RECONCILING MULTIPLE GENES TREES VIA SEGMENTAL DUPLICATIONS AND LOSSES

Riccardo Dondi[1], Manuel Lafond[2], Céline Scornavacca[3]

[1] Università degli Studdi di Bergamo, [2] Université de Sherbrooke, [3] ISEM, Université de Montpellier

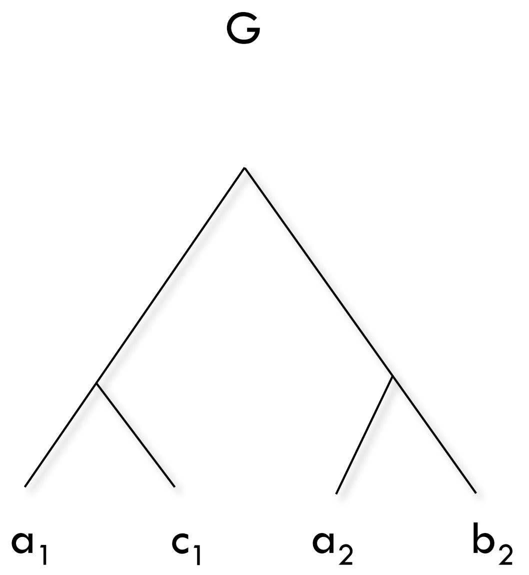
The plan

In this talk we…

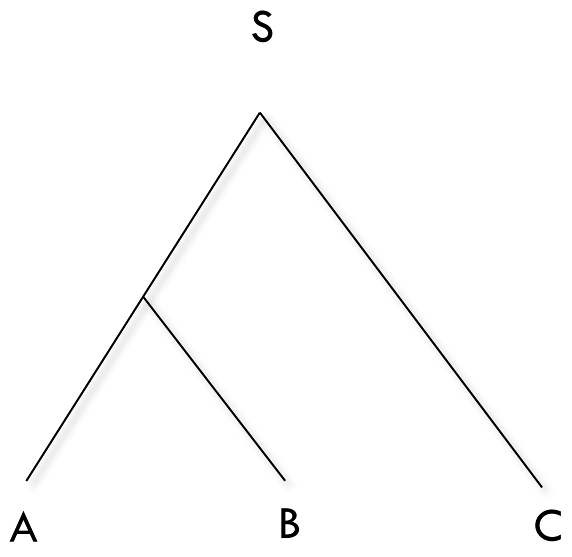
- …reconcile gene trees with species trees, but:
  - there are **many gene trees**, and
  - Duplications/losses can affect **several genes**.
- …detect whole genome duplications.
- …try to simulate genome evolution with segmental events.

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



Gene tree

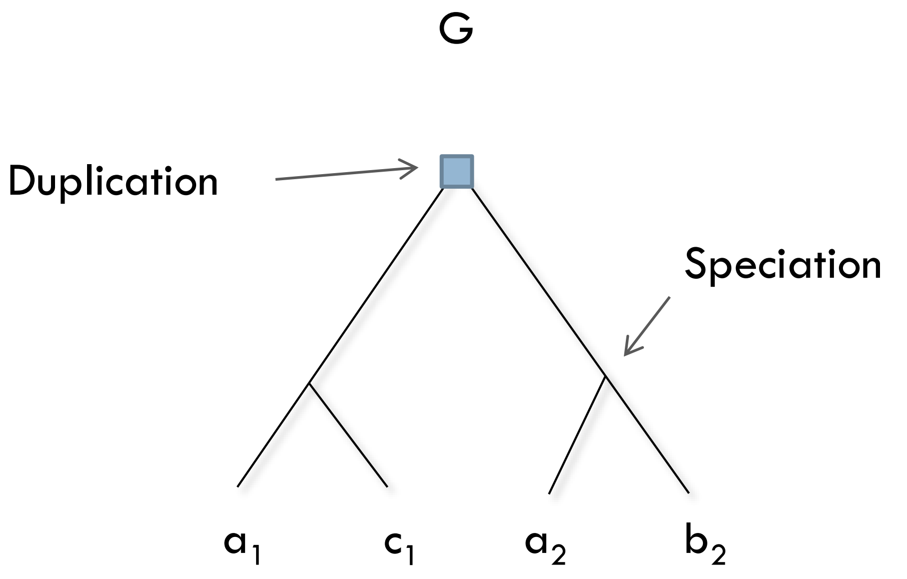


Species tree

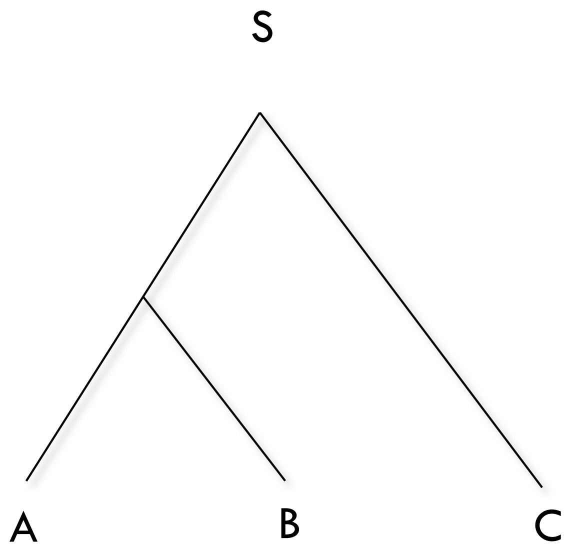
Notation tip: gene name = lowercase species

