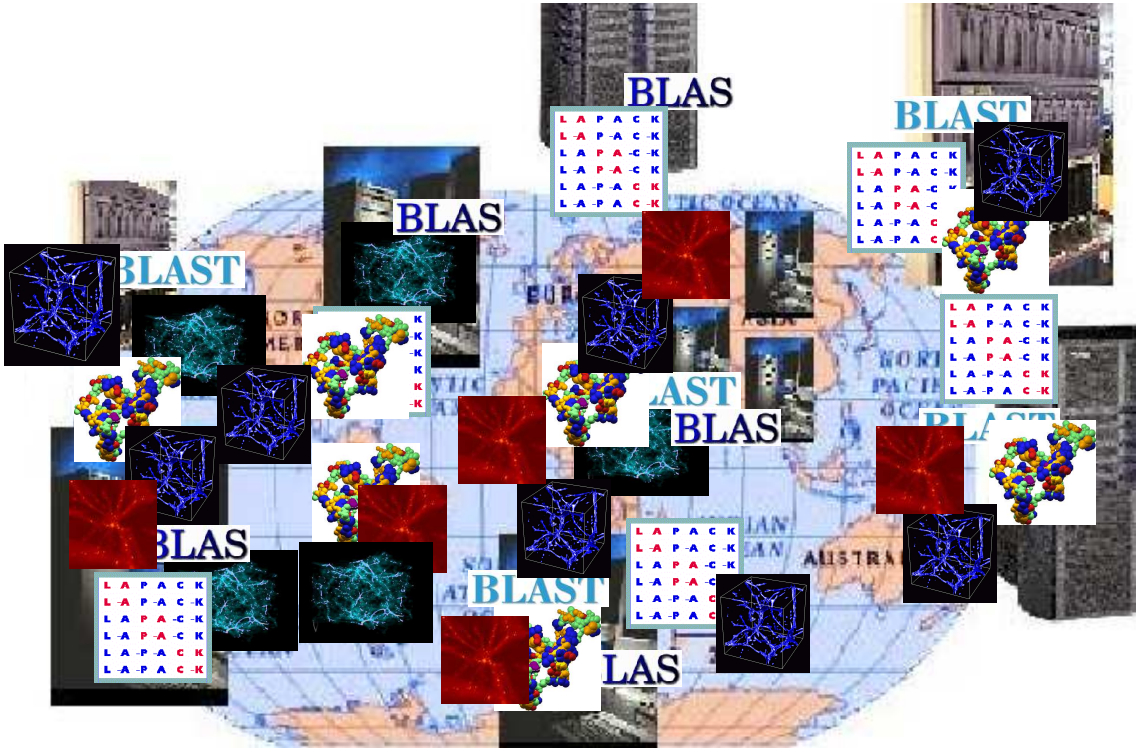

DGEMM
Linux*
Mem > 1 GB
?
Service
Discovery
BLAS
EUROPE
AFRICA
SOUTH
AMERICA
INDIAN
OCEAN
NORTH
PACIFIC
OCEAN

Context

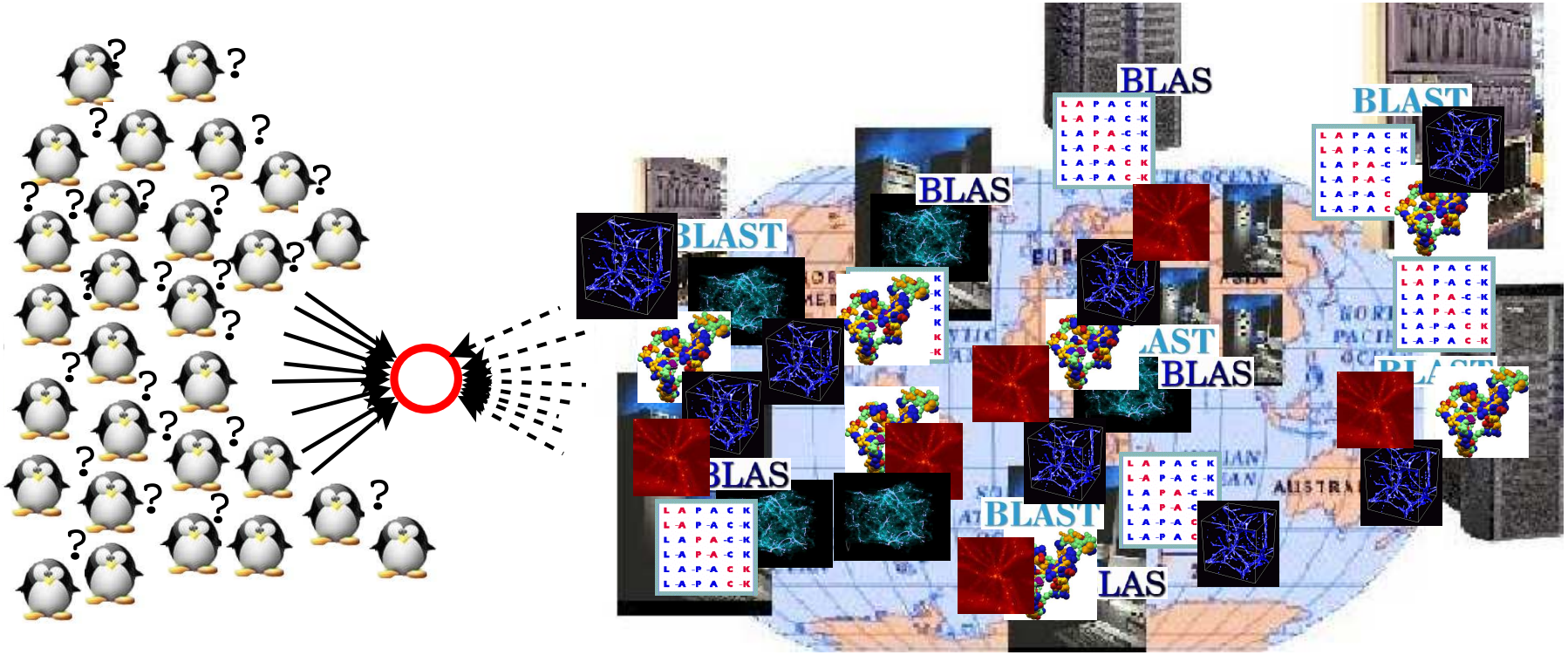

DGEMM
Linux*
Mem > 1 GB
?
Service Discovery
140.77.26.11
BLAS
EUROPE
AFRICA
SOUTH AMERICA
INDIAN OCEAN
NORTH PACIFIC OCEAN

Context

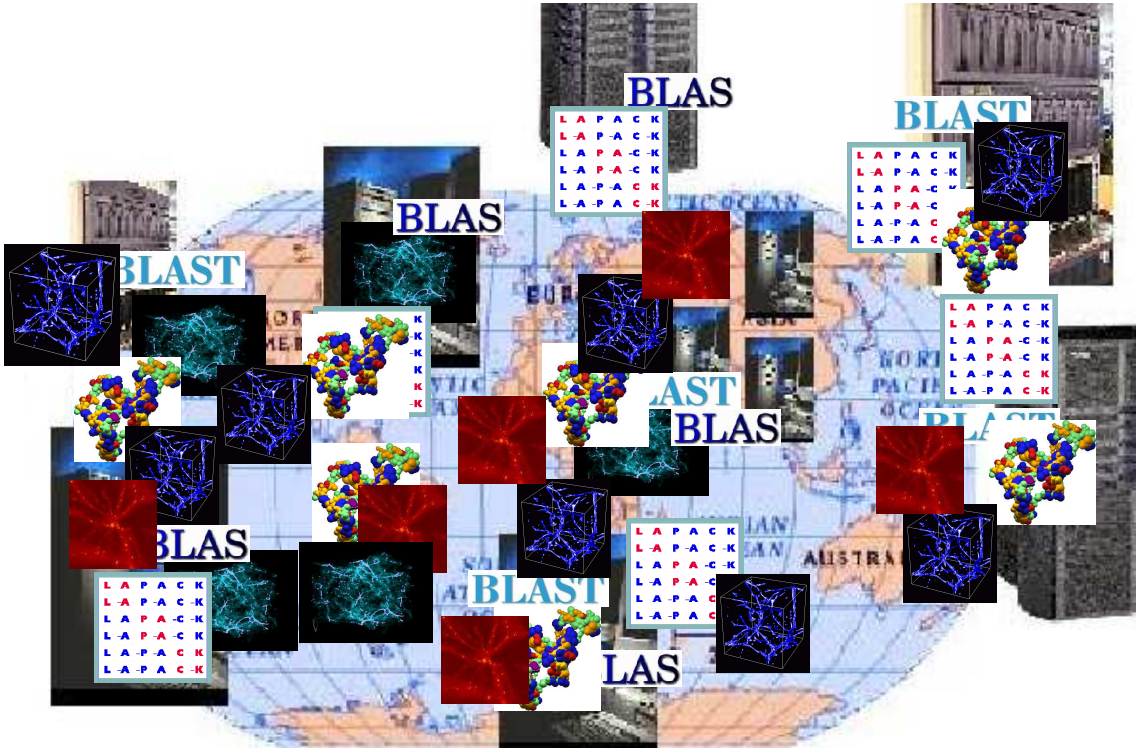

Service
Discovery


BLAS
BLAST
BLAS
BLAST
BLAS
BLAS
BLAST
LAS

Context



Context



Context

- Service discovery in GRID Computing
  - Services (binary file, library) installed on servers
  - Servers declare their services, client discovers them
- Target platforms: **Peer-to-Peer Platform**
  - Decentralized algorithms (no central infrastructure)
  - Distributed data structure for service retrieval
  - Large scale systems
  - Dynamic (joins and leaves of nodes)
  - Fault-tolerance

Context

- Service discovery in GRID Computing
  - Services (binary file, library) installed on servers
  - Servers declare their services, client discovers them
- Target platforms: **Peer-to-Peer Platform**
  - Decentralized algorithms (no central infrastructure)
  - Distributed data structure for service retrieval
  - Large scale systems
  - Dynamic (joins and leaves of nodes)
  - Fault-tolerance

Context

- Service discovery in GRID Computing
  - Services (binary file, library) installed on servers
  - Servers declare their services, client discovers them
- Target platforms: **Peer-to-Peer Platform**
  - Decentralized algorithms (no central infrastructure)
  - Distributed data structure for service retrieval
  - Large scale systems
  - Dynamic (joins and leaves of nodes)
  - Fault-tolerance

Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding:
2. **Structured**
   - Distributed Hashing Table (DHT)
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - Tree (Range Search)

Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding: Costly and Partial Search
2. **Structured**
   - Distributed Hashing Table (DHT)
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - Tries (or Prefix Trees)

Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding: Costly and Partial Search
2. **Structured**
   - Distributed Hashing Table (DHT)
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or **Prefix Trees**)

Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding: Costly and Partial Search
2. **Structured**
   - Distributed Hashing Table (DHT) Exact Queries Only
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or **Prefix Trees**)

Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding: Costly and Partial Search
2. **Structured**
   - Distributed Hashing Table (DHT) Exact Queries Only
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or **Prefix Trees**)

Trie-based Overlays

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Related Works
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)
  - **Fault-tolerance**: either ignored or based on replication
    - Replication: Costly. What can be done if k is reached?
    - Does not recover after arbitrary failures (*e.g.*, memory corruption)

Trie-based Overlays

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Related Works
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)
  - **Fault-tolerance**: either ignored or based on replication
    - Replication: Costly. What can be done if k is reached?
    - Does not recover after arbitrary failures (*e.g.*, memory corruption)

Trie-based Overlays

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Related Works
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)
  - **Fault-tolerance**: either ignored or based on replication
    - Replication: Costly. What can be done if k is reached?
    - Does not recover after arbitrary failures (*e.g.*, memory corruption)

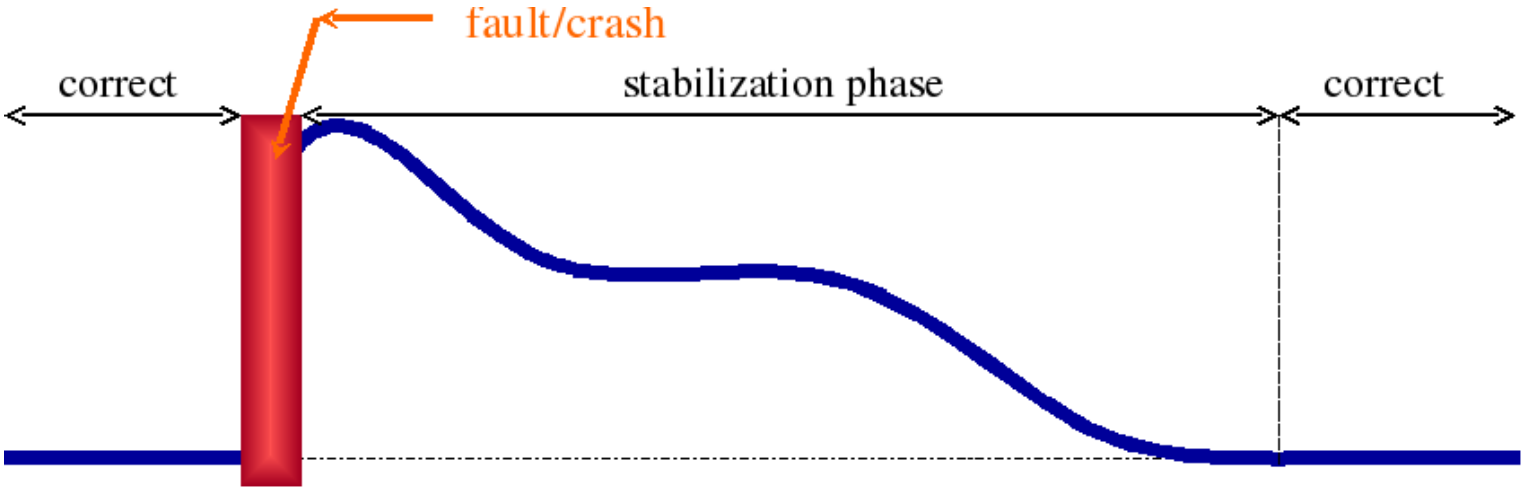
Trie-based Overlays

- Advantages
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries
- Related Works
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)
  - **Fault-tolerance**: either ignored or based on replication
    - Replication: Costly. What can be done if k is reached?
    - Does not recover after arbitrary failures (*e.g.*, memory corruption)

Best-Effort $\rightarrow$ **Self-Stabilization**

Self-Stabilization

- General technique to tolerate transient faults
- Guaranteed to converge to the intended behavior



Self-Stabilizing Overlays for Peer-to-Peer Networks

- Self-Stabilizing Tree (Herault *et al.* – 2007)
- Self-Stabilizing Hypertree (Dolev and Kat – 2007)
- Snap-Stabilizing Tries (Caron *et al.* – 2006)
  - Assumes a rooted connected tree
  - Written in a coarse grain communication model

Self-Stabilizing Overlays for Peer-to-Peer Networks

- Self-Stabilizing Tree (Herault *et al.* – 2007)
- Self-Stabilizing Hypertree (Dolev and Kat – 2007)
- Snap-Stabilizing Tries (Caron *et al.* – 2006)
  - Assumes a rooted connected tree
  - Written in a coarse grain communication model

Contributions

Self-Stabilizing Trie Maintenance

- Built in the *P2P* model
- Message-Passing
- Arbitrary initial topology
- Comprehensive proof of stabilization
- Simulation results

Outline

Outline

P2P Network

Physical Network

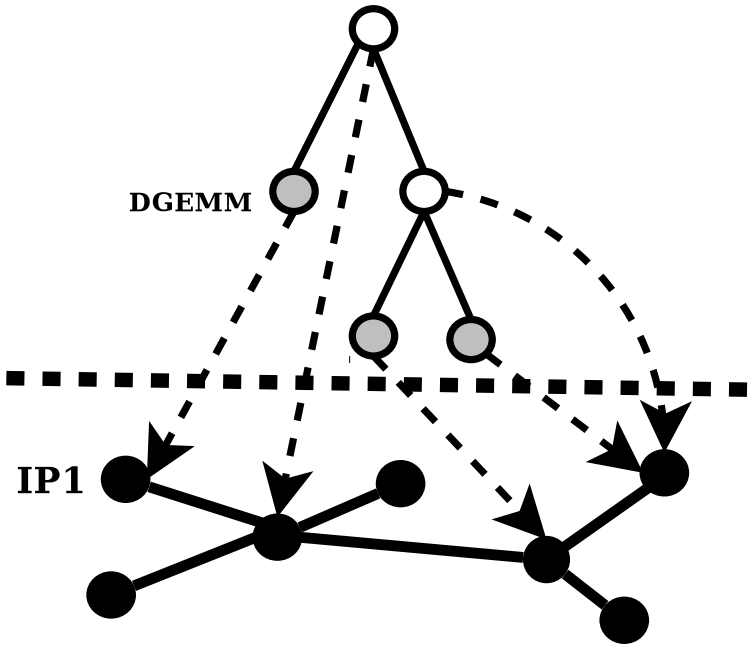
Basic entity: **peer** (processor)

- Message-Passing
- P_1 can communicate with P_2 if and only if P_1 "*knows*" P_2.
- Runs logical *nodes*

Logical Tree

Basic entity: (tree) **node**

- Executes the protocol
- Distributed among peers



P2P Network

Physical Network

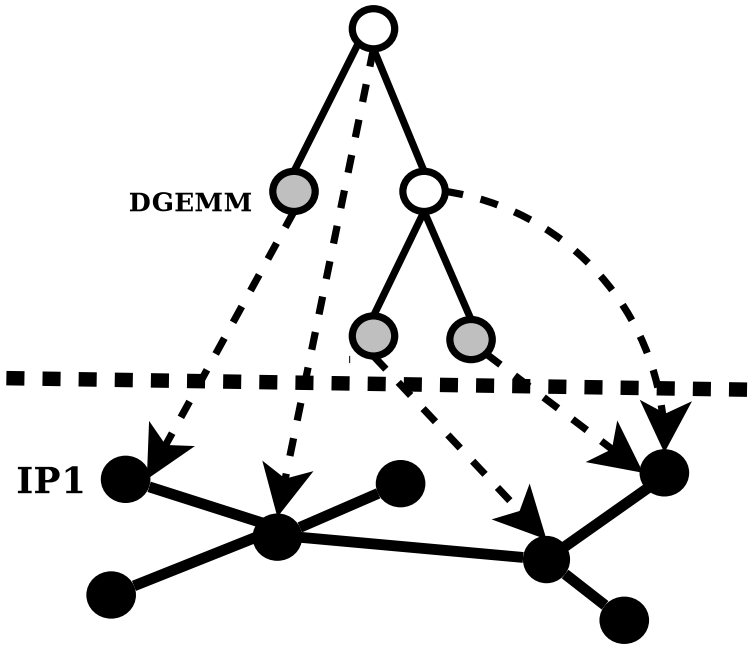
Basic entity: **peer** (processor)