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



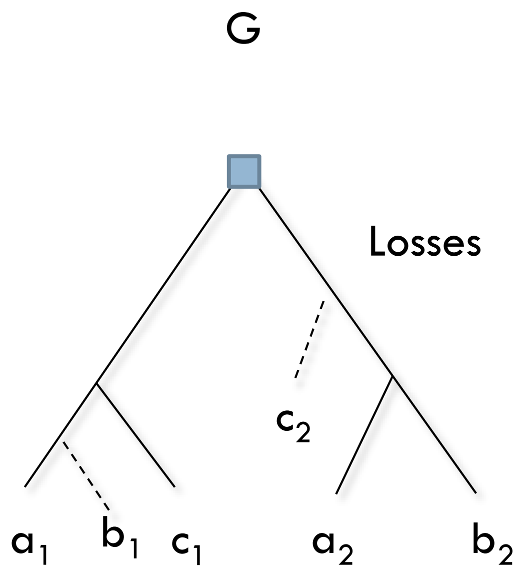
Gene tree



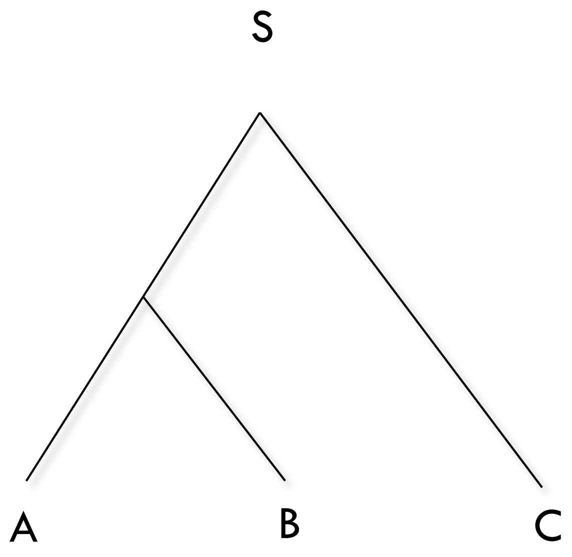
Species tree

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



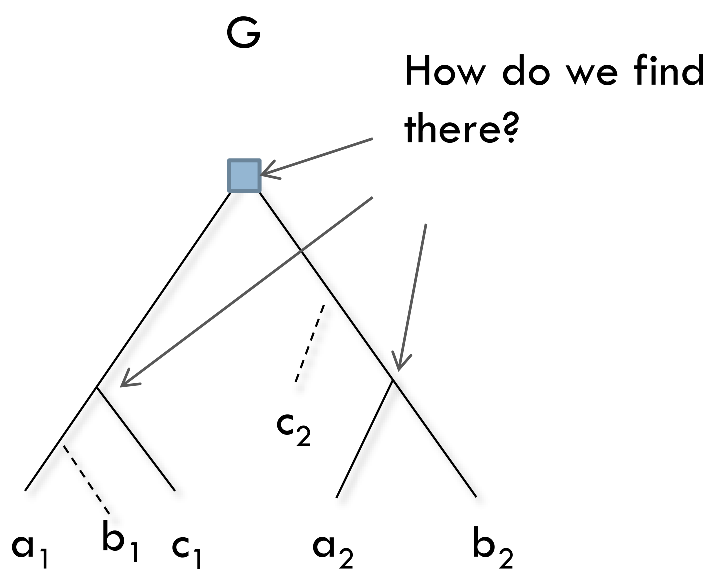
Gene tree



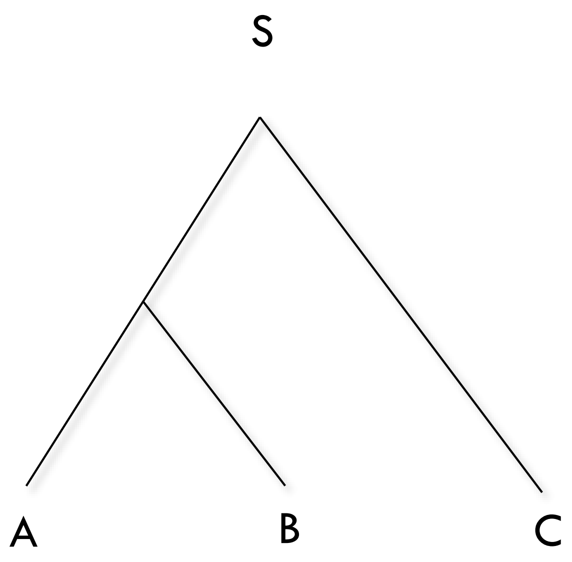
Species tree

Reconciliation

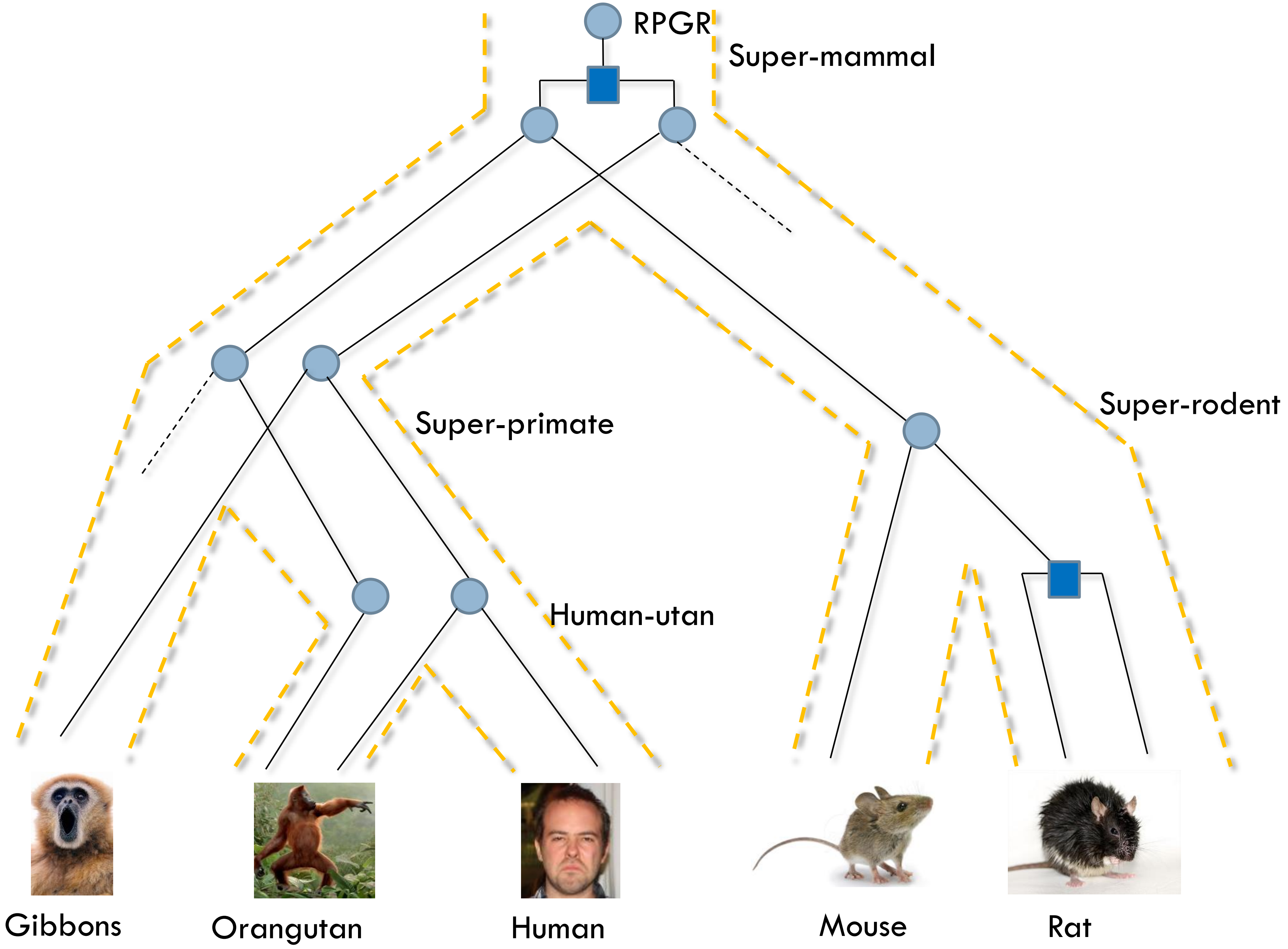
Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



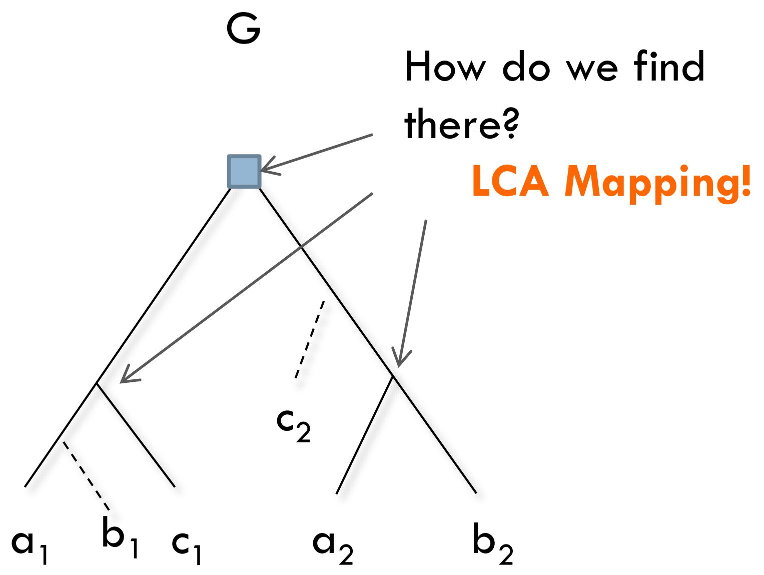
Gene tree



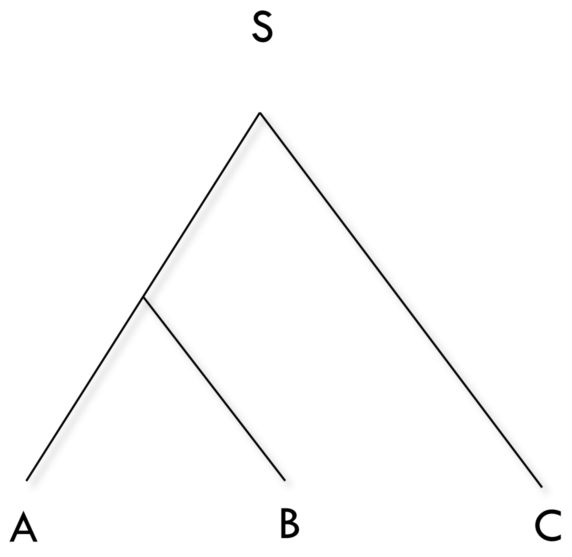
Species tree


RPGR
Super-mammal
Super-primate
Human-utan
Super-rodent
Gibbons
Orangutan
Human
Mouse
Rat

LCA Mapping



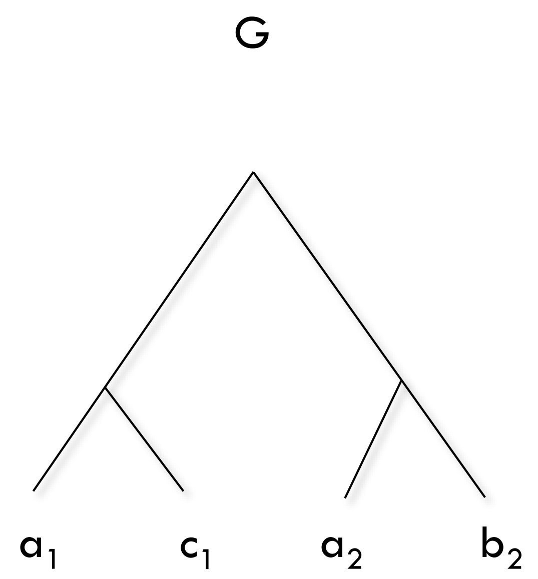
Gene tree



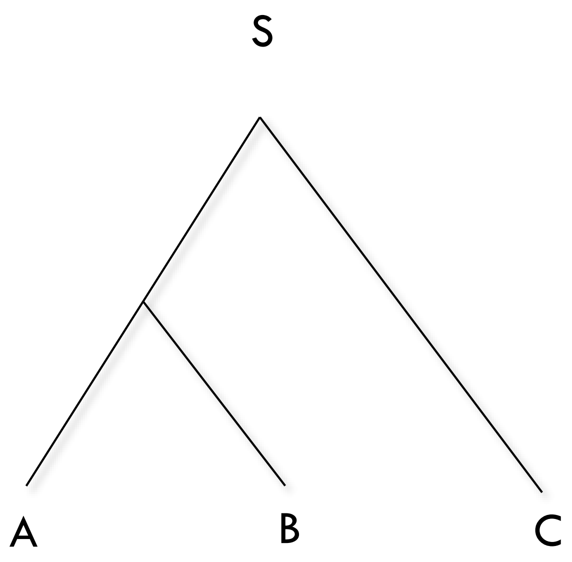
Species tree

LCA Mapping

Map each ancestral gene to the **species** that is the **lowest common ancestor (LCA)** of the descending mapped species.



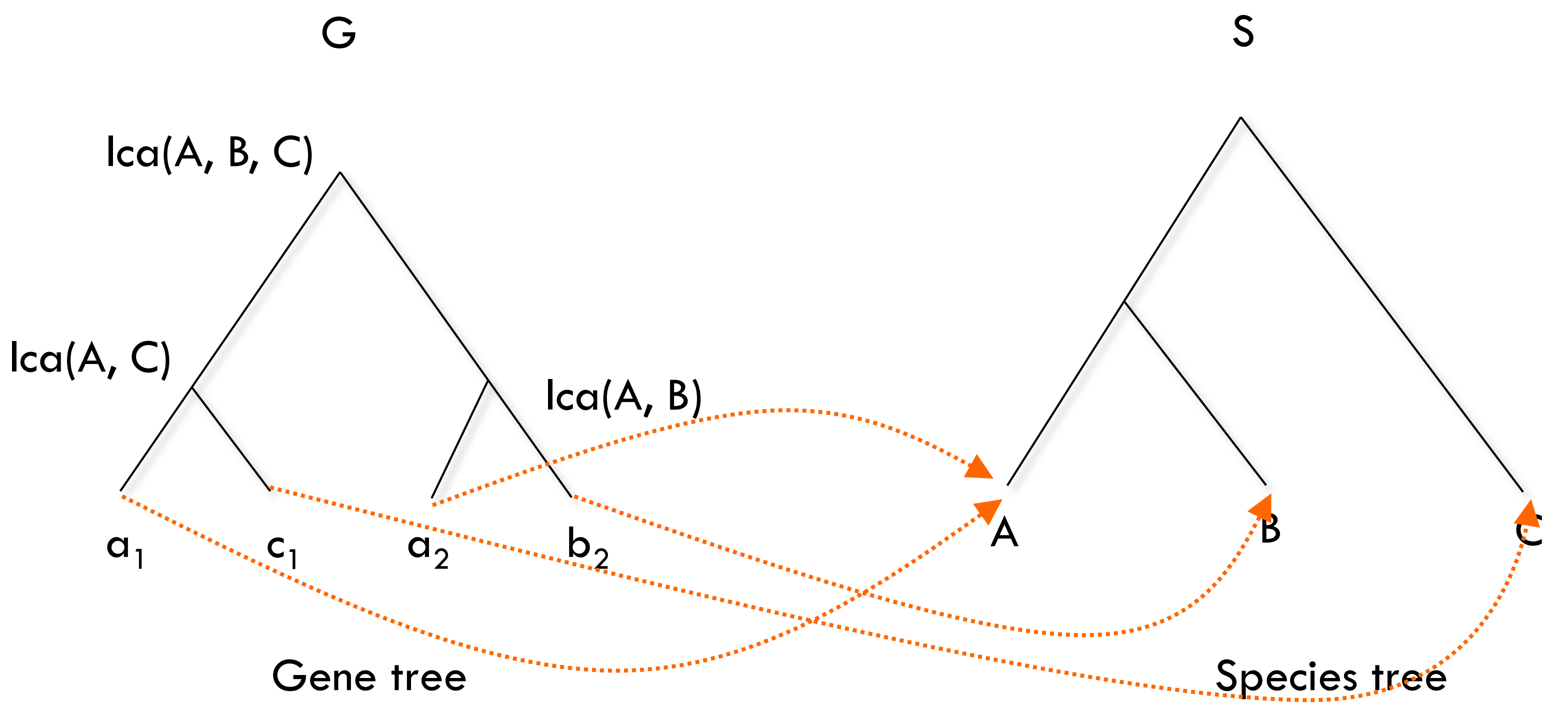
Gene tree



Species tree

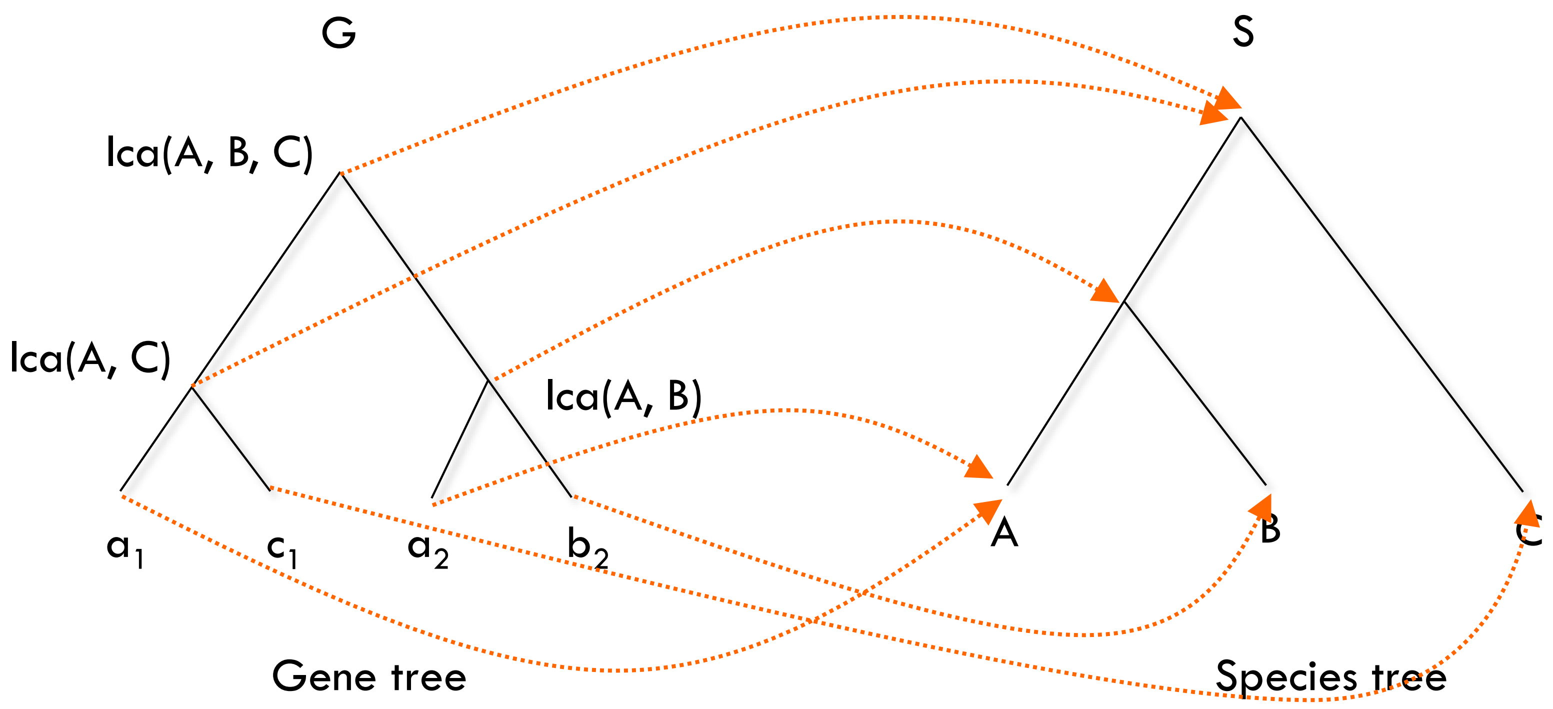
LCA Mapping

Map each ancestral gene to the **species** that is the **lowest common ancestor (LCA)** of the descending mapped species.



LCA Mapping

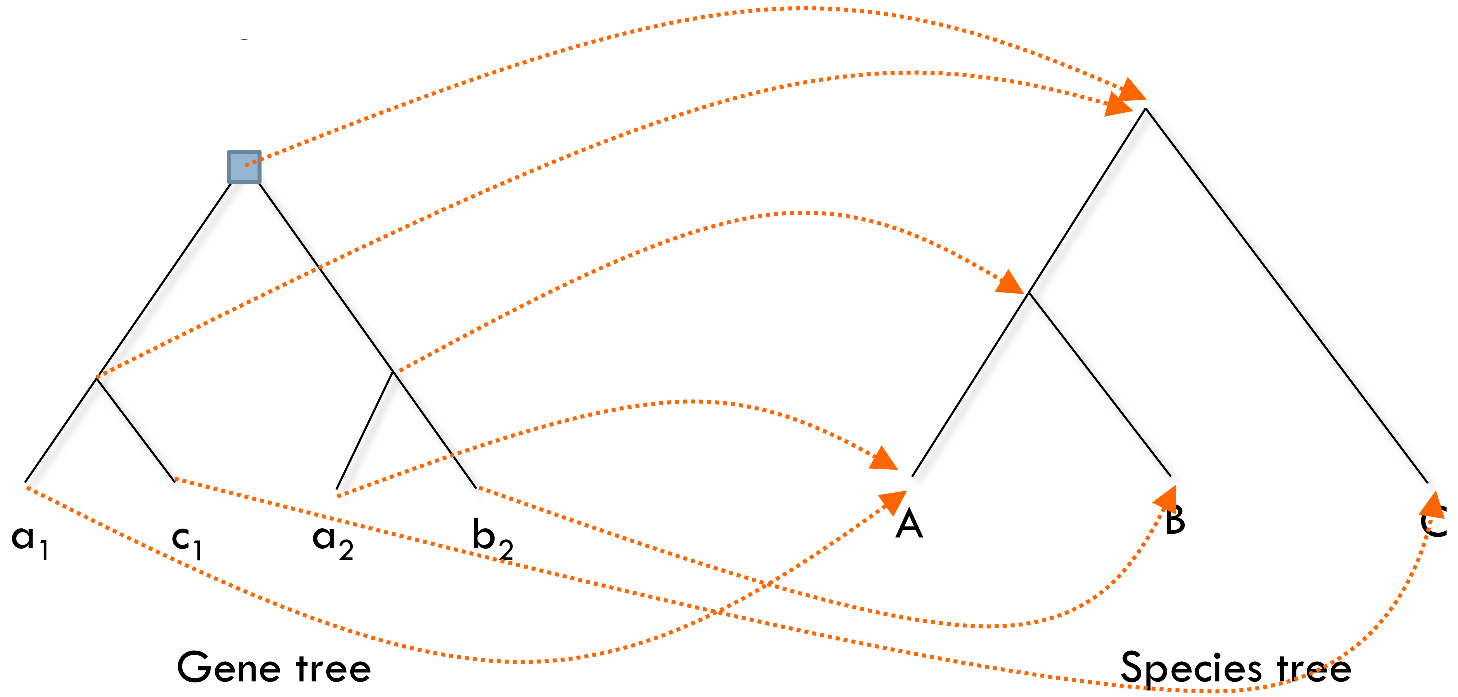
Map each ancestral gene to the **species** that is the **lowest common ancestor (LCA)** of the descending mapped species.