- Message-Passing
- P_1 can communicate with P_2 if and only if P_1 "*knows*" P_2.
- Runs logical *nodes*

Logical Tree

Basic entity: (tree) **node**

- Executes the protocol
- Distributed among peers



Outline

Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height

Definition

Each node is the Proper Greatest Common Prefix of any pair of its children

Proper Greatest Common Prefix Tree (PGCP Tree)

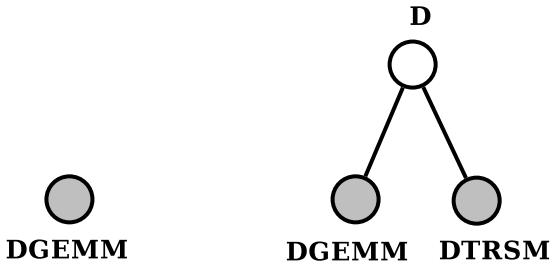
Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height

Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

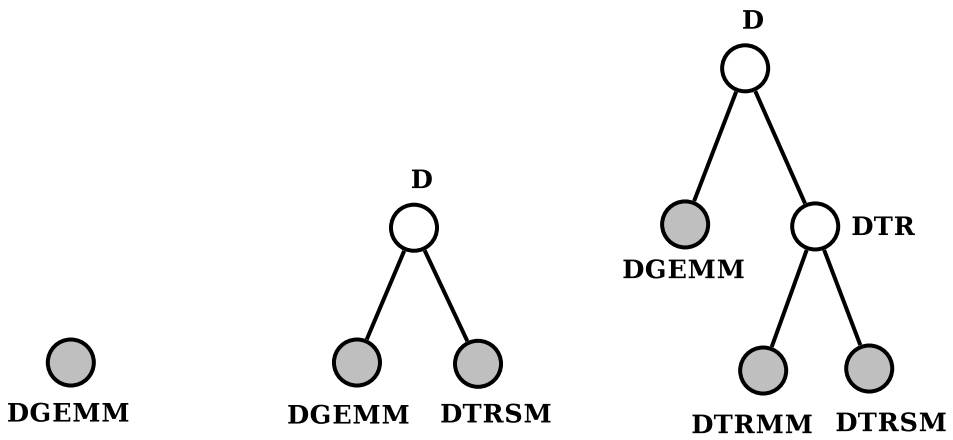
- Dynamically constructed
- Bounded degree and height



Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

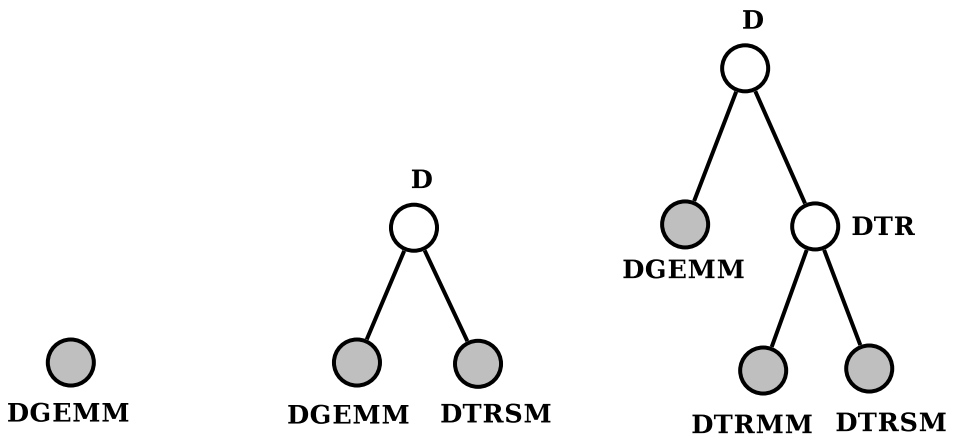
- Dynamically constructed
- Bounded degree and height



Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

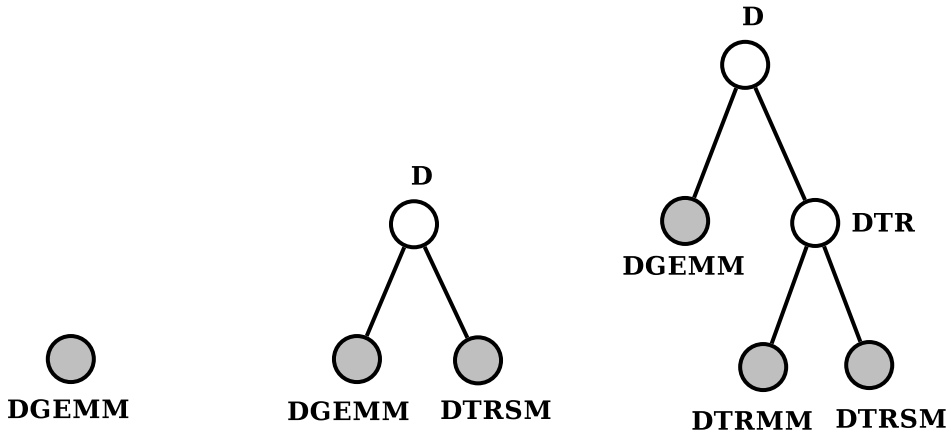
- Dynamically constructed
- Bounded degree and height



Proper Greatest Common Prefix Tree (PGCP Tree)

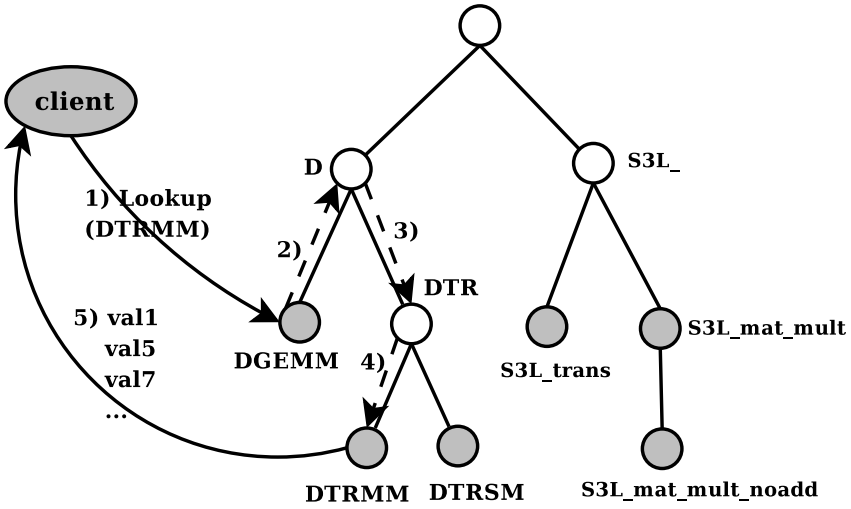
Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height



Lookup

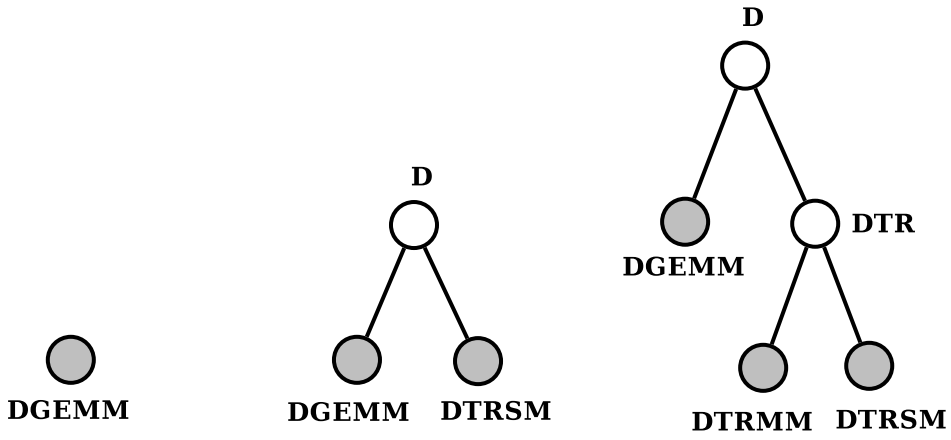
- Exact match
- Autocompletion
- Range queries



Proper Greatest Common Prefix Tree (PGCP Tree)

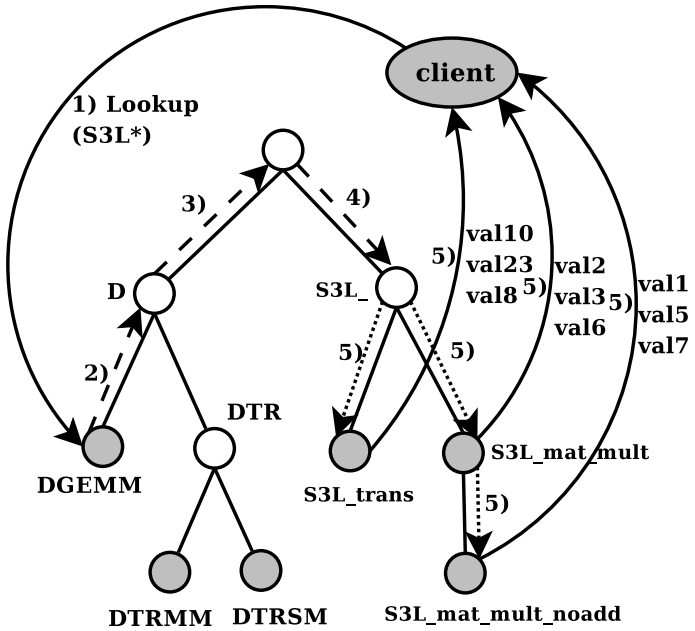
Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height



Lookup

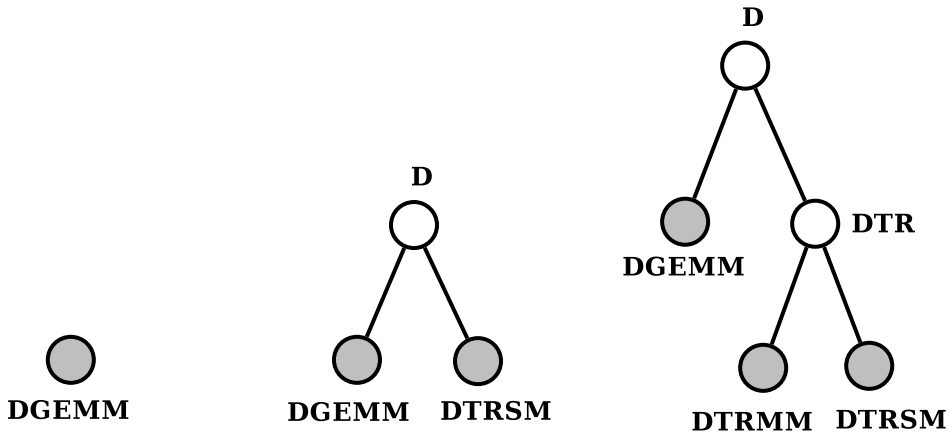
- Exact match
- Autocompletion
- Range queries



Proper Greatest Common Prefix Tree (PGCP Tree)

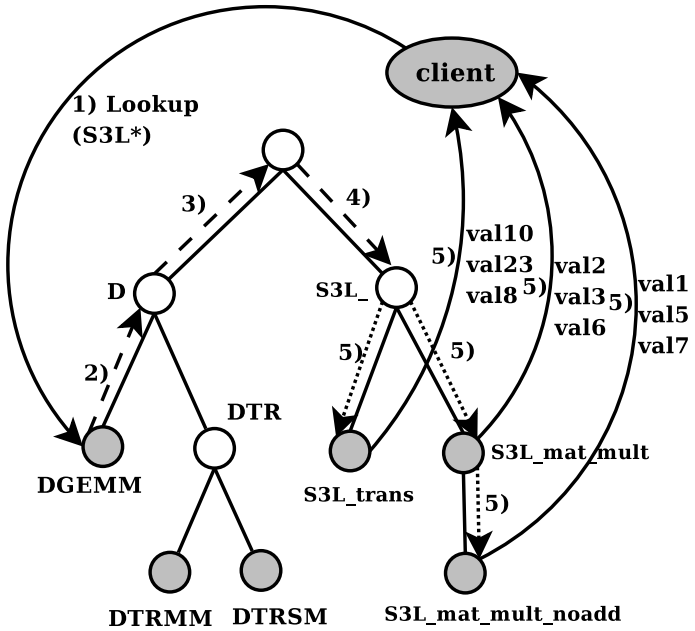
Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height



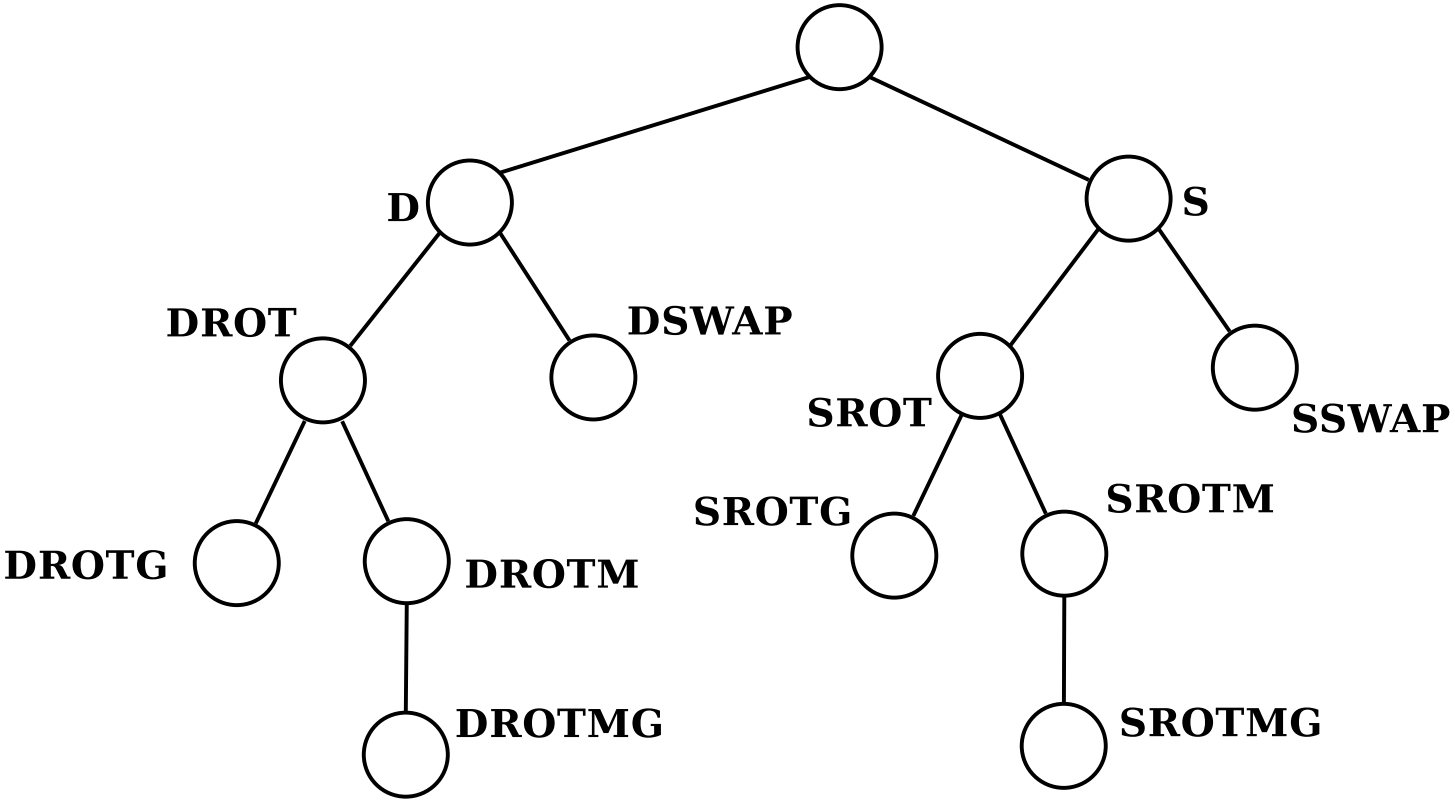
Lookup

- Exact match
- Autocompletion
- Range queries



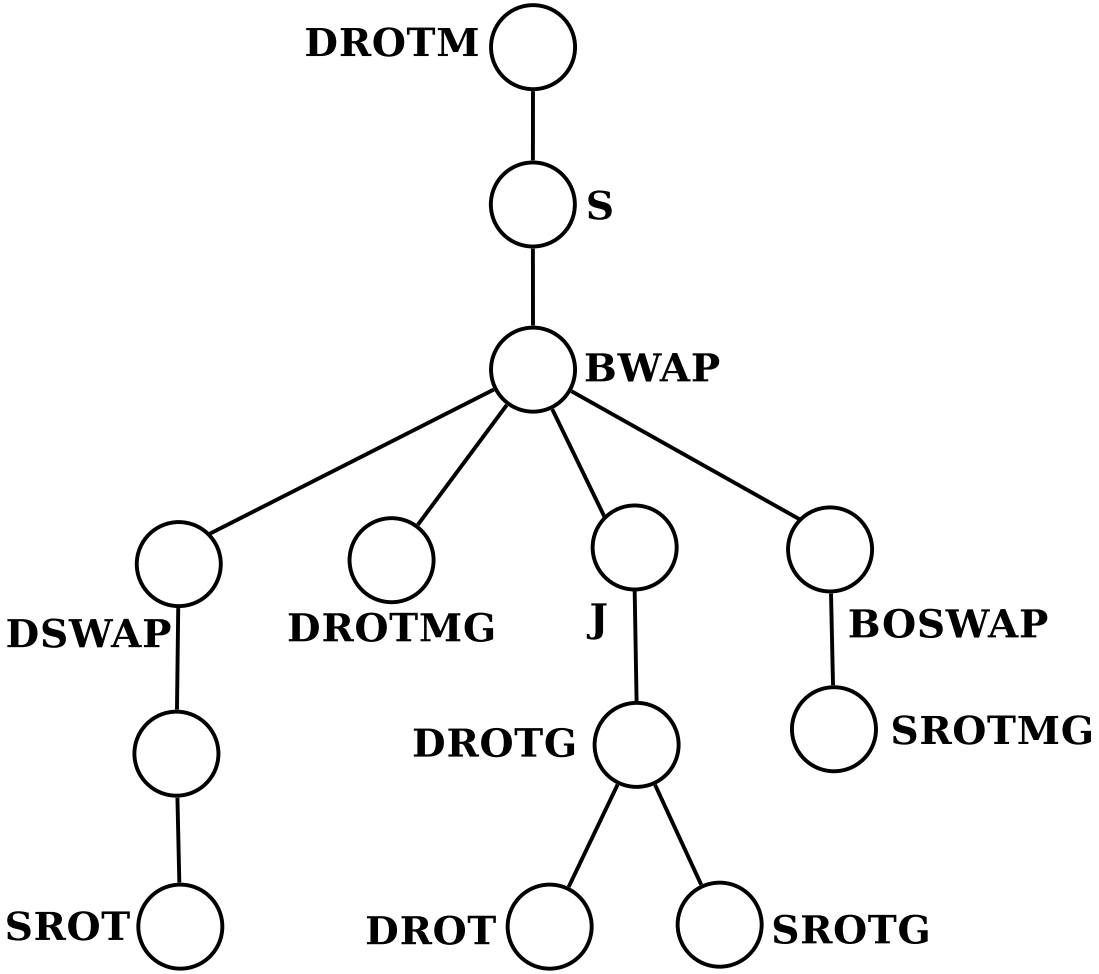
Arbitrary Initial Configuration

Expected PGCP Tree



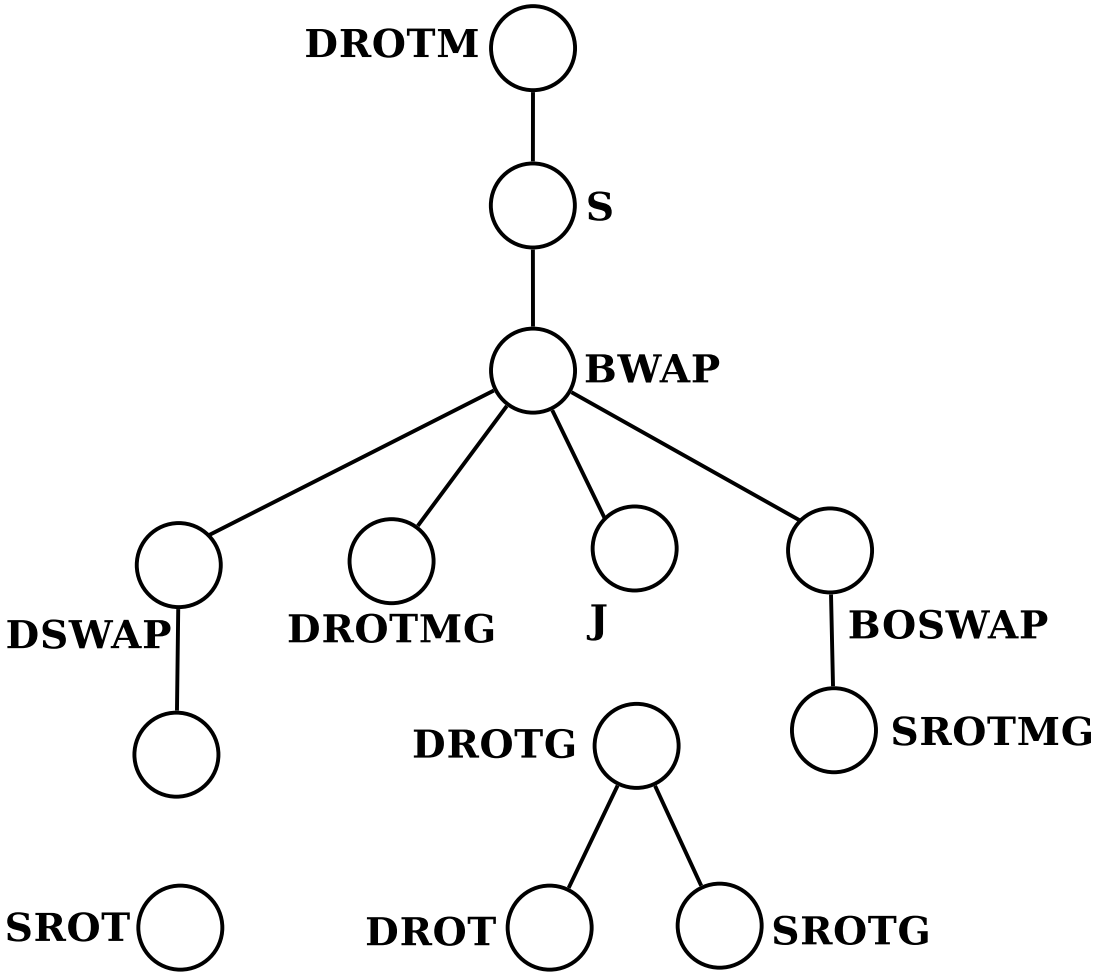
Arbitrary Initial Configuration

An Arbitrary Labeled Tree



Arbitrary Initial Configuration

An Arbitrary Labeled Forest



Outline

Protocol

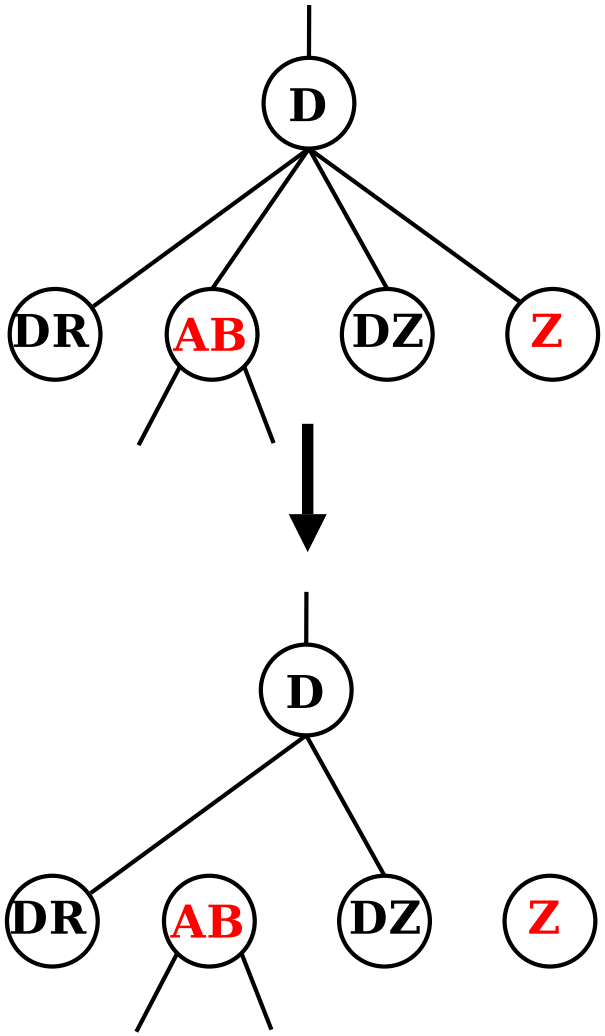
Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing

Protocol

Infinite Cycle (Timeout)

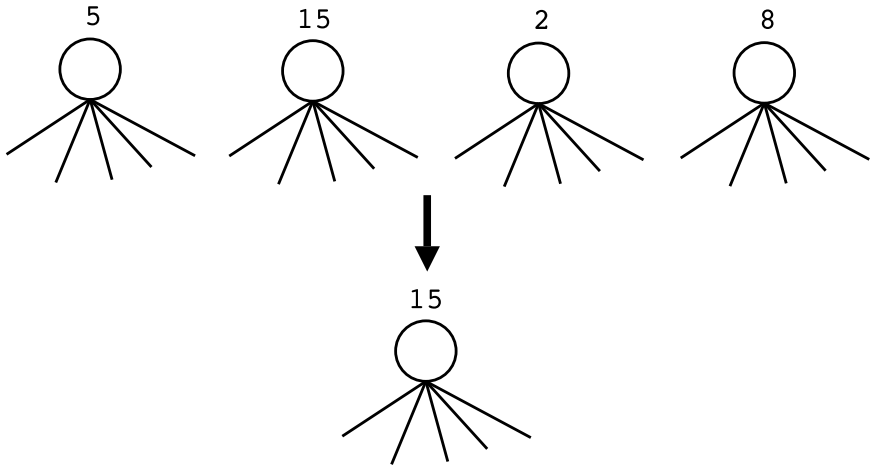
1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - Loop



Protocol

Infinite Cycle (Timeout)

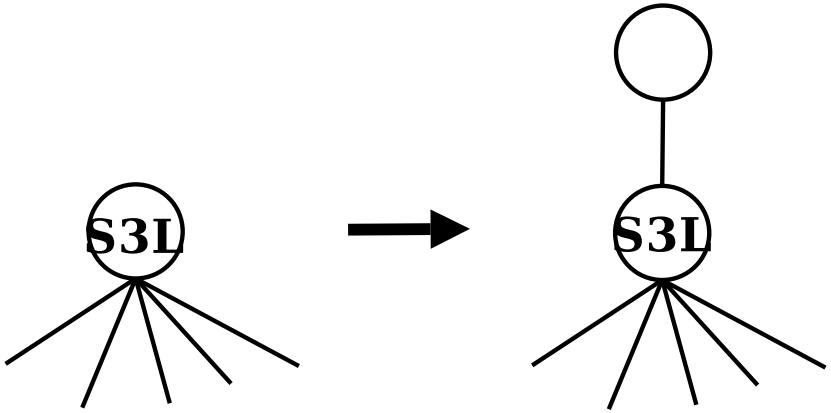
1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP



Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP



Protocol

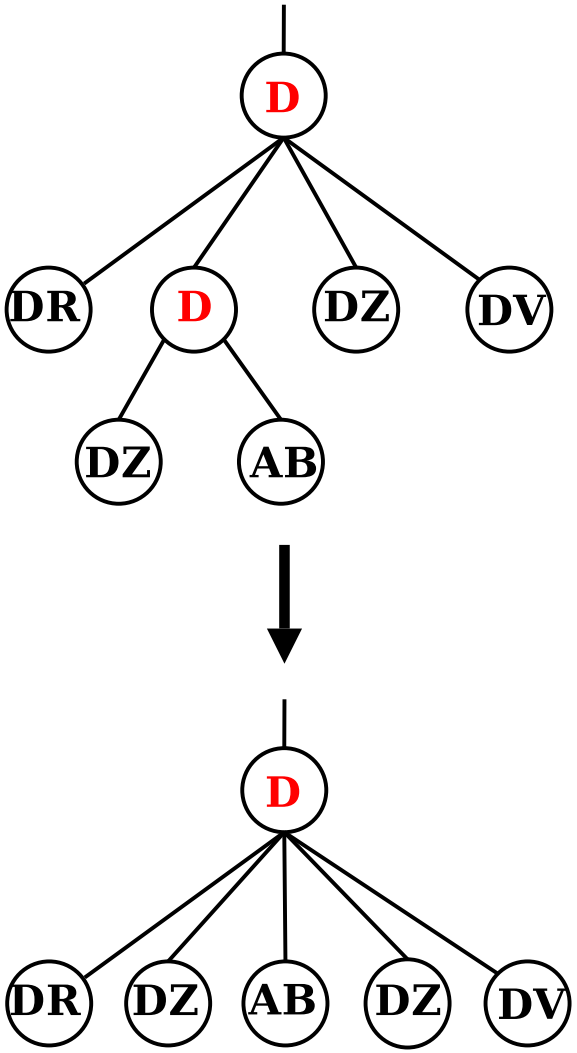
Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP

Protocol

Infinite Cycle (Timeout)

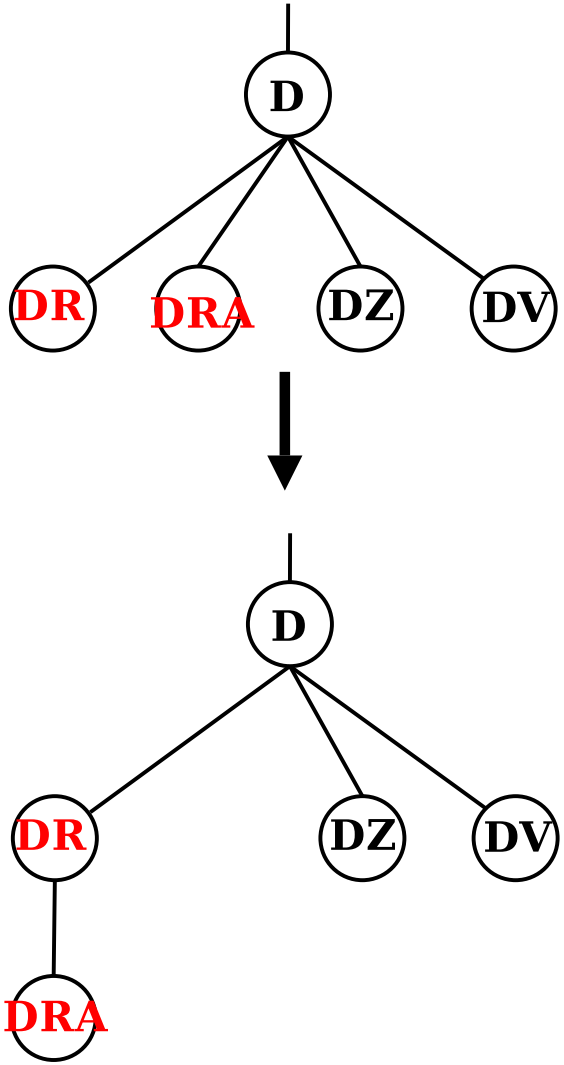
1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP



Protocol

Infinite Cycle (Timeout)

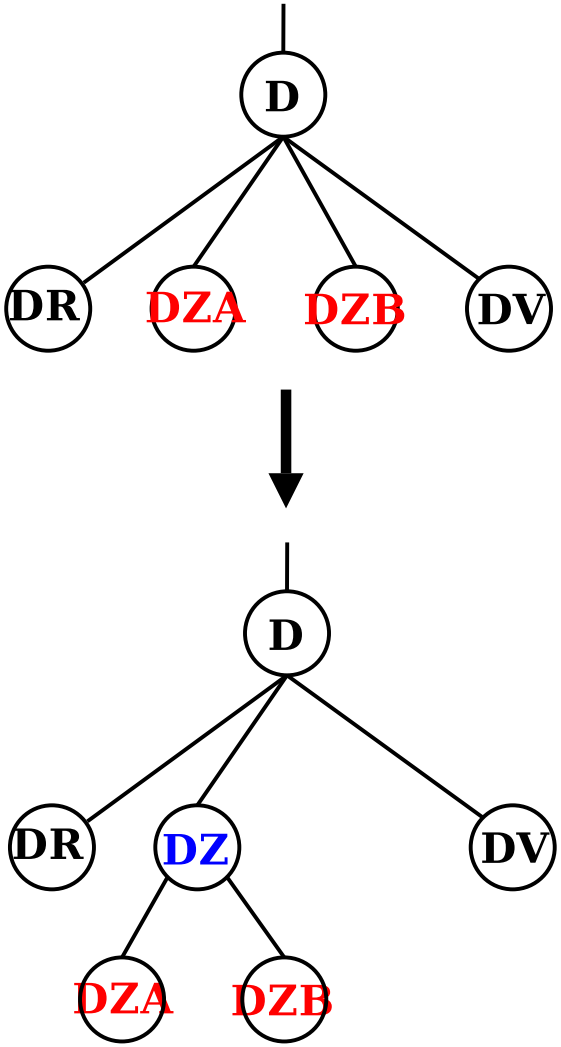
1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP



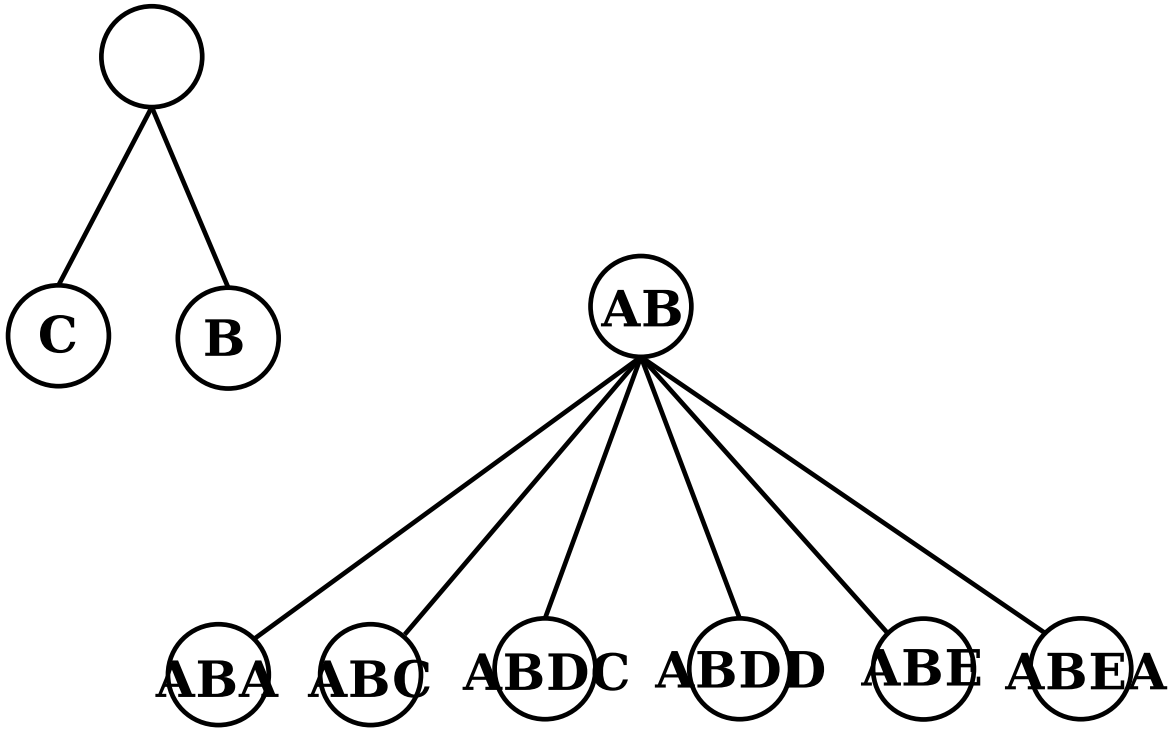
Protocol

Infinite Cycle (Timeout)