LCA Mapping

Map each ancestral gene to the **species** that is the **lowest common ancestor (LCA)** of the descending mapped species.

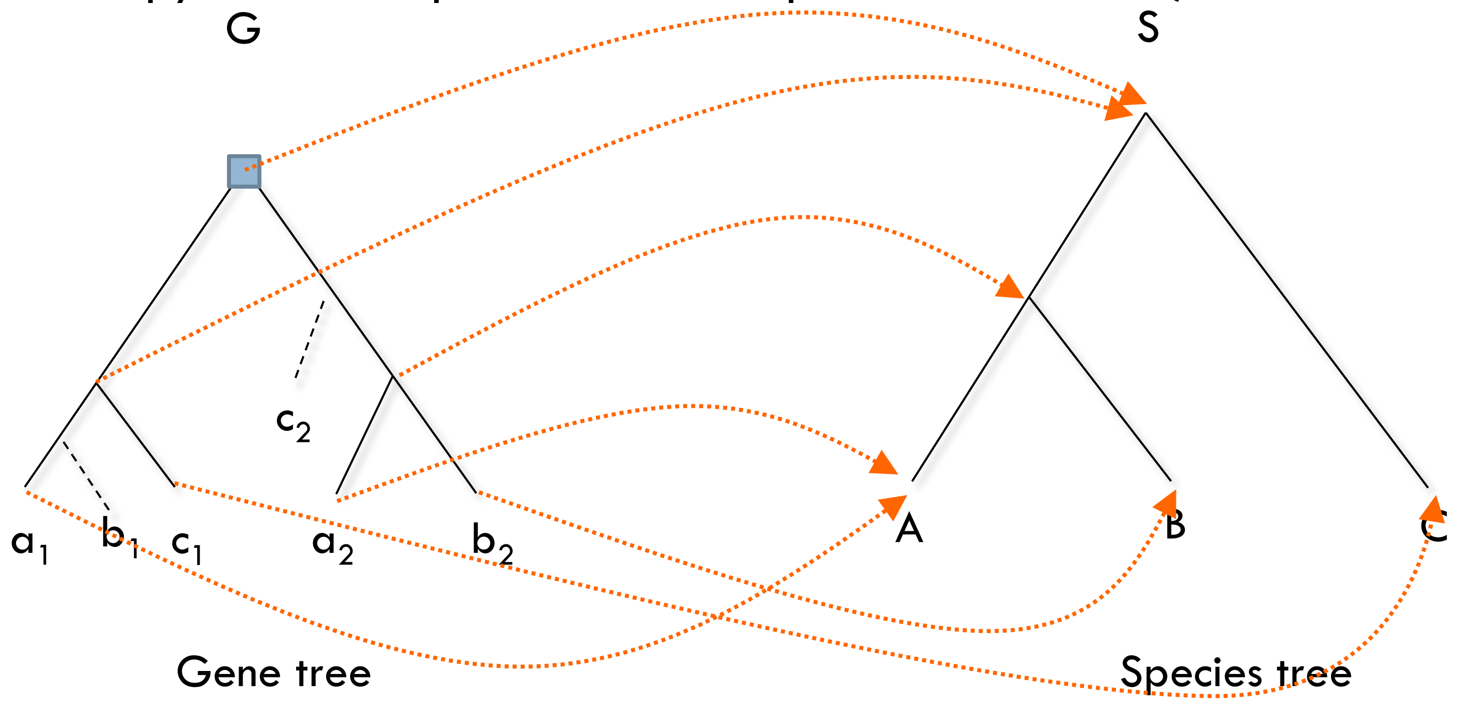
- **Rule**: a node of G **must be a Dup** if it maps to the same species as a child.



LCA Mapping

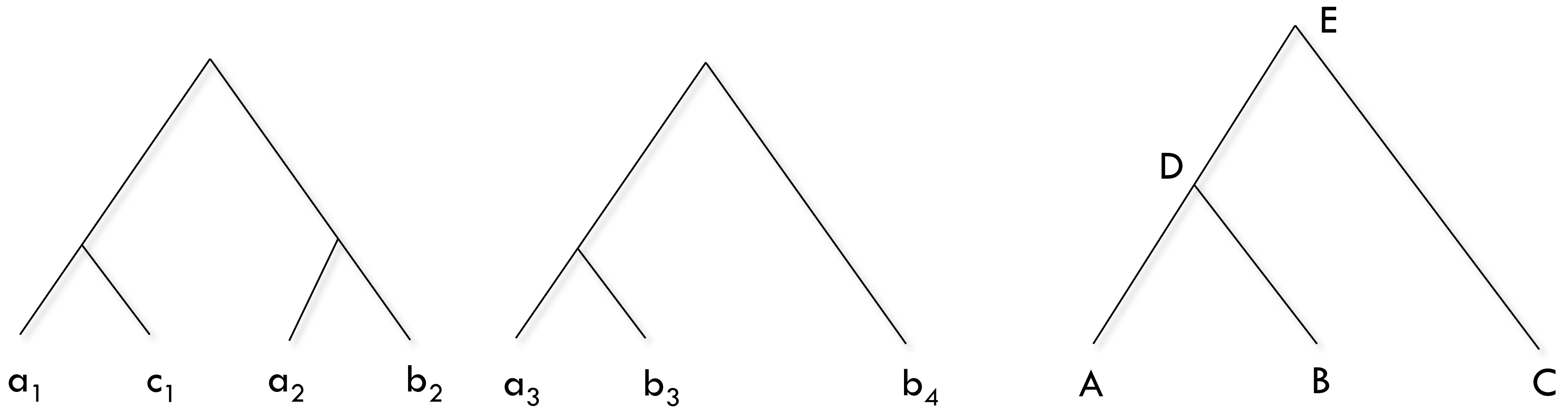
Map each ancestral gene to the **species** that is the **lowest common ancestor (LCA)** of the descending mapped species.

- **Rule**: a node of G **must be a Dup** if it maps to the same species as a child.
- Each copy should be present in each species – otherwise, losses.



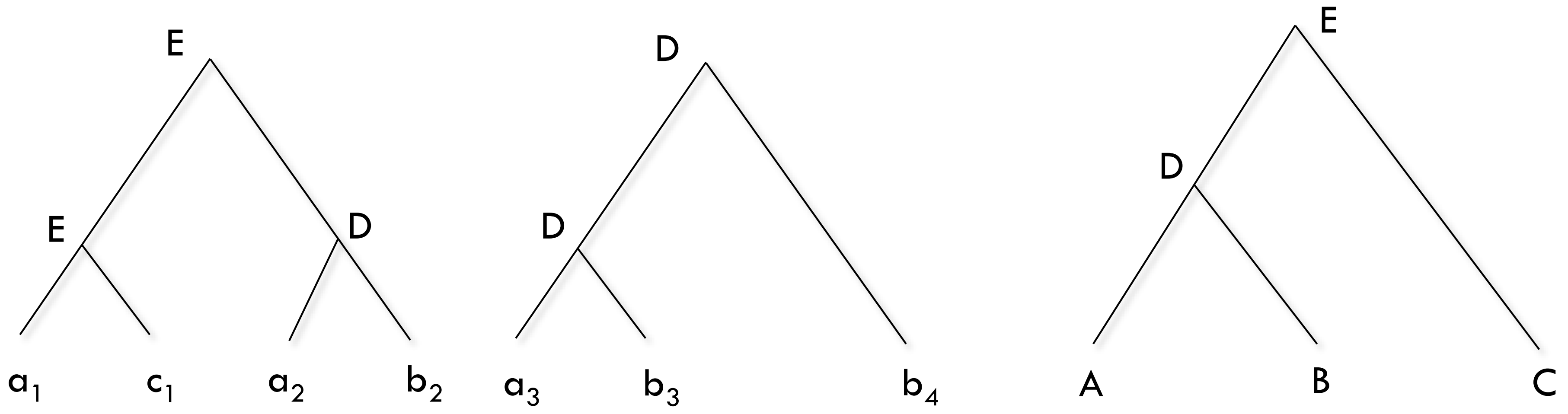
LCA Mapping

Now let's have more than one gene tree.