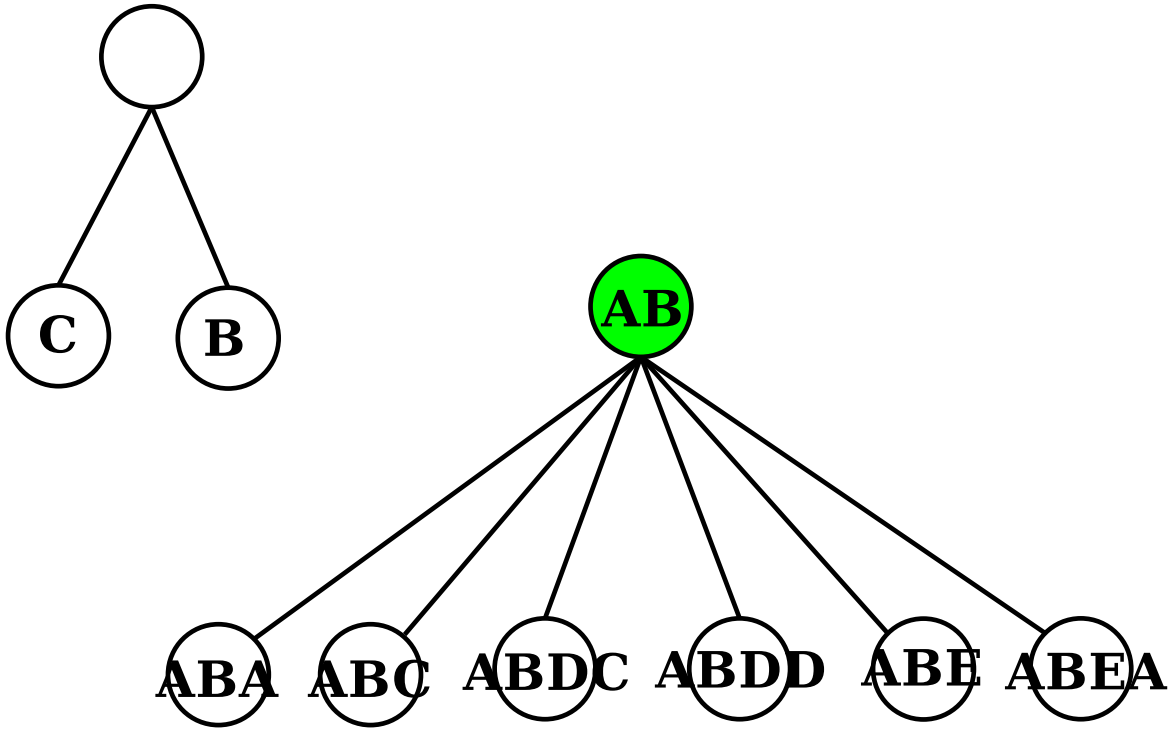
1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots
2. Children processing
   - Merge
   - Prefix Relation
   - GCP