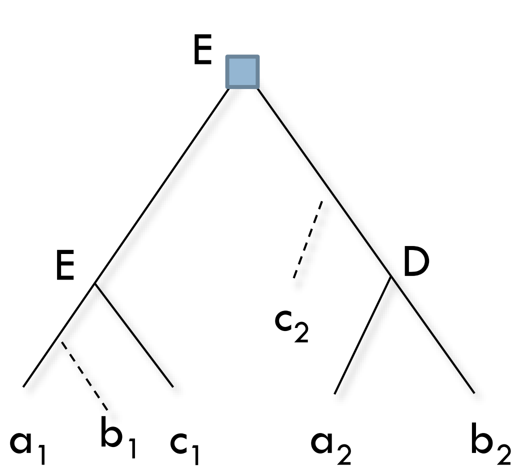
LCA Mapping

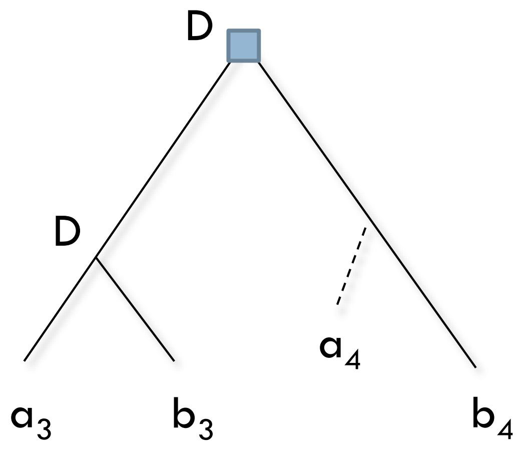
Now let's have more than one gene tree.

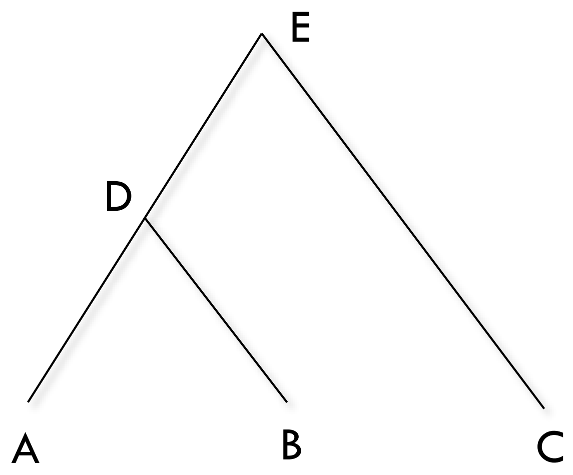


LCA Mapping

Now let's have more than one gene tree.



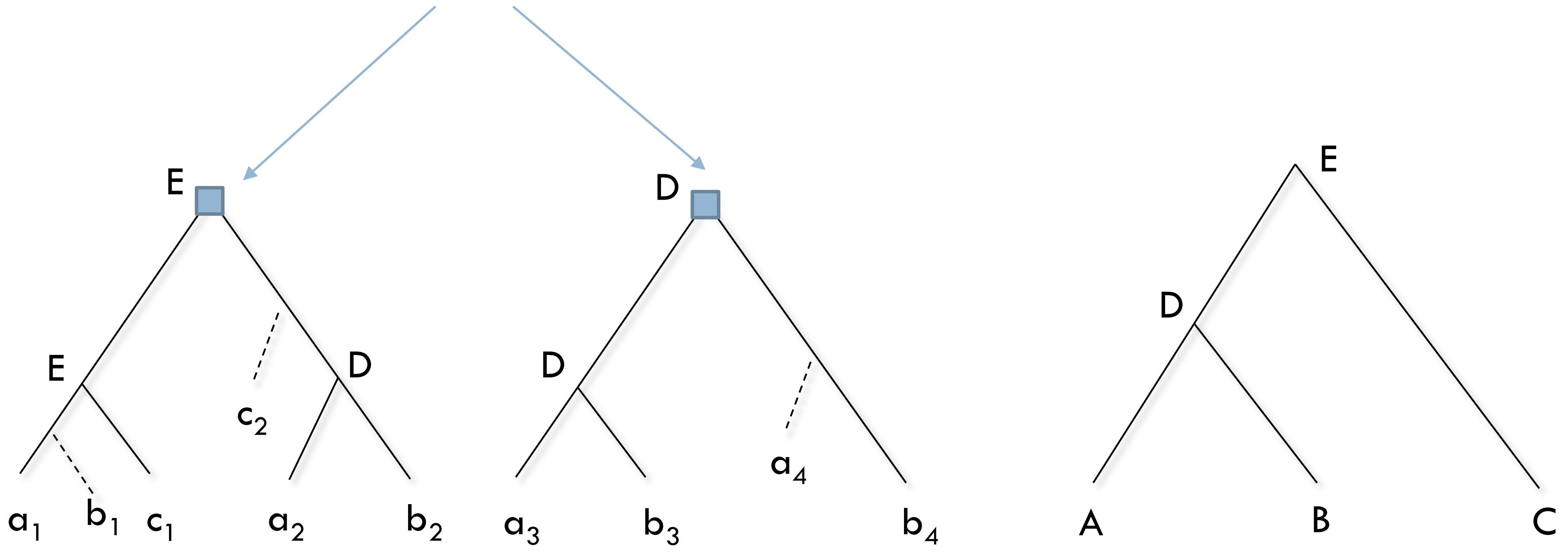




LCA Mapping

Now let's have more than one gene tree.

Maybe these duplications **are the same**! (e.g. a block duplication of a segment)

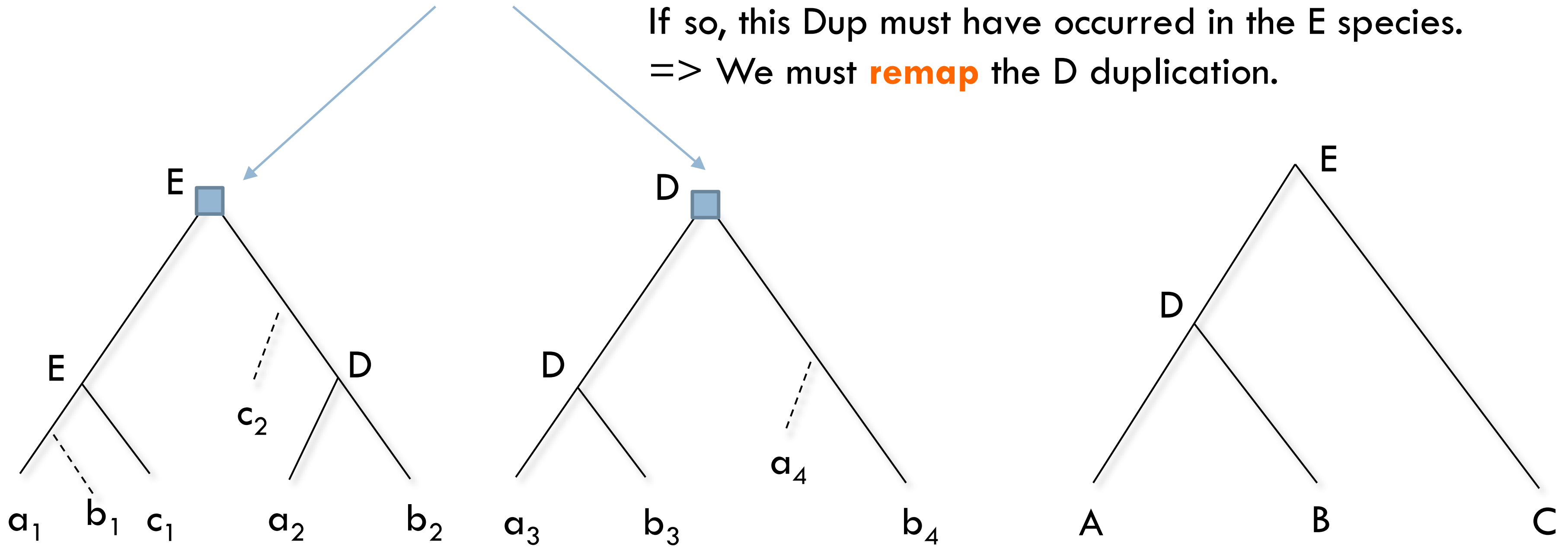


LCA Mapping

Now let's have more than one gene tree.

Maybe these duplications **are the same**! (e.g. a block duplication of a segment)

If so, this Dup must have occurred in the E species.
=> We must **remap** the D duplication.

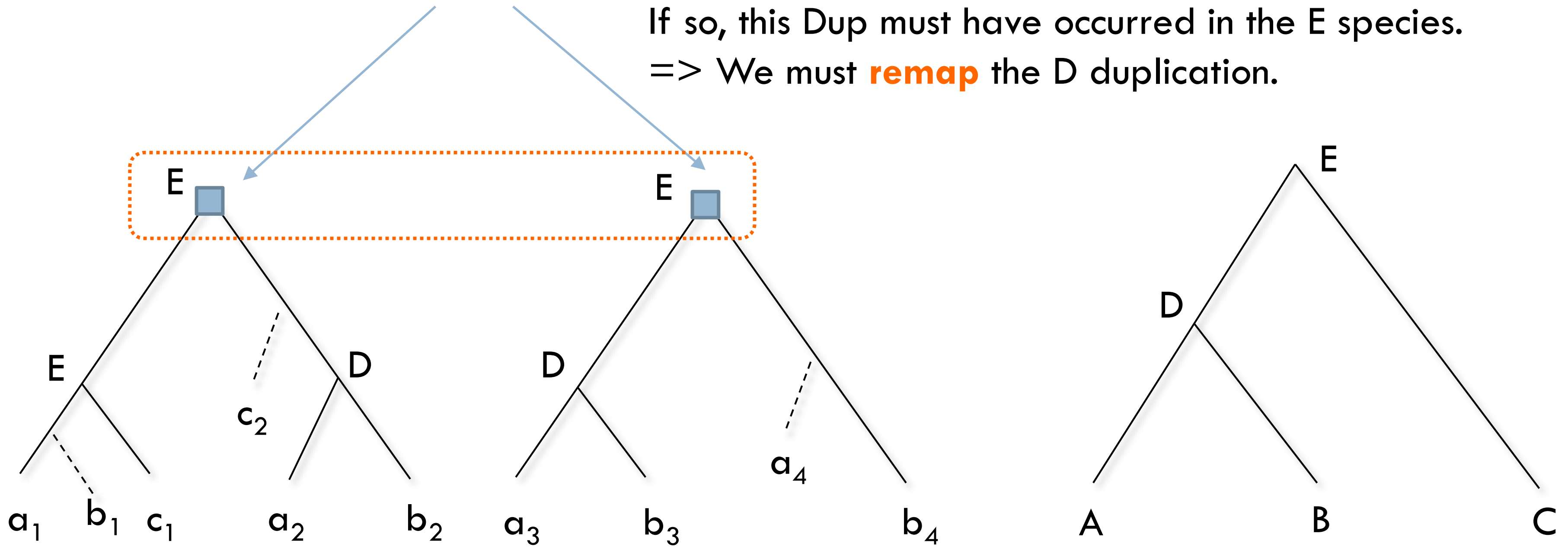


LCA Mapping

Now let’s have more than one gene tree.

Maybe these duplications **are the same**! (e.g. a block duplication of a segment)

If so, this Dup must have occurred in the E species.
=> We must **remap** the D duplication.

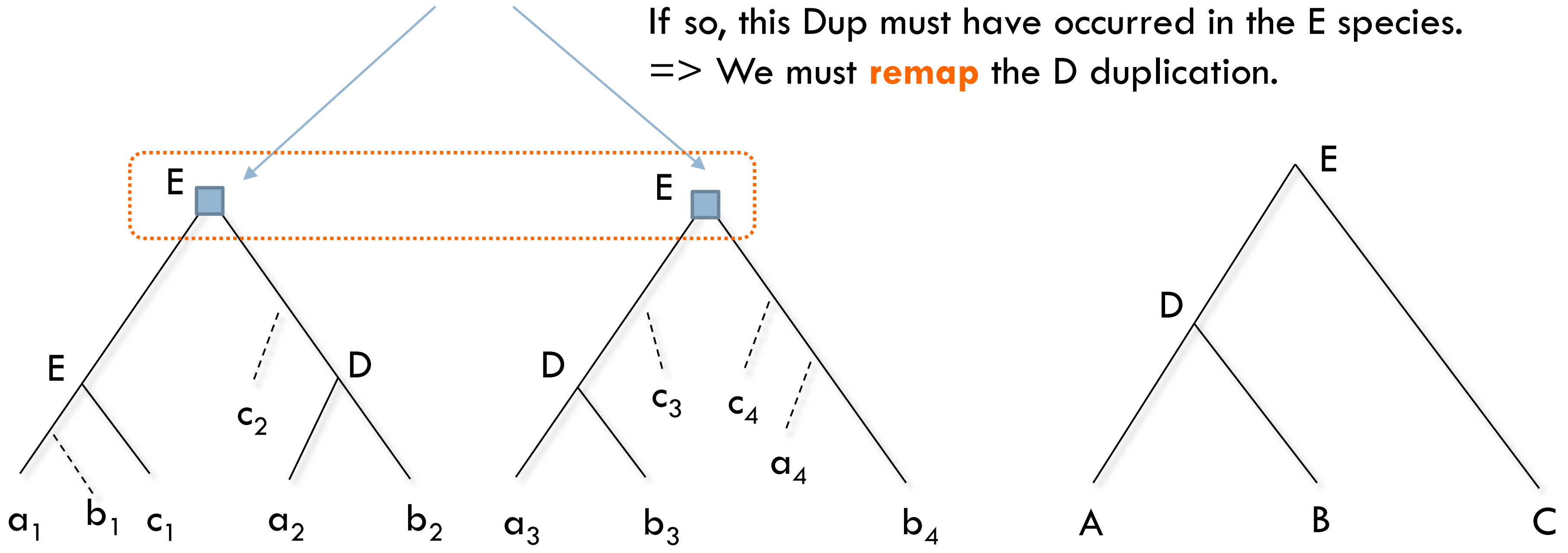


LCA Mapping

Now let's have more than one gene tree.

Maybe these duplications **are the same**! (e.g. a block duplication of a segment)

If so, this Dup must have occurred in the E species.
=> We must **remap** the D duplication.



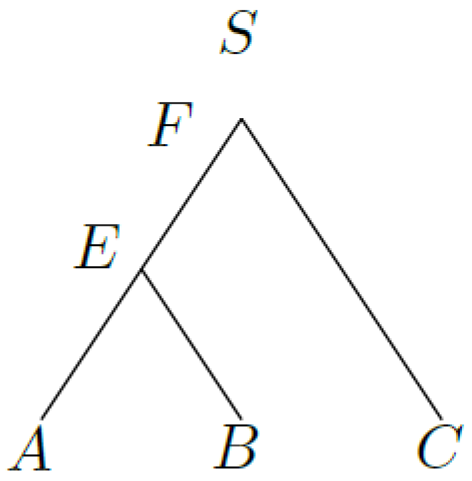
1 DUP, 5 LOSSES (before, we had 2 DUPS, 3 LOSSES)

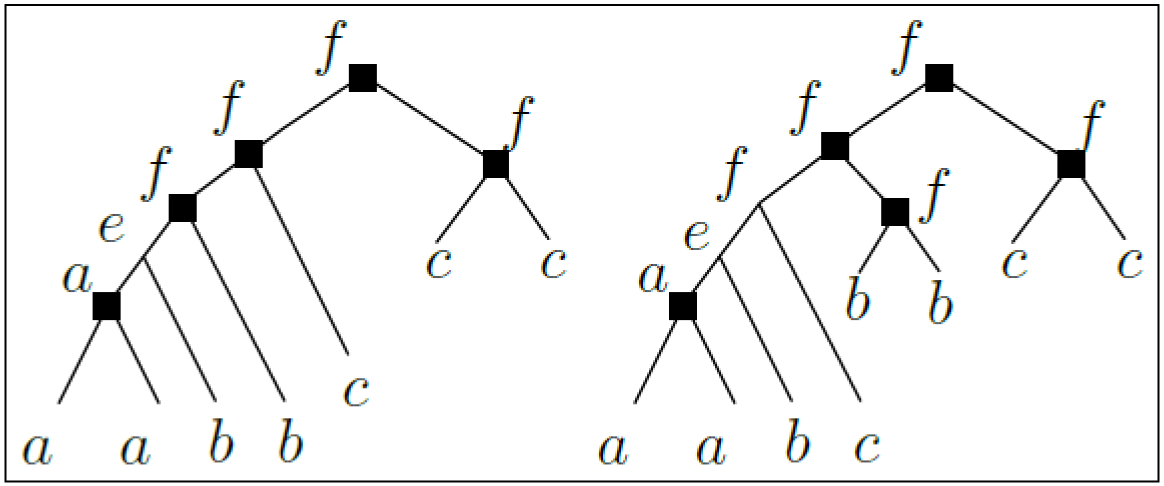
Reconciling with segmental Dups

- If we know the mapping, computing the number of **segmental Dups** is easy.
- **Losses** are also easy to compute.
- **Challenge**: find the best mapping.

Reconciling with segmental Dups

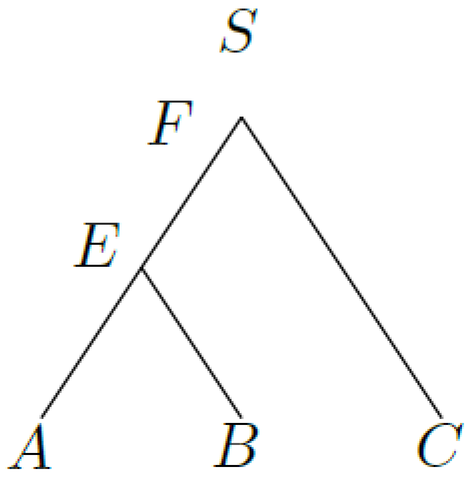
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?

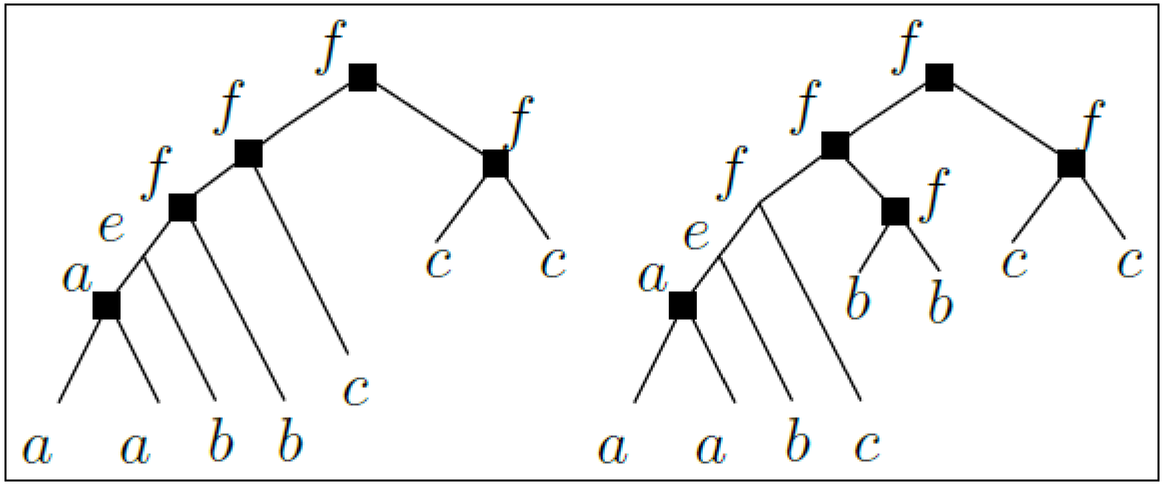




Reconciling with segmental Dups

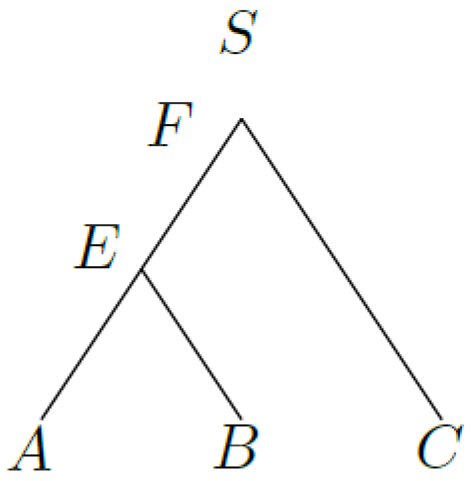
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