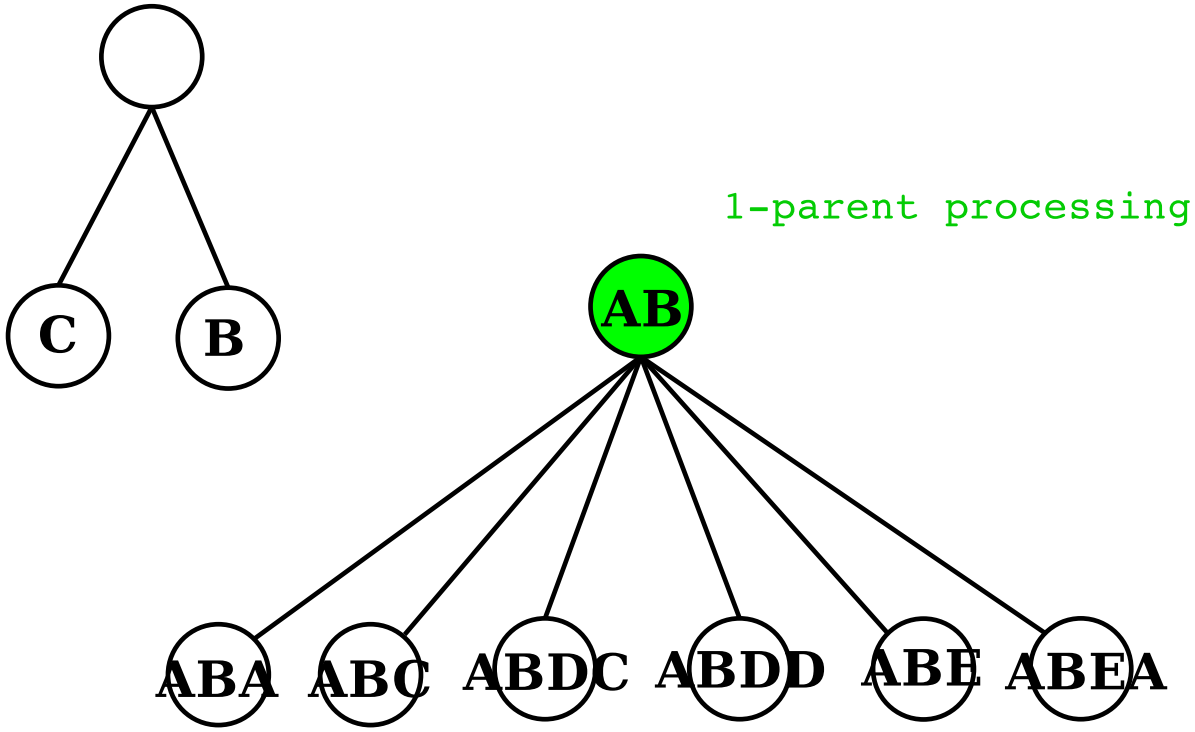
Protocol



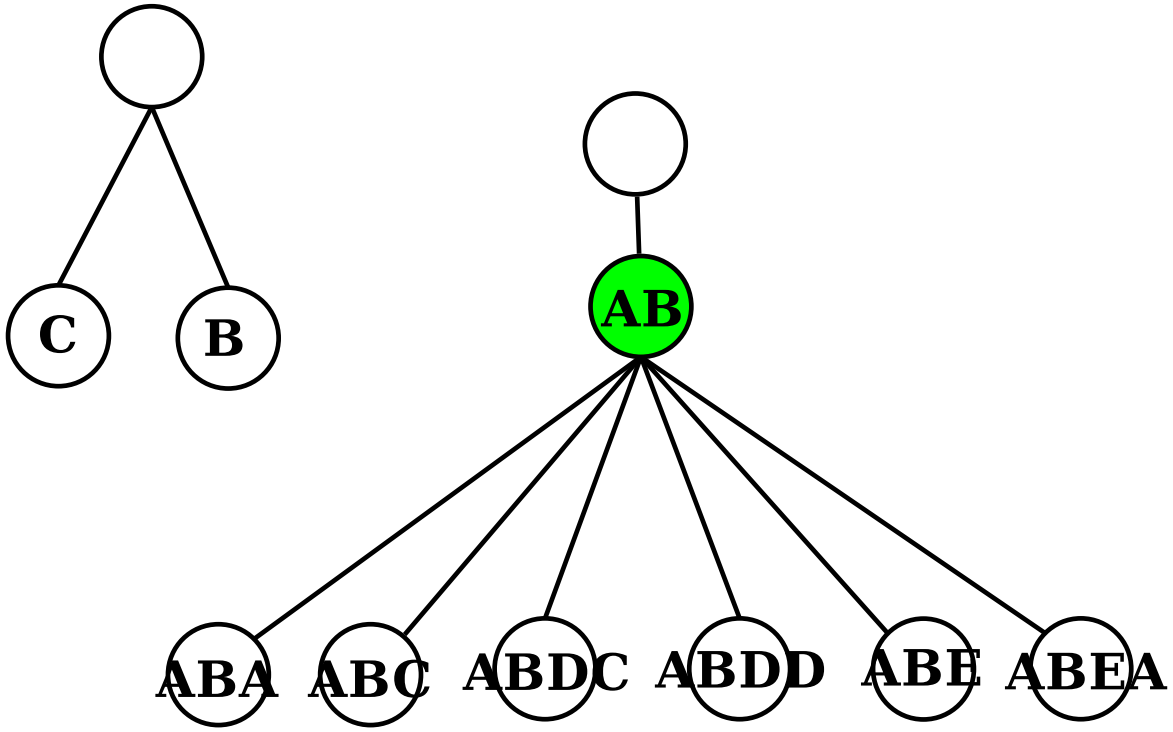
Protocol



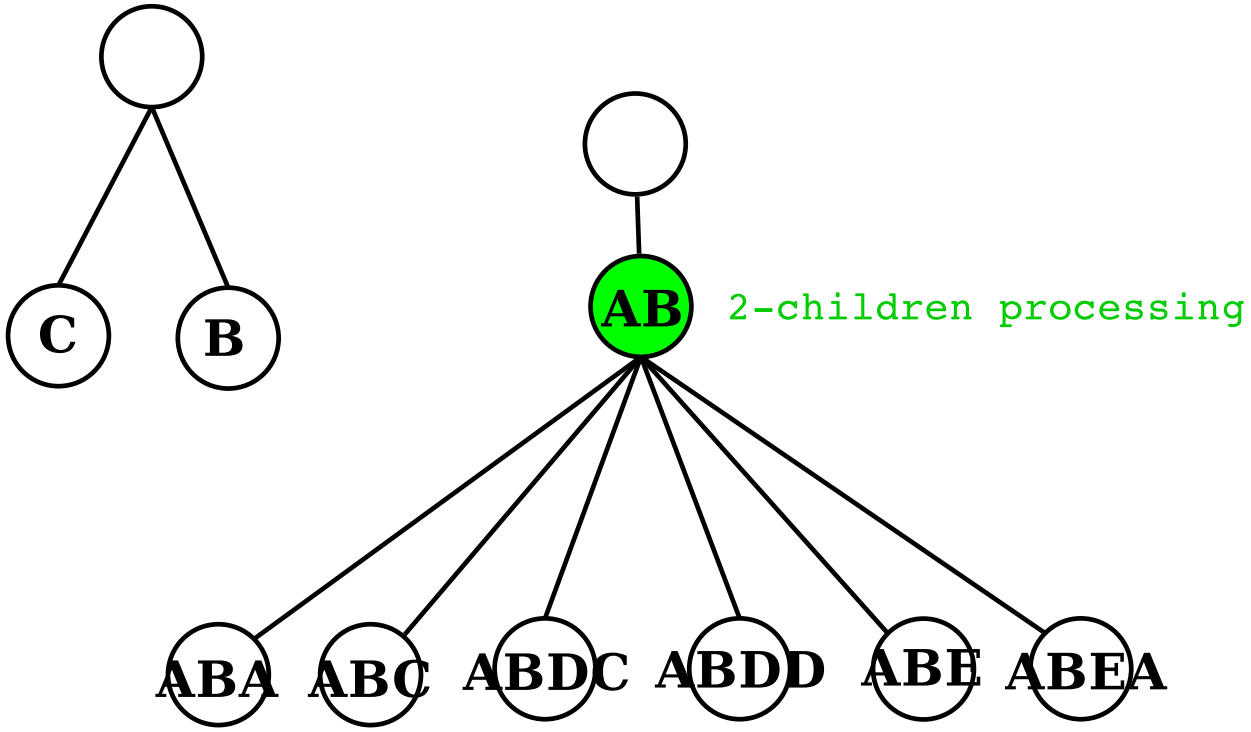
Protocol



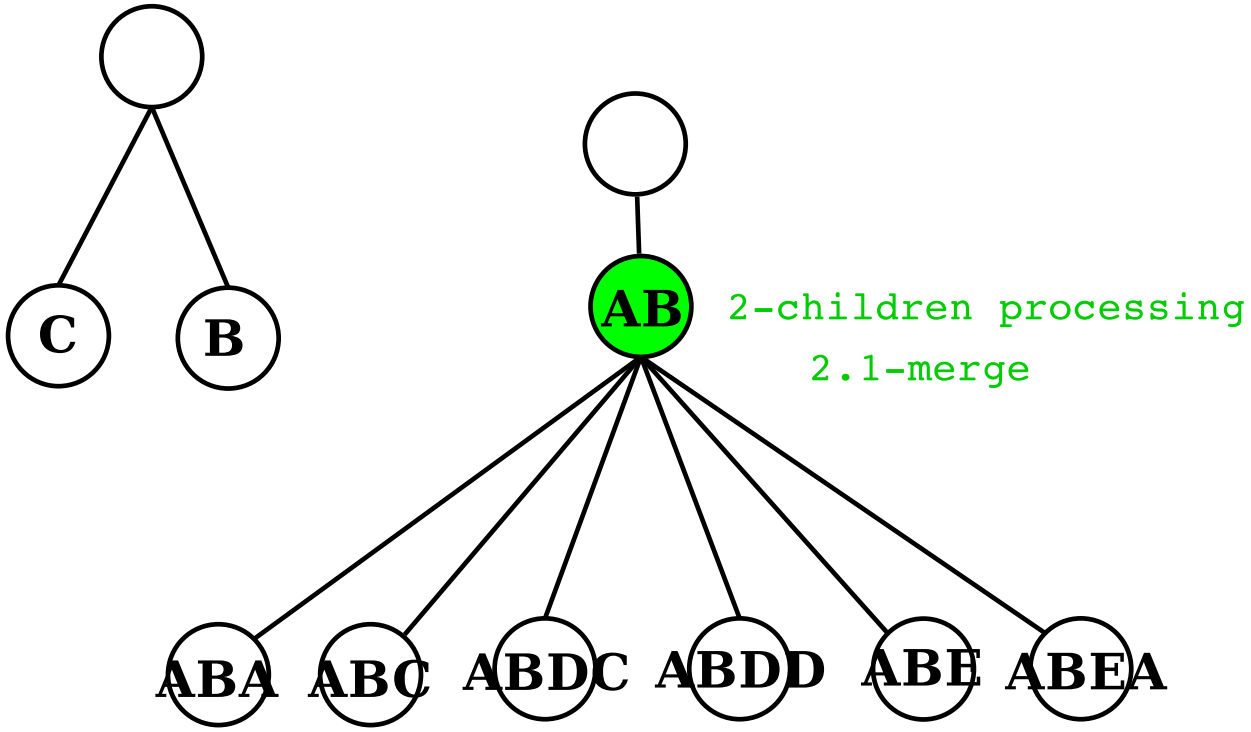
Protocol



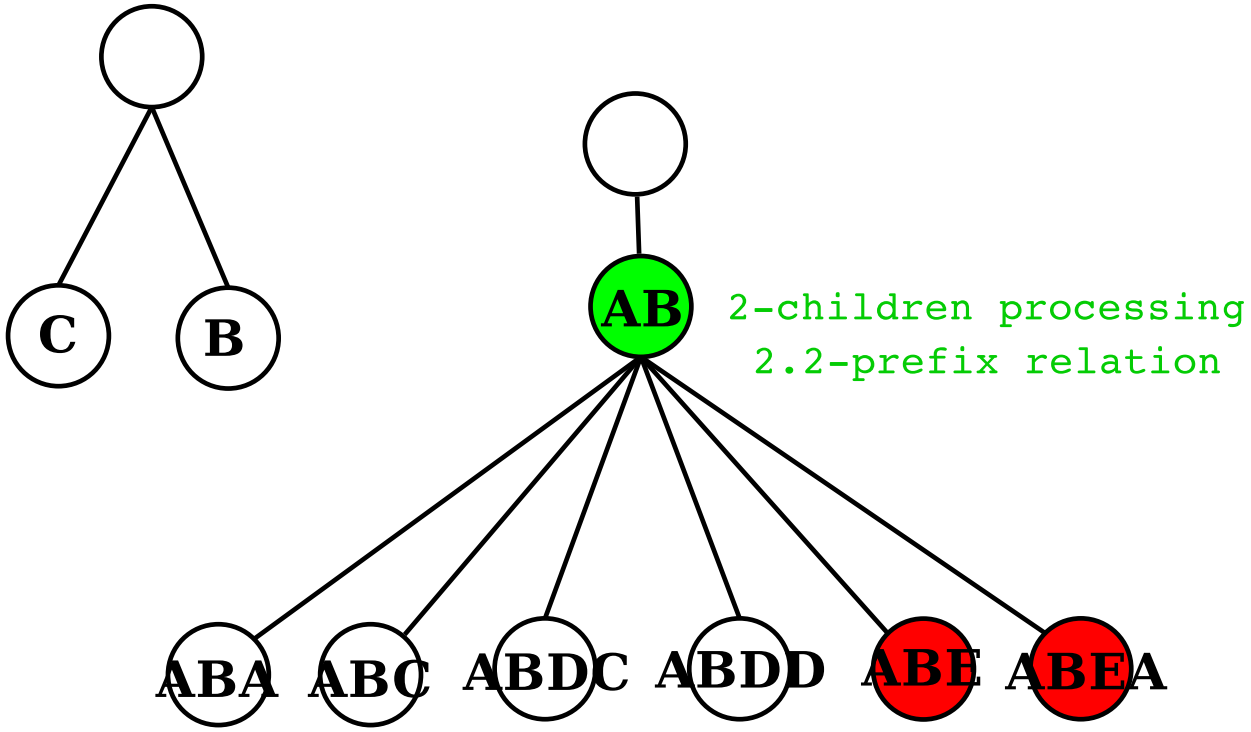
Protocol



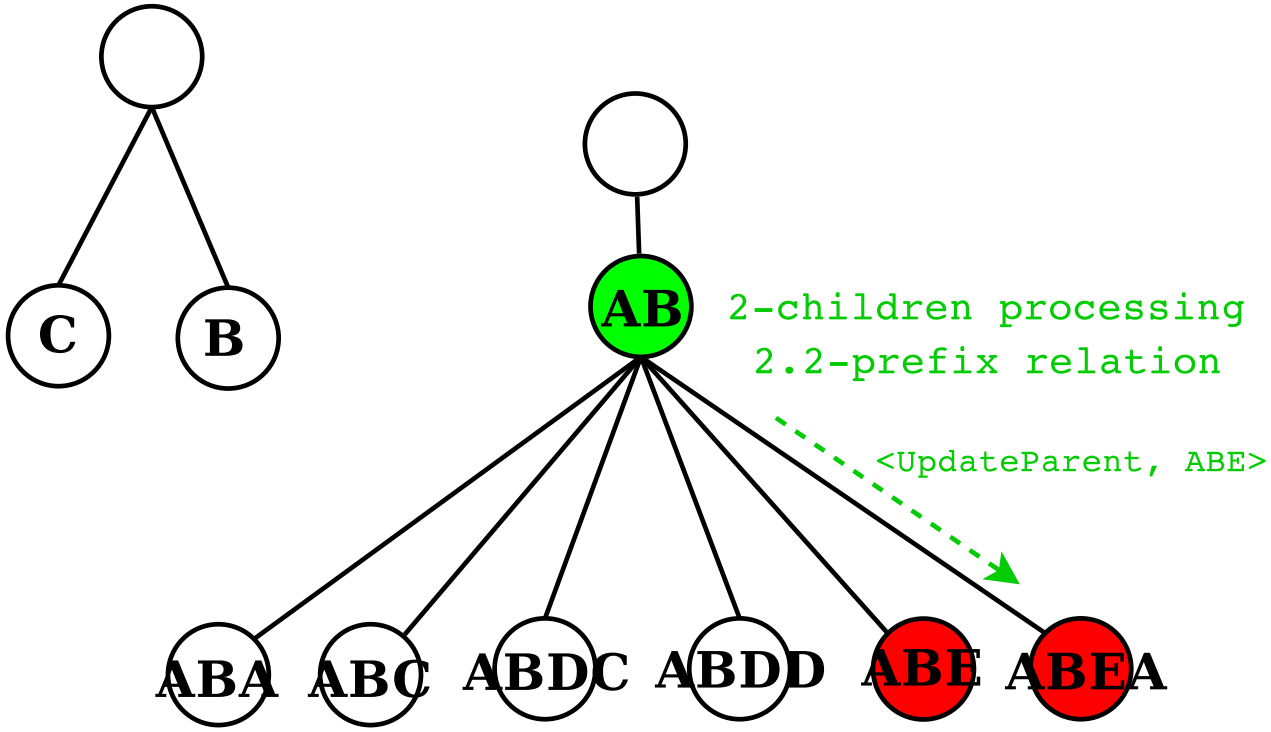
Protocol


C
B
AB
2-children processing
2.1-merge
ABA
ABC
ABDC
ABDD
ABE
ABEA

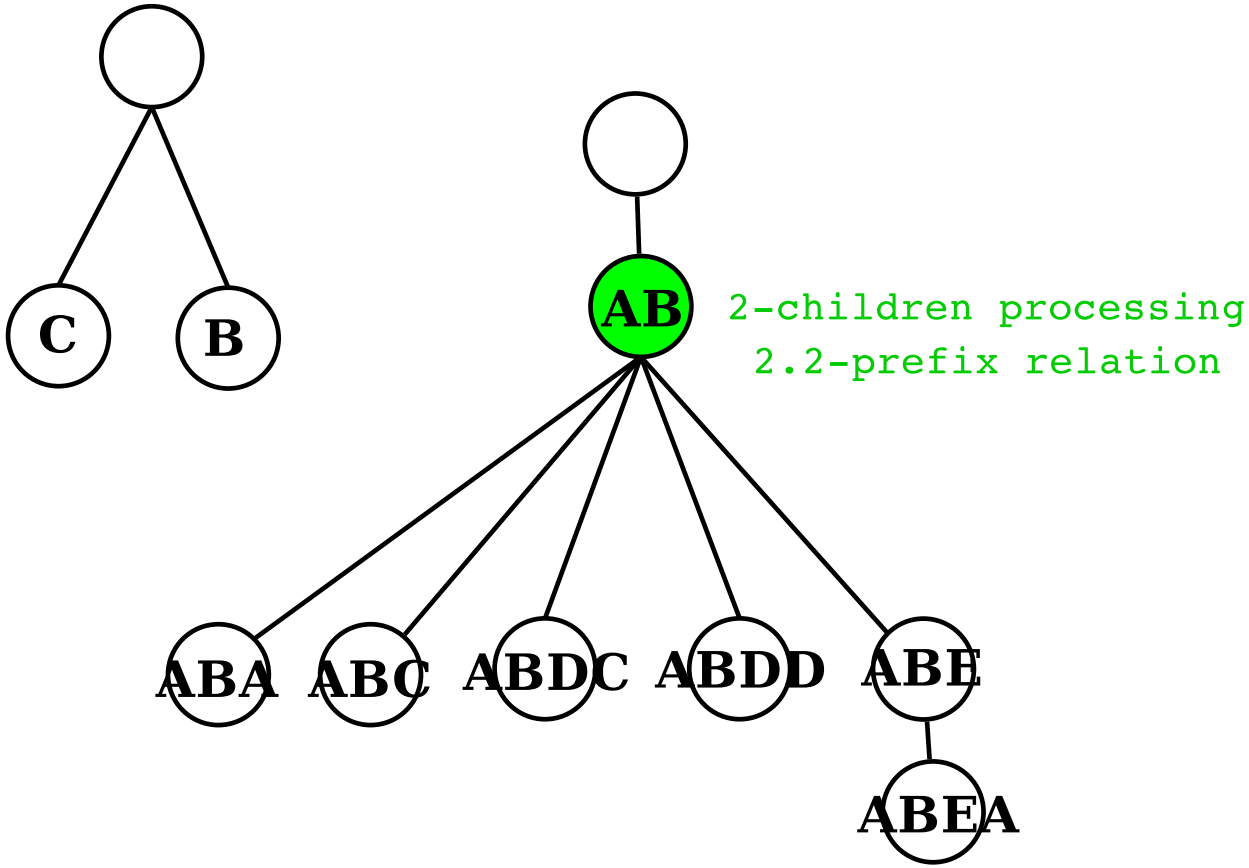
Protocol


C
B
AB
2-children processing
2.2-prefix relation
ABA
ABC
ABDC
ABDD
ABE
ABEA

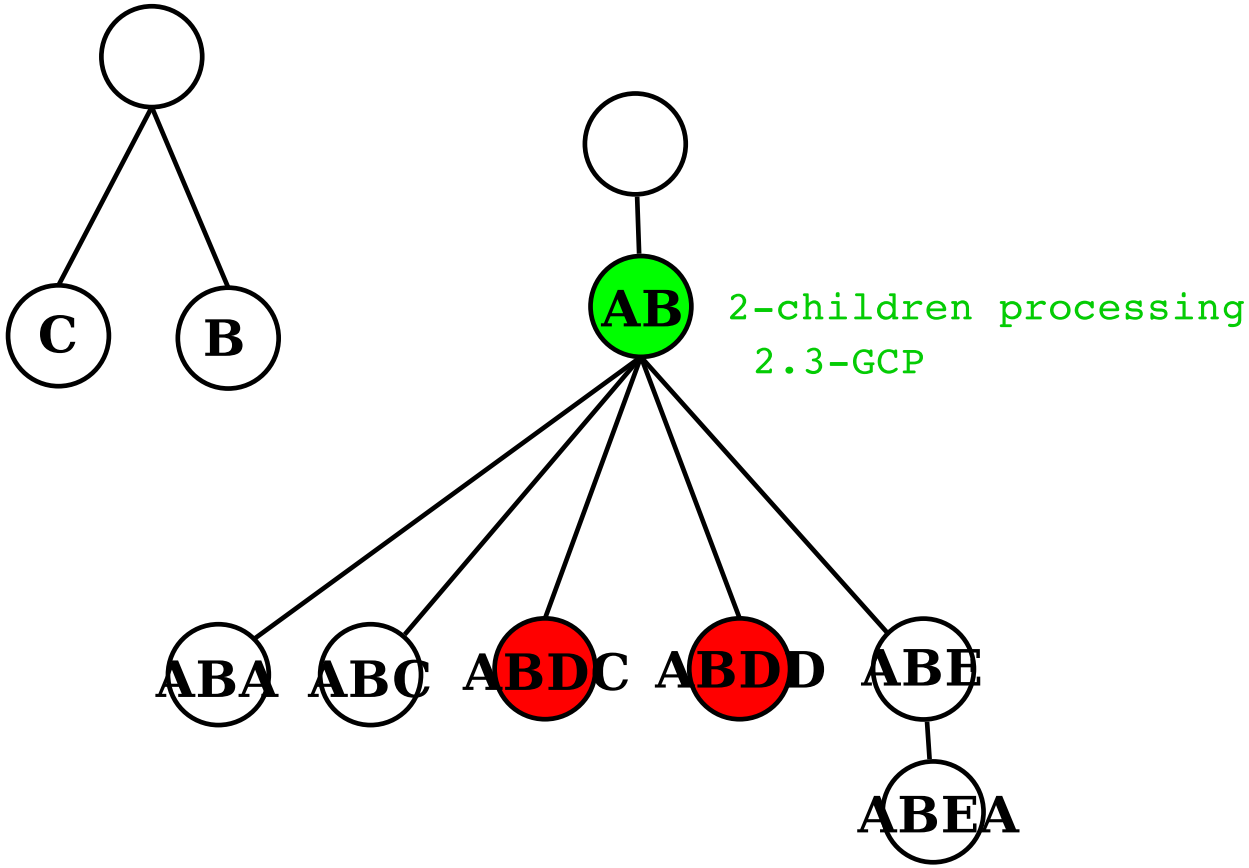
Protocol


C
B
AB
2-children processing
2.2-prefix relation
<UpdateParent, ABE>
ABA
ABC
ABDC
ABDD
ABE
ABEA