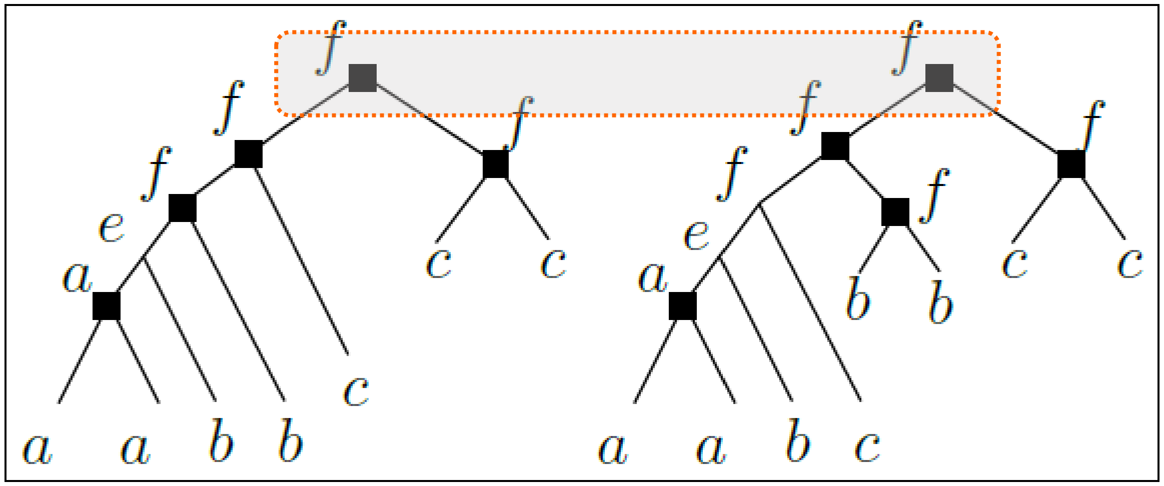




Reconciling with segmental Dups

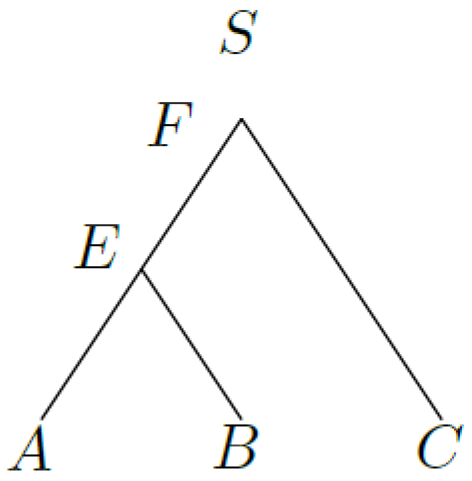
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

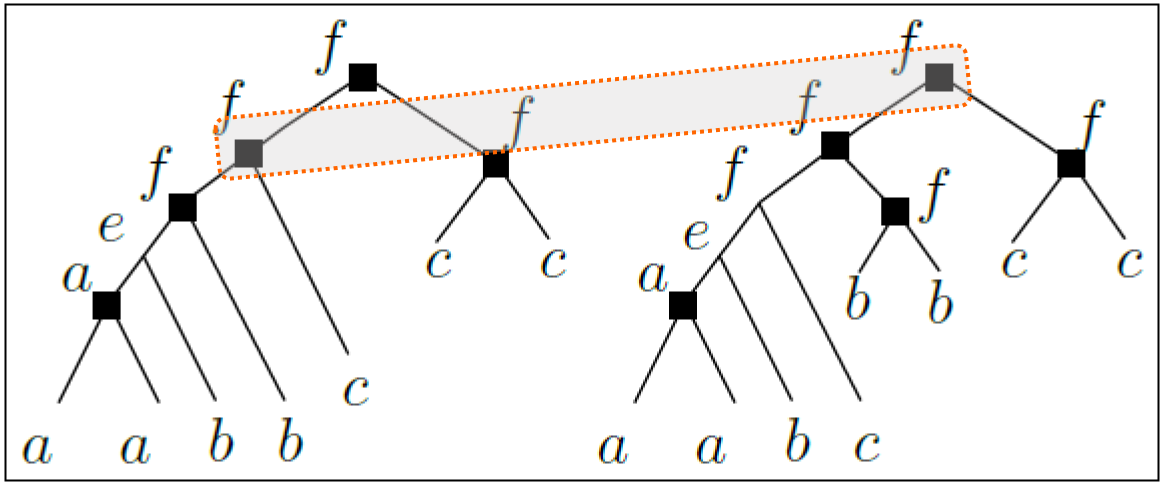




Reconciling with segmental Dups

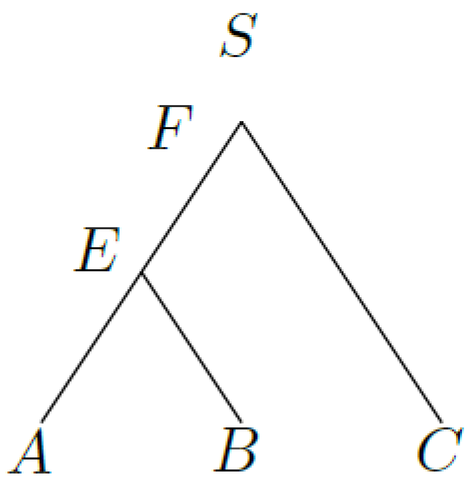
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

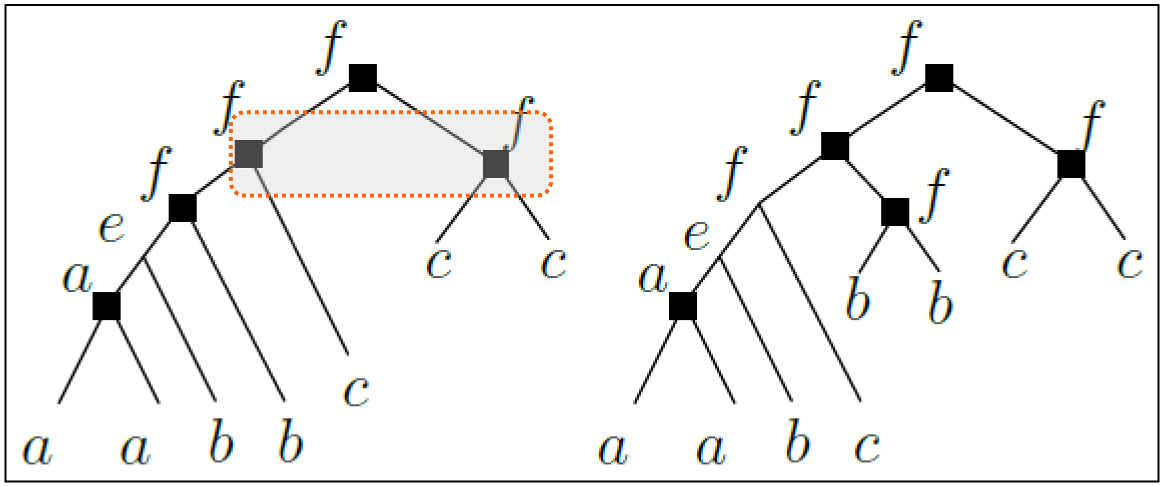




Reconciling with segmental Dups

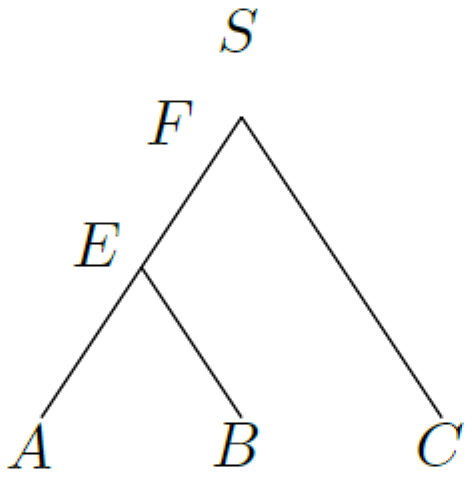
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