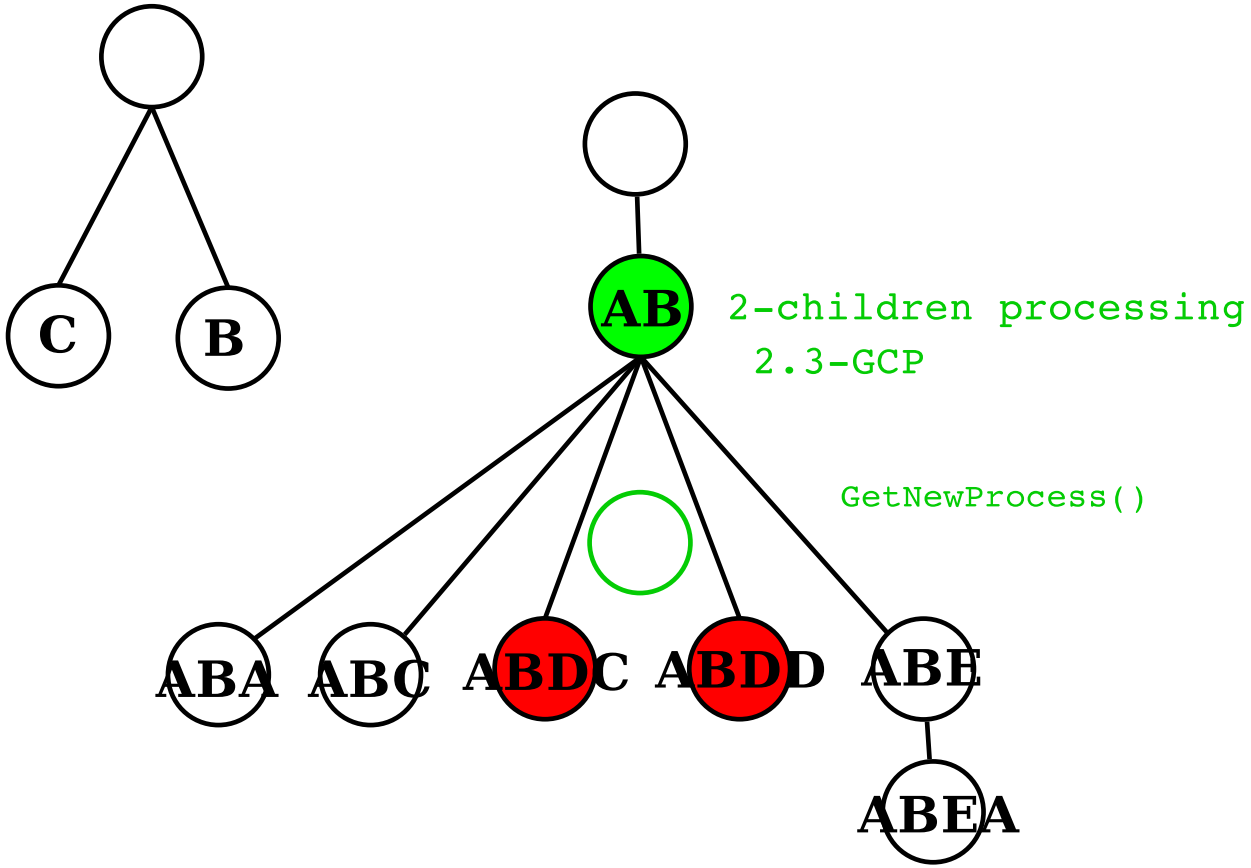
Protocol


C
B
AB
2-children processing
2.2-prefix relation
ABA
ABC
ABDC
ABDD
ABE
ABEA

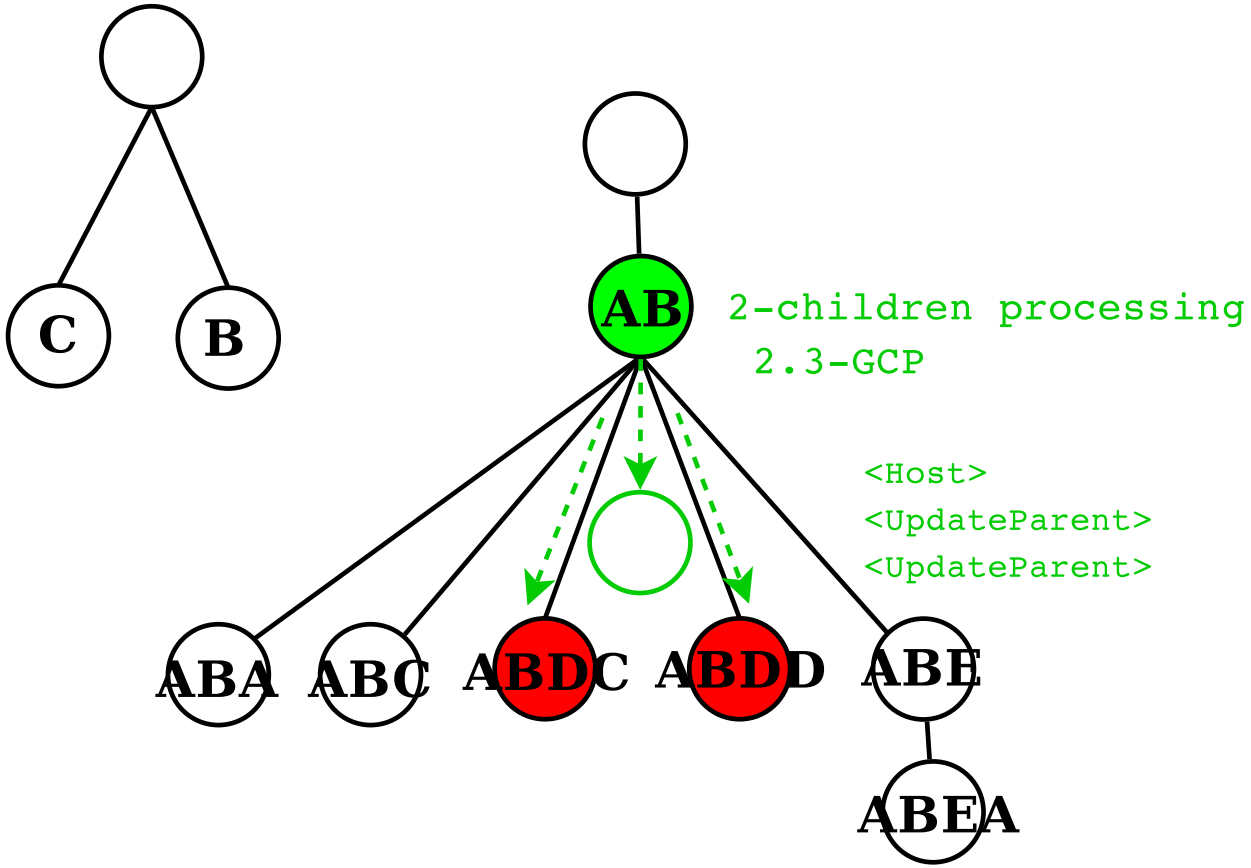
Protocol



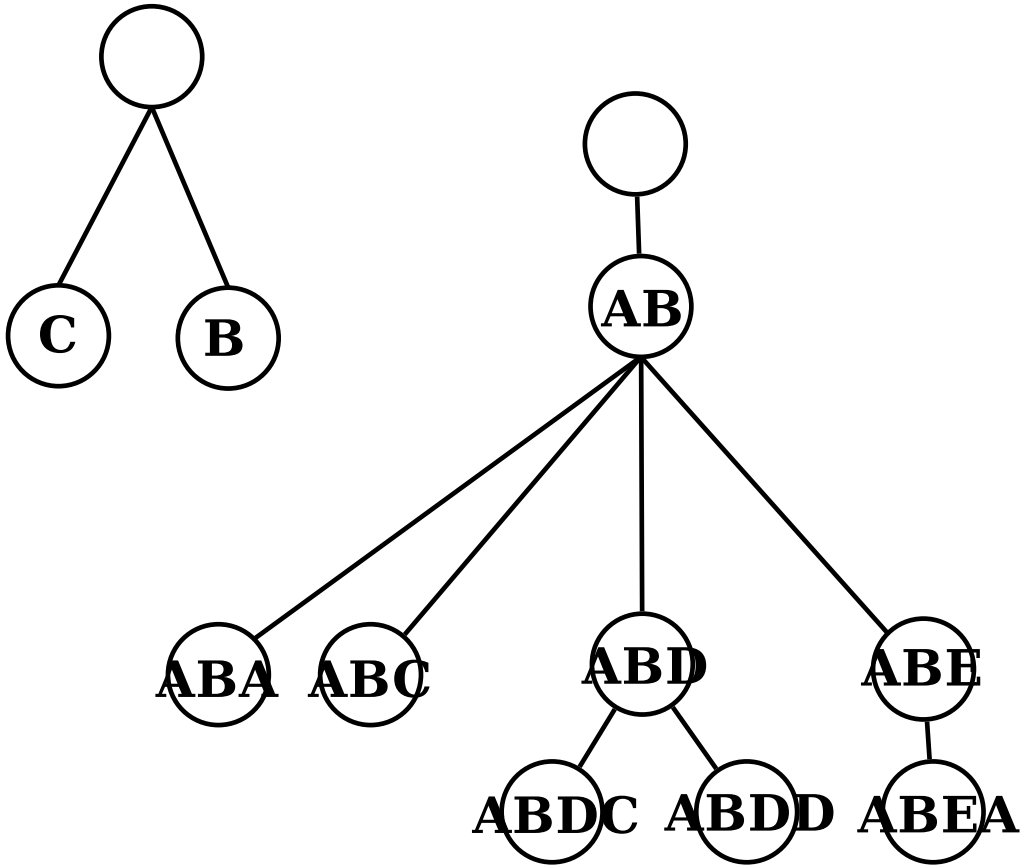
Protocol


C
B
AB
2-children processing
2.3-GCP
GetNewProcess()
ABA
ABC
ABDC
ABDD
ABE
ABEA

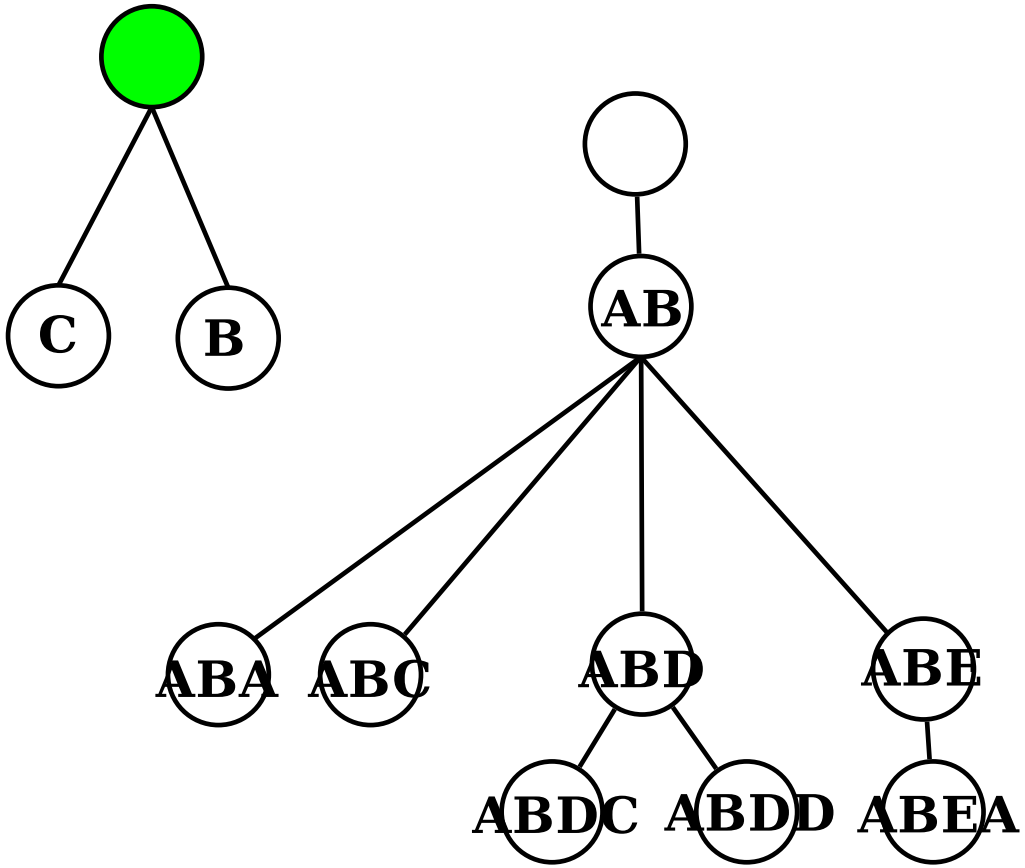
Protocol


C
B
AB
2-children processing
2.3-GCP
<Host>
<UpdateParent>
<UpdateParent>
ABA
ABC
ABDC
ABDD
ABE
ABEA

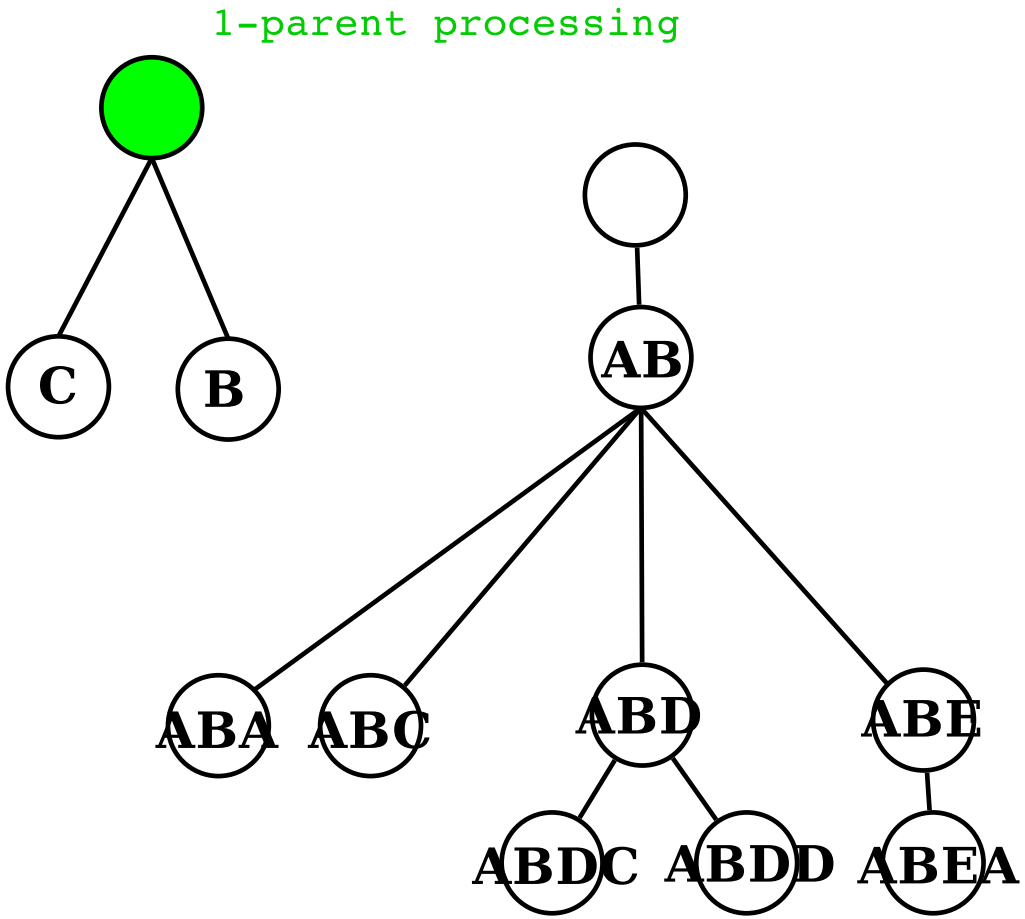
Protocol



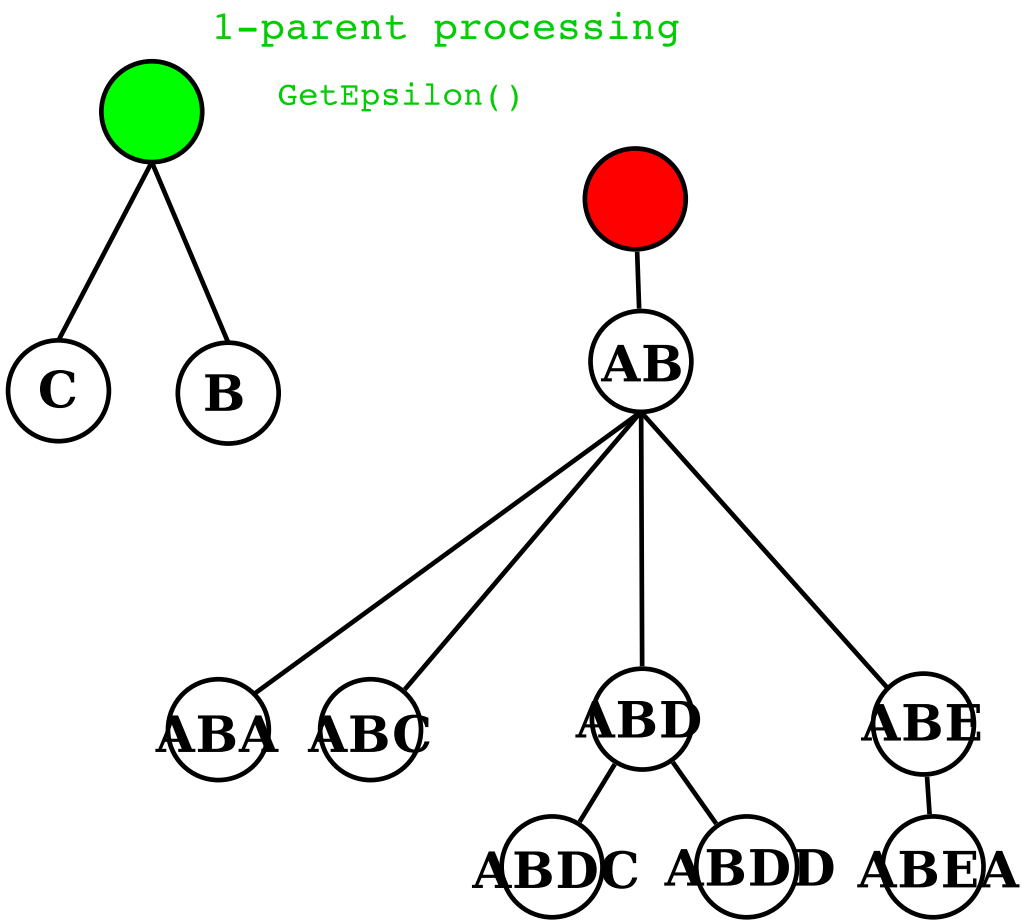
Protocol



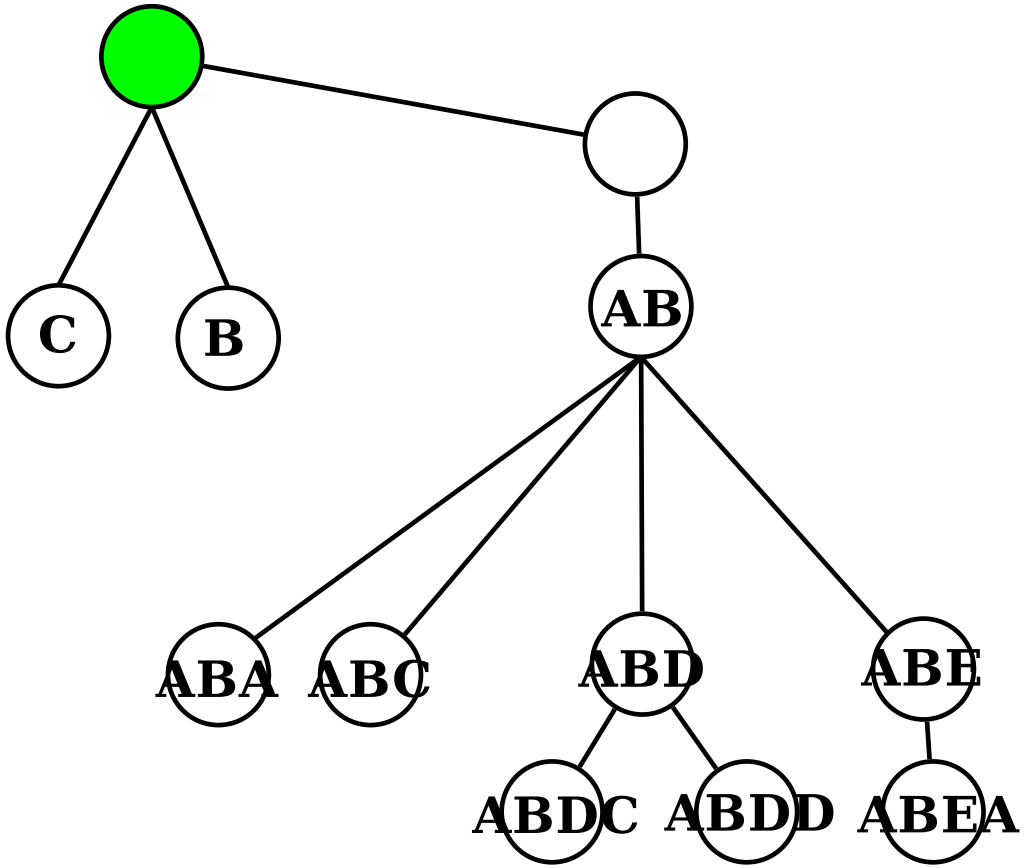
Protocol



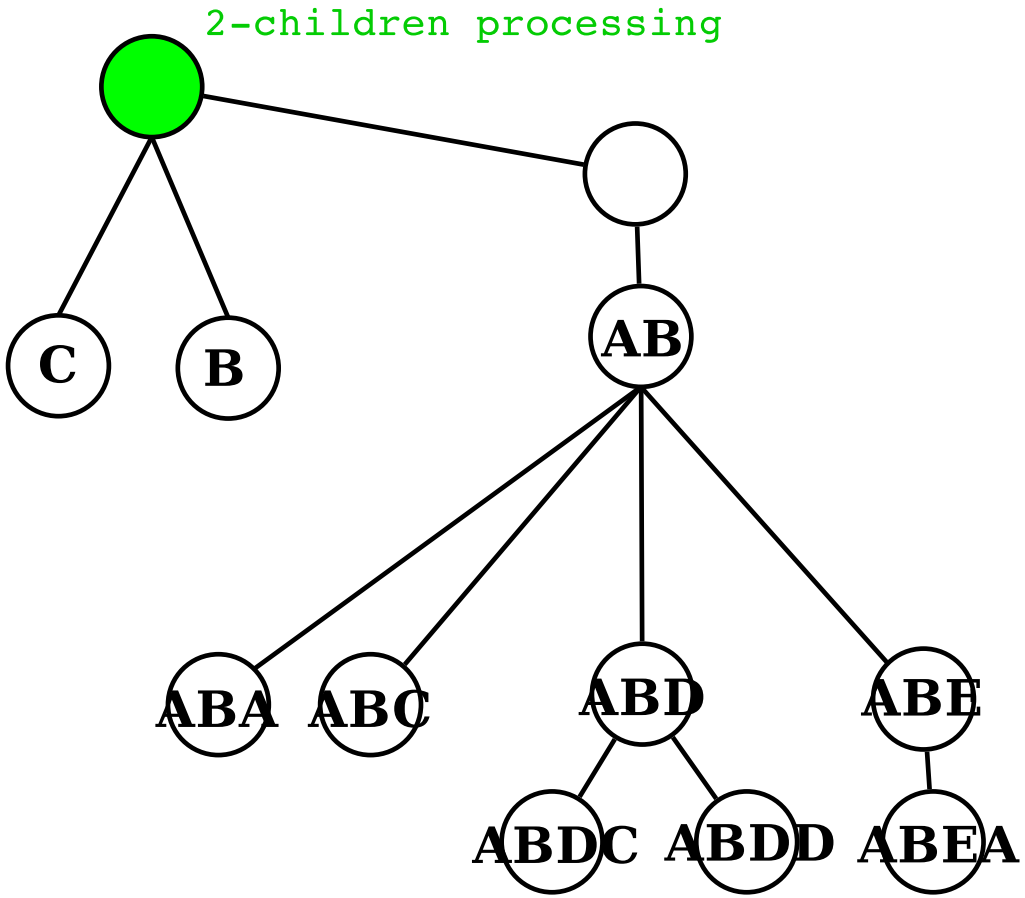
Protocol


1-parent processing
GetEpsilon()
C
B
AB
ABA
ABC
ABD
ABE
ABDC
ABDD
ABEA

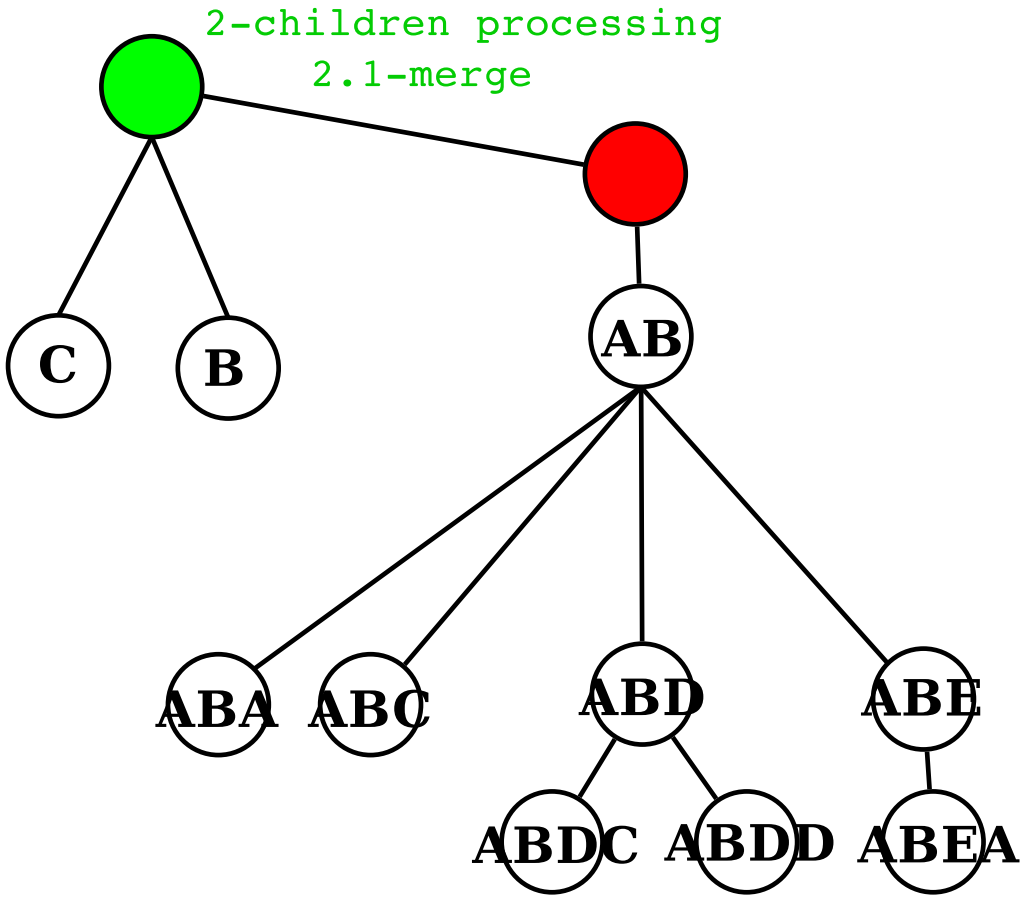
Protocol



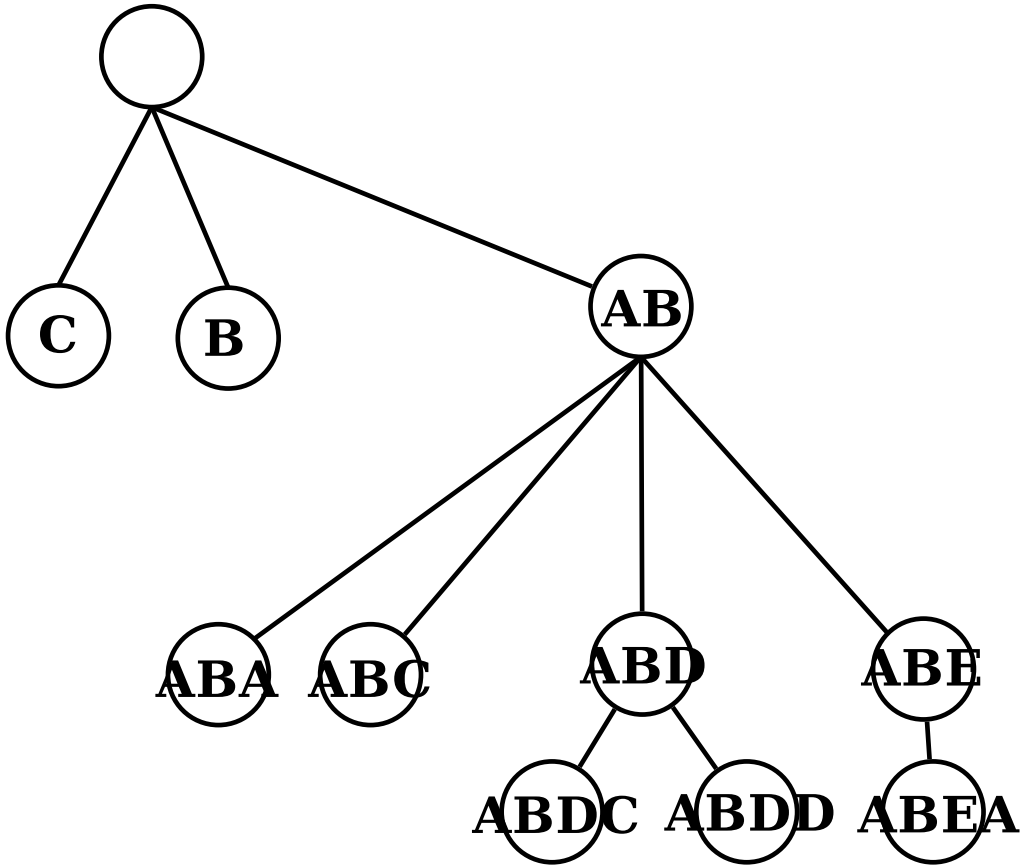
Protocol



Protocol

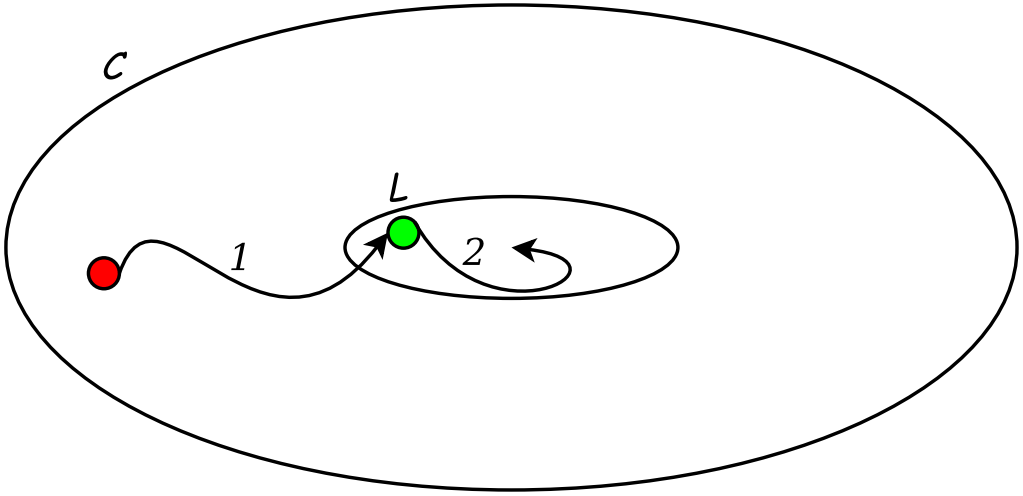

2-children processing
2.1-merge
C
B
AB
ABA
ABC
ABD
ABE
ABDC
ABDD
ABEA

Protocol



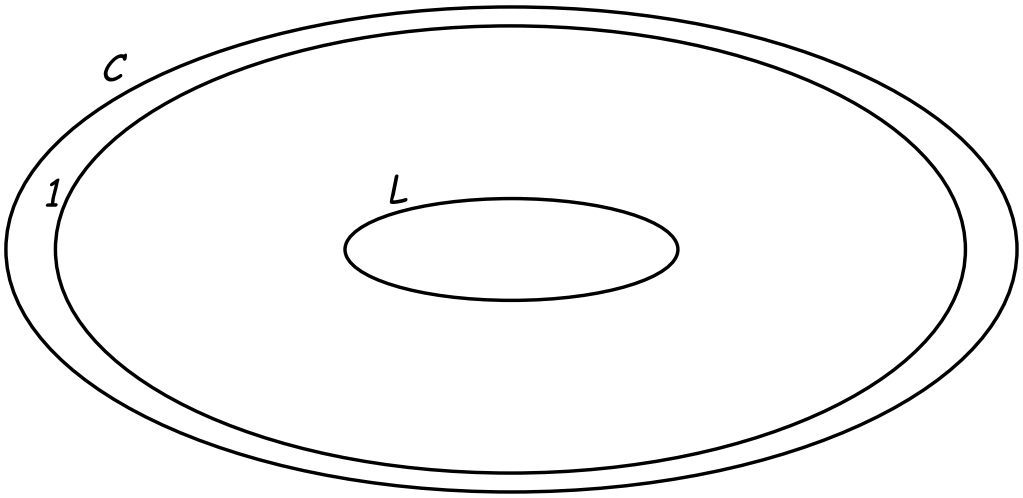
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



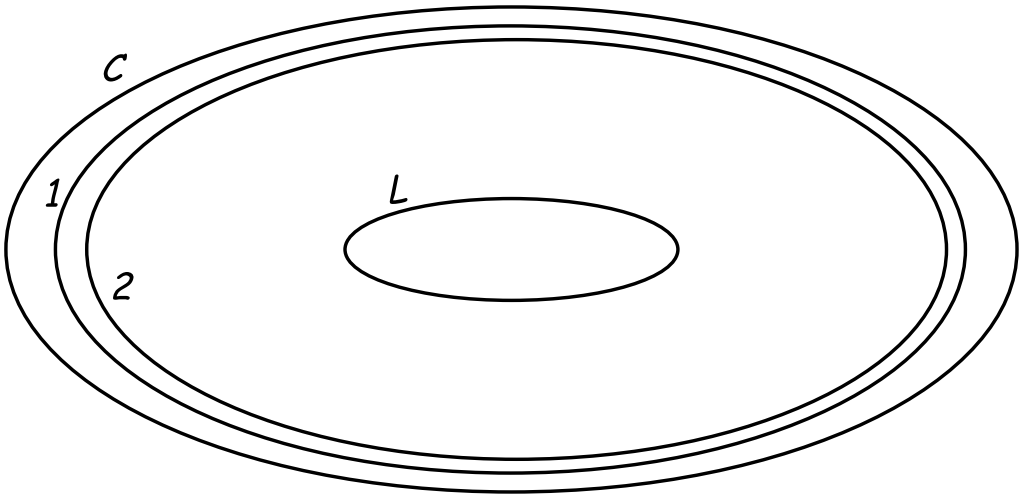
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



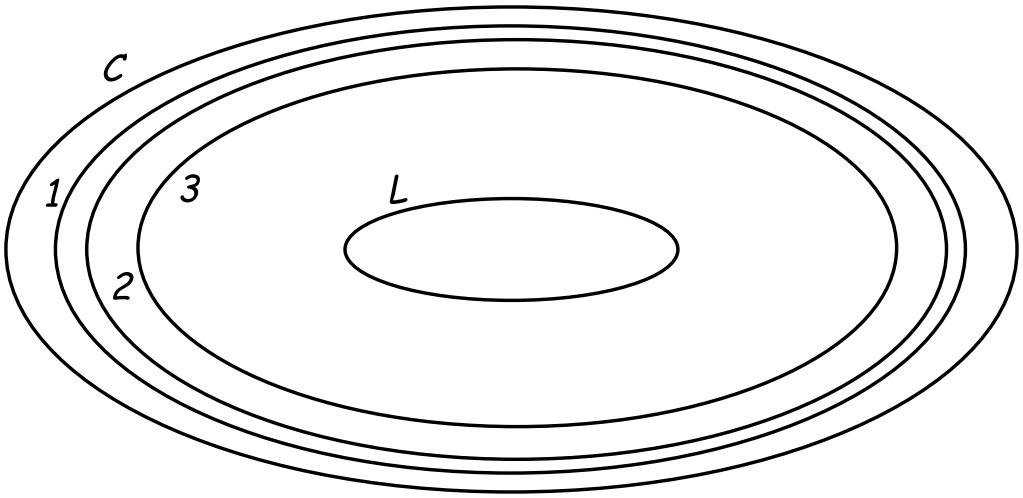
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



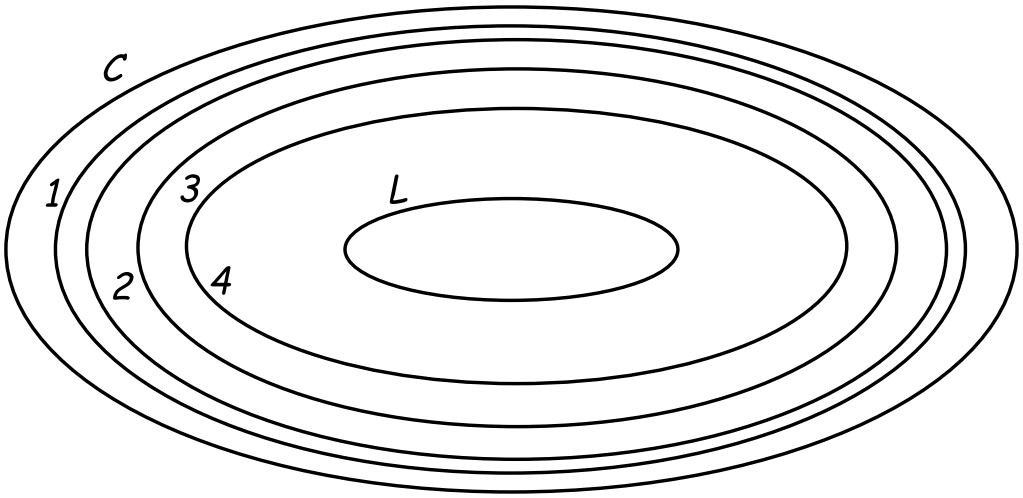
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



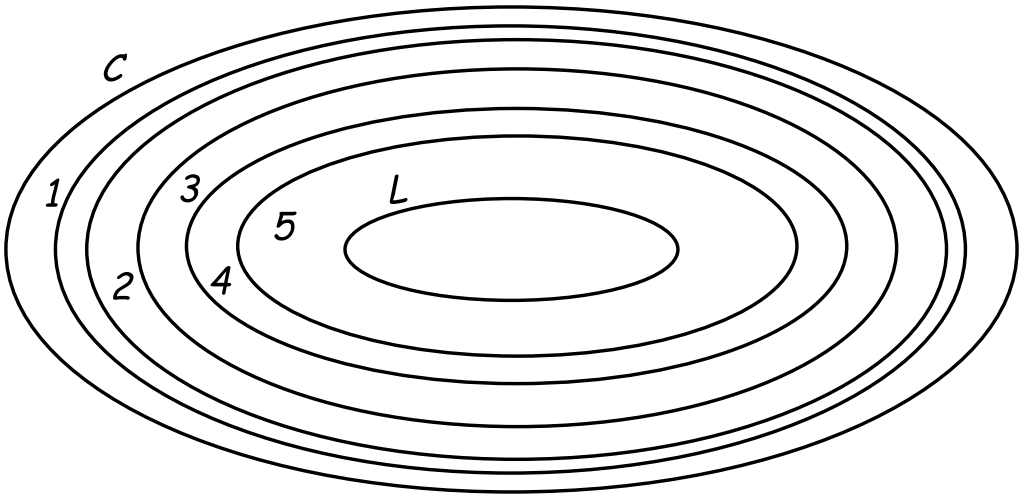
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



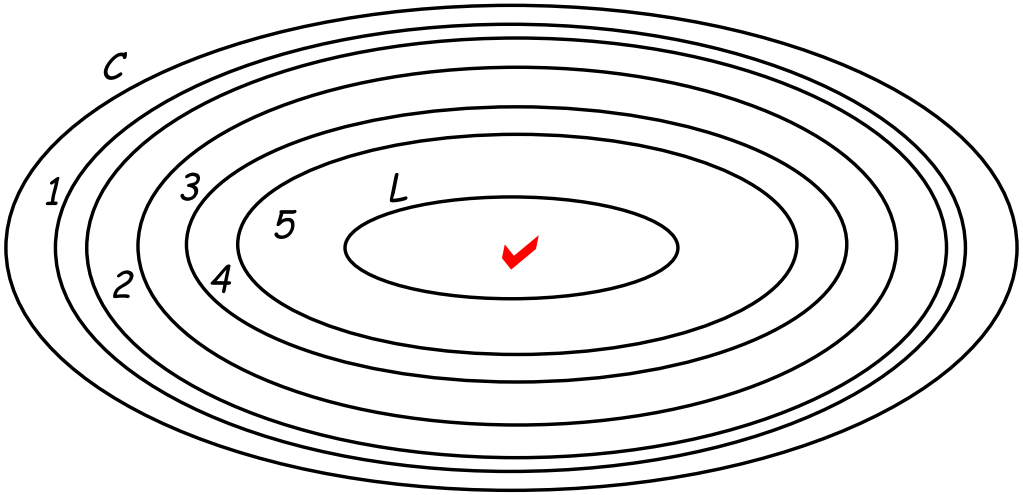
Sketch of proof

1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition

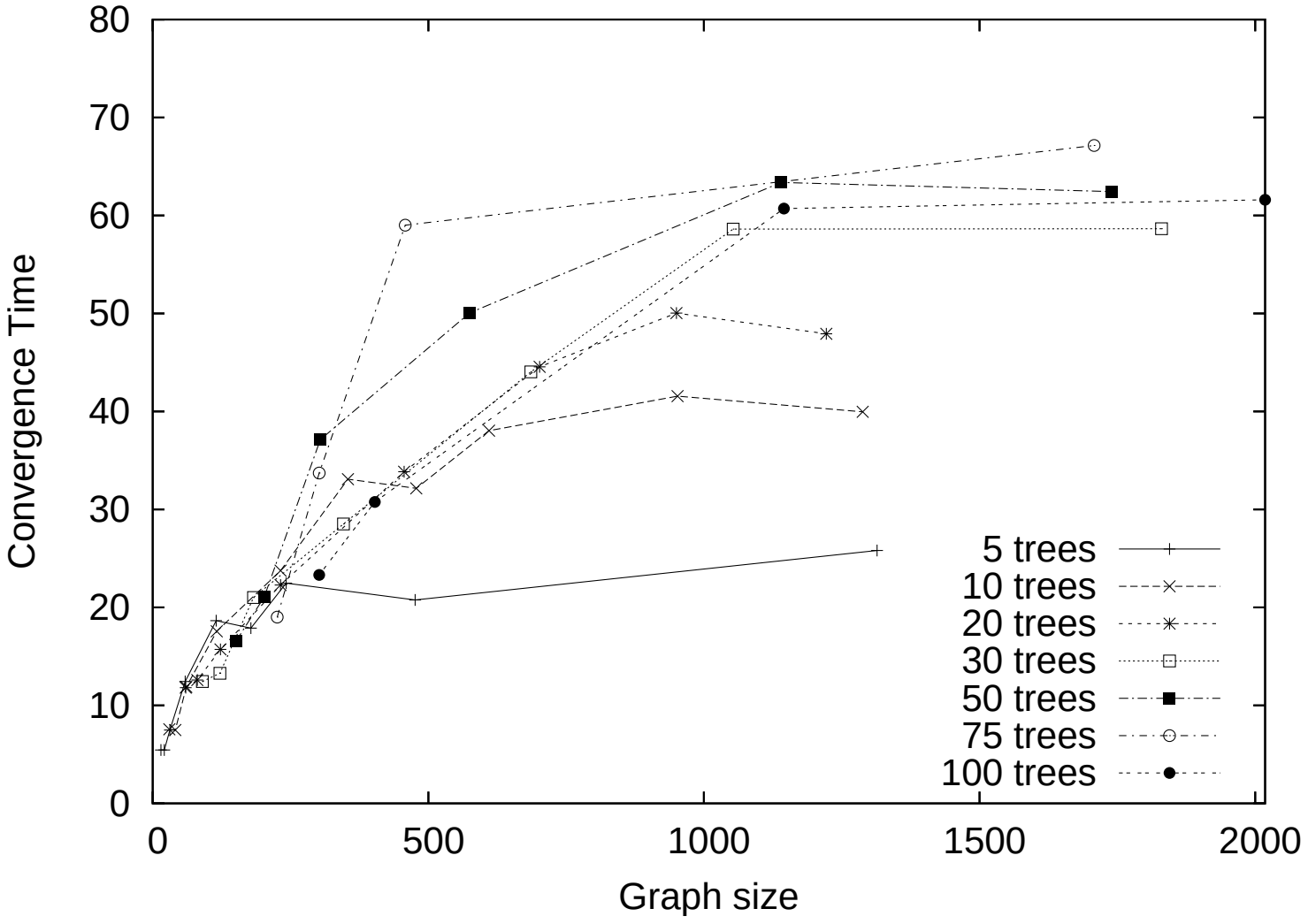


Sketch of proof

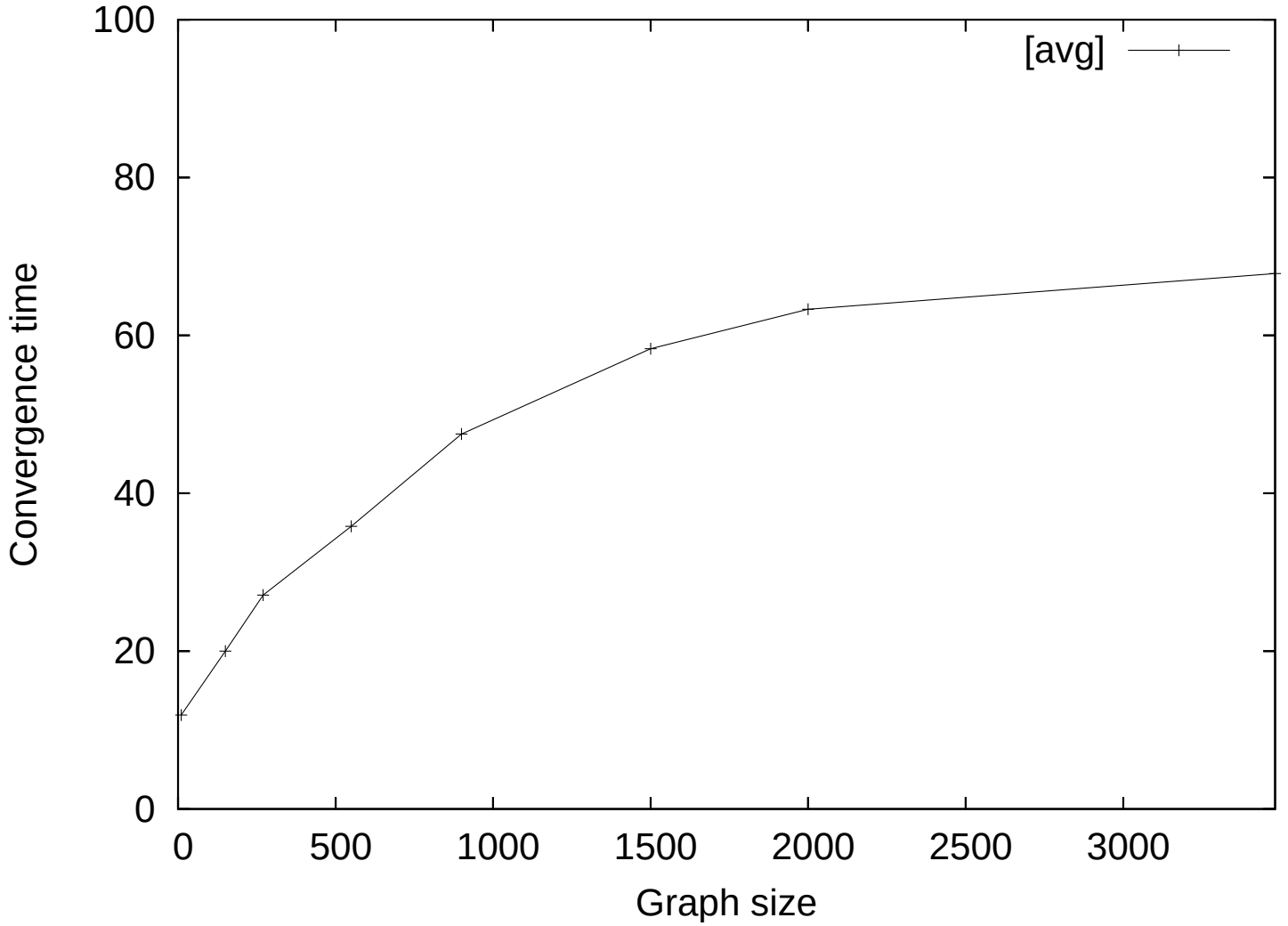
1. Stabilized communications (Awerbuch *et al.* 1996)
2. Only present processes
3. Parent's copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition



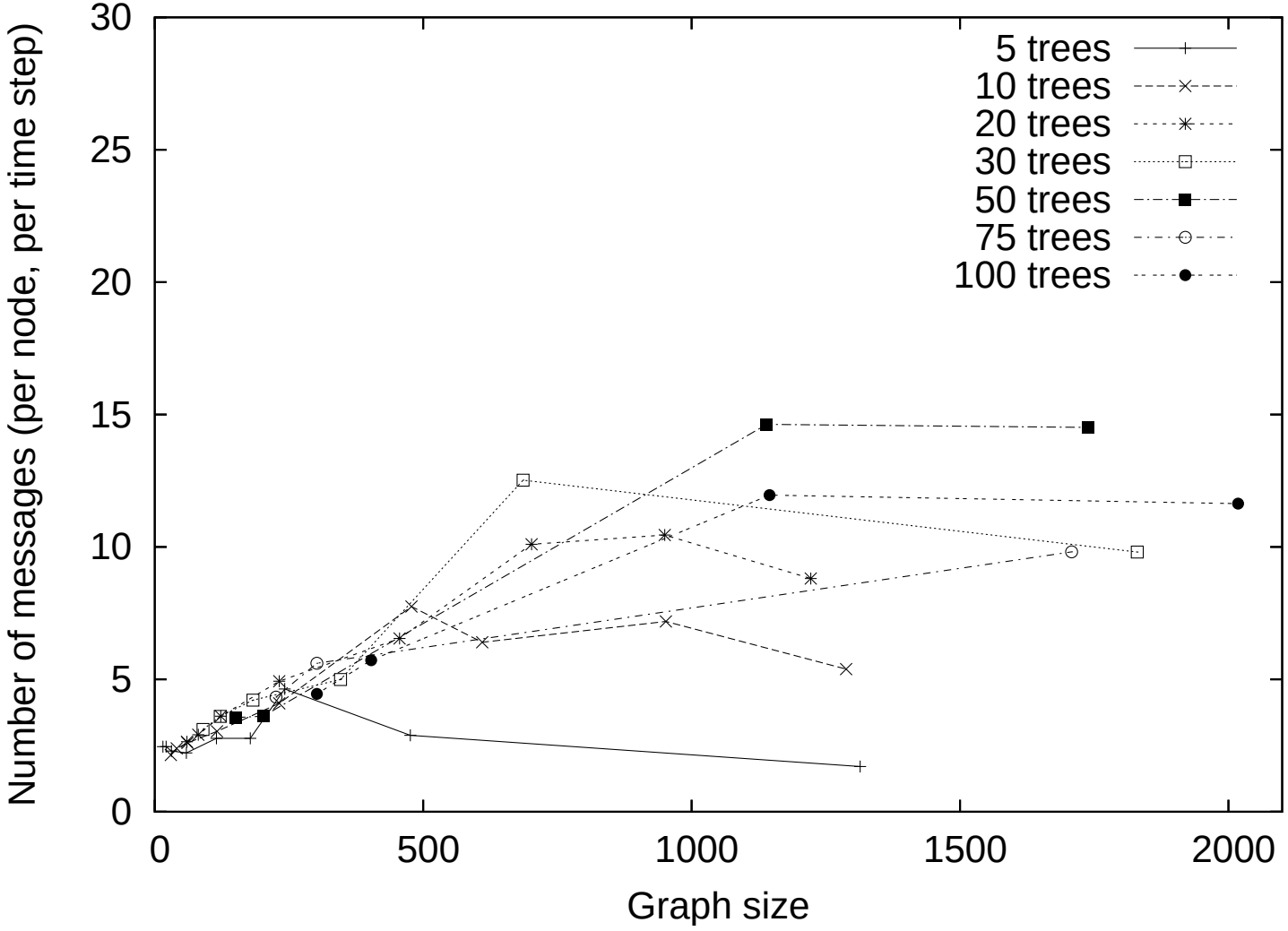
Simulation: Convergence Time



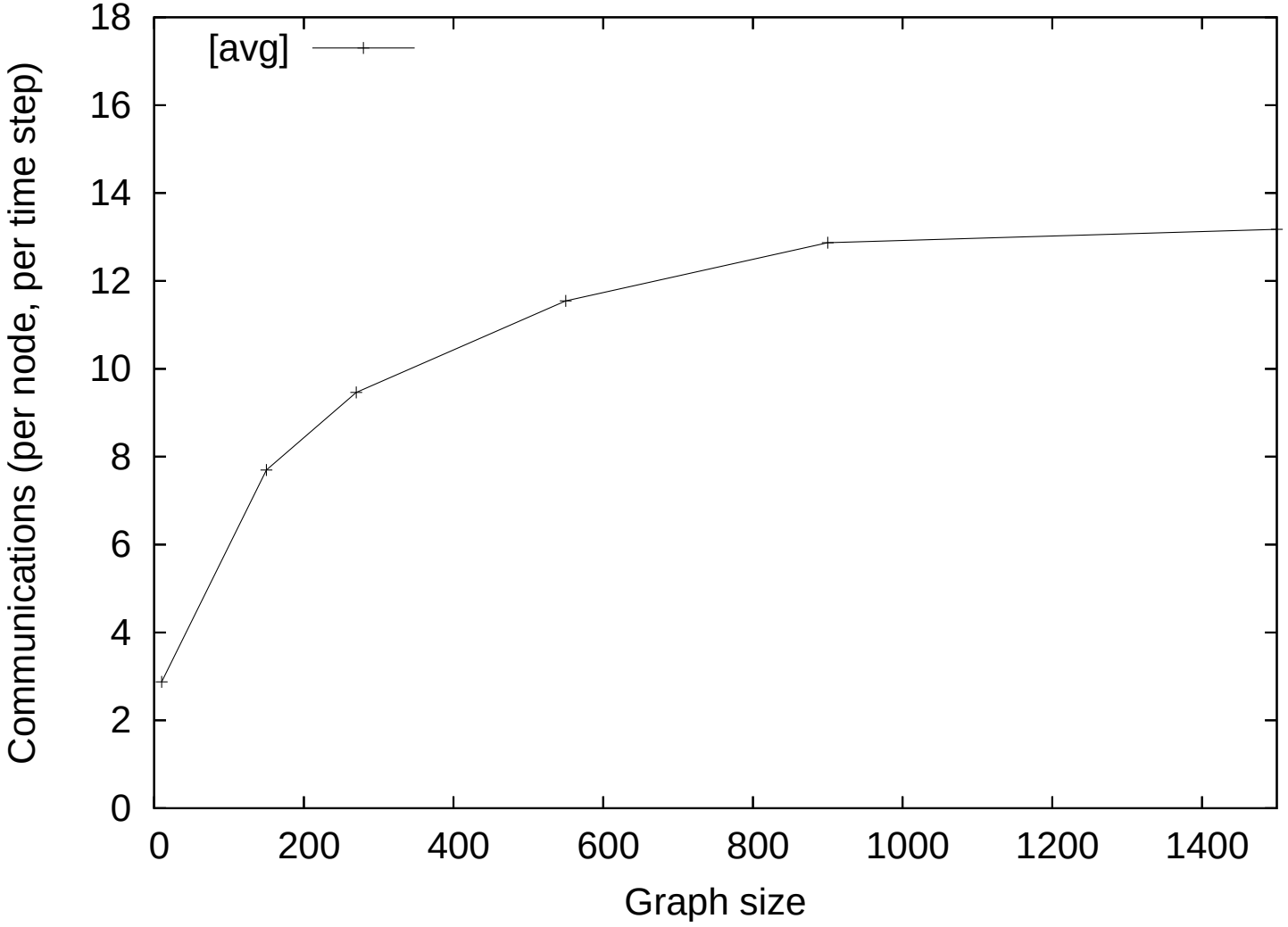
Simulation: Convergence Time



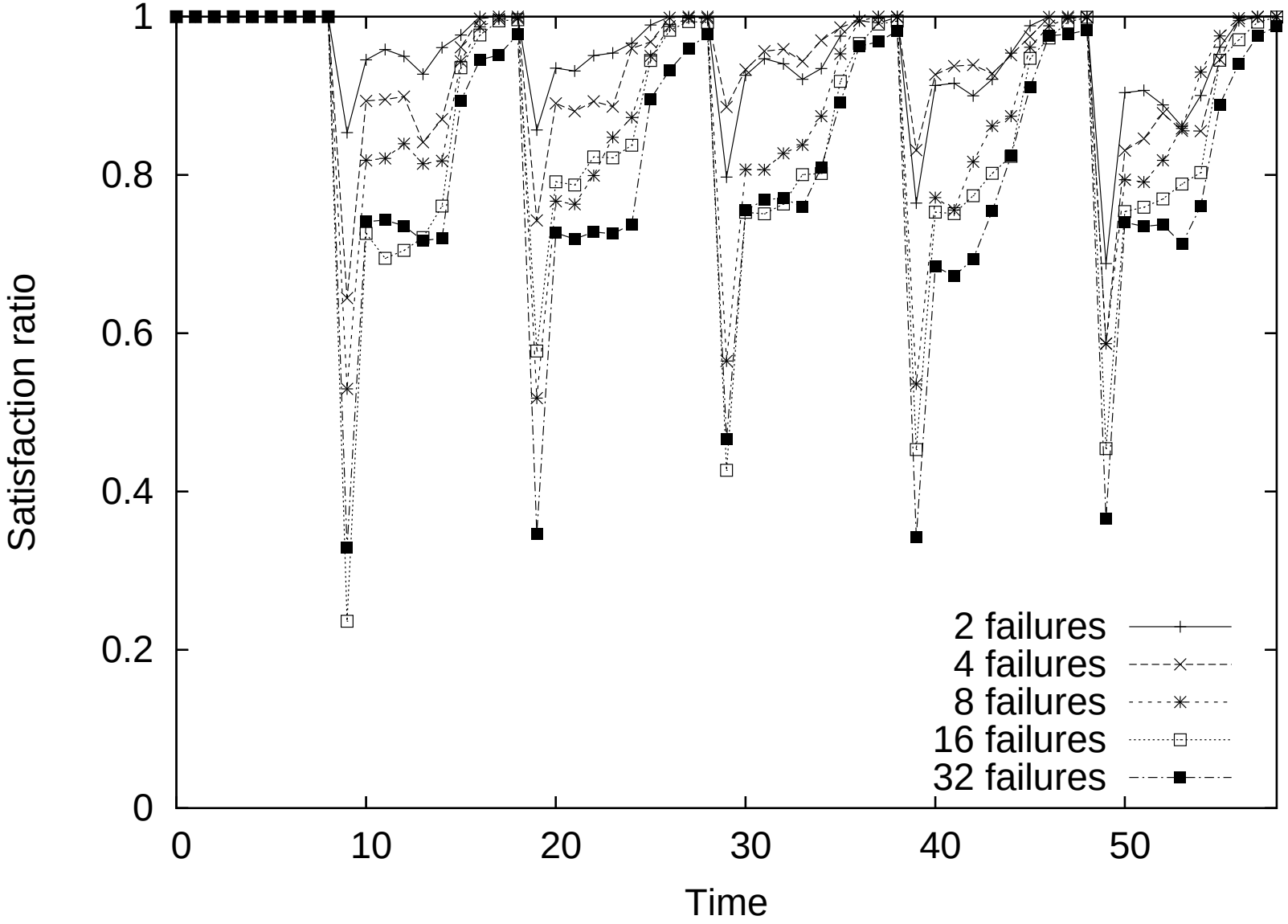
Simulation: Communication Amount



Simulation: Communication Amount



Simulation: Performance



Outline

Conclusion

- Contributions
  - *Practical* P2P prefix tree maintenance
  - Comprehensive correctness proof
  - Promising simulations for scalability
- On-going and future work
  - Better metrics of performance
  - Optimizations (cache, shortcuts)
  - Combination with redondancy
  - Implementation and deployment over a real platform

Conclusion

- Contributions
  - *Practical* P2P prefix tree maintenance
  - Comprehensive correctness proof
  - Promising simulations for scalability
- On-going and future work
  - Better metrics of performance
  - Optimizations (cache, shortcuts)
  - Combination with redondancy
  - Implementation and deployment over a real platform