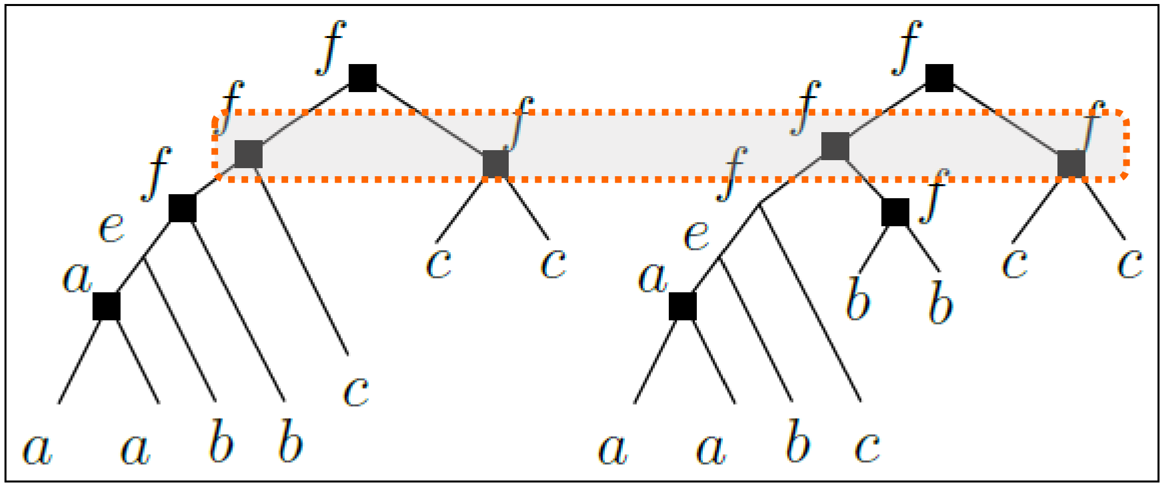




Reconciling with segmental Dups

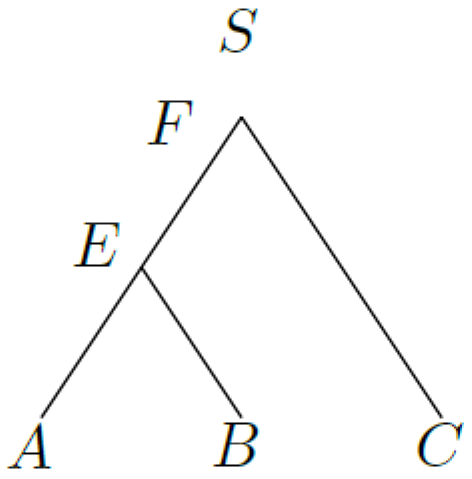
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

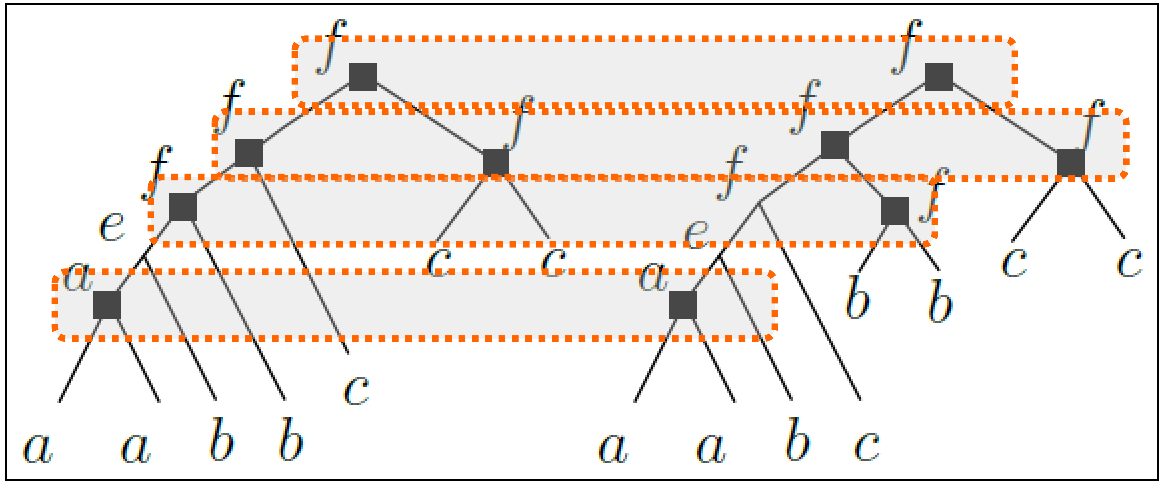




Reconciling with segmental Dups

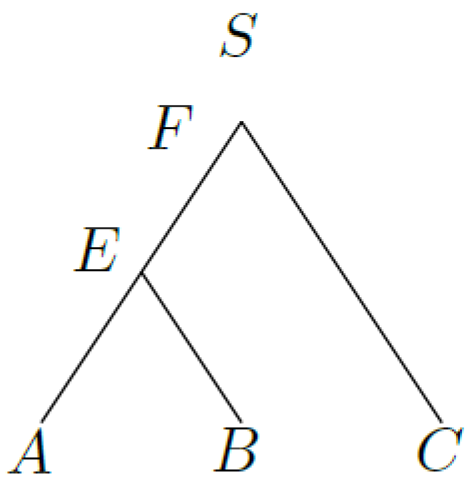
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

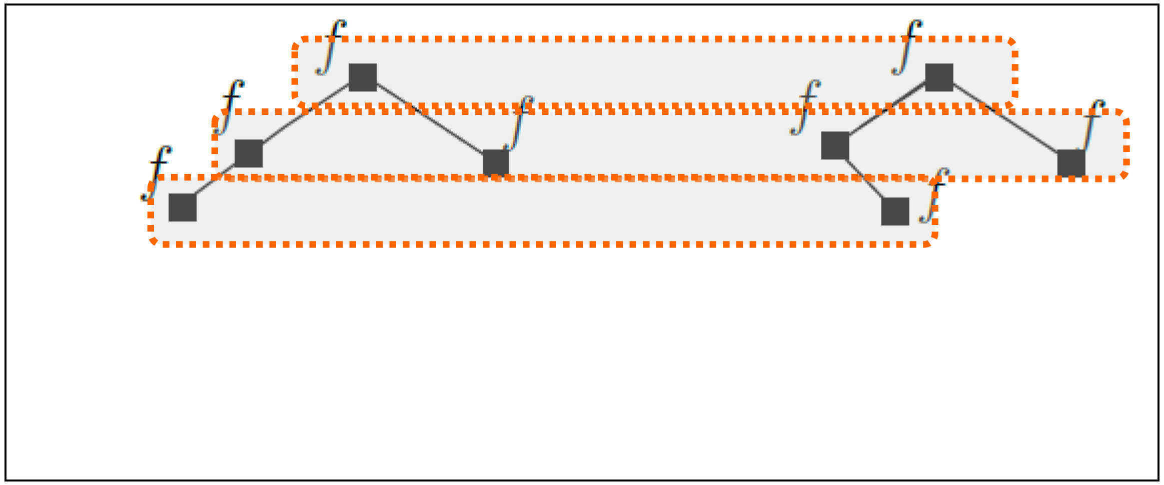




Reconciling with segmental Dups

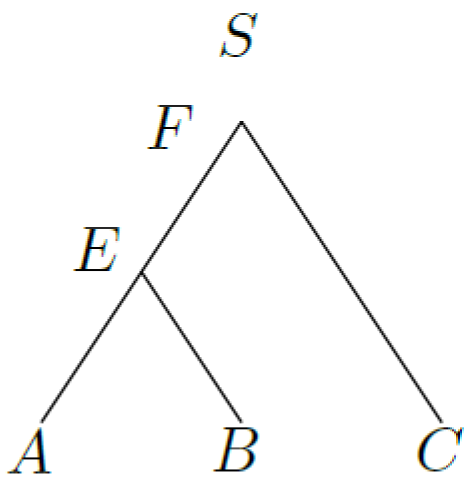
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »

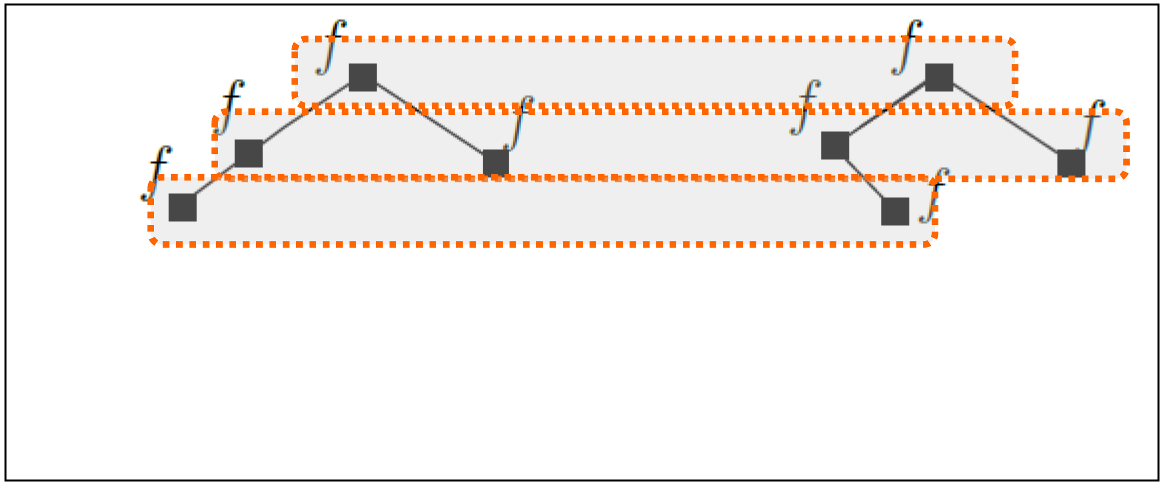




Reconciling with segmental Dups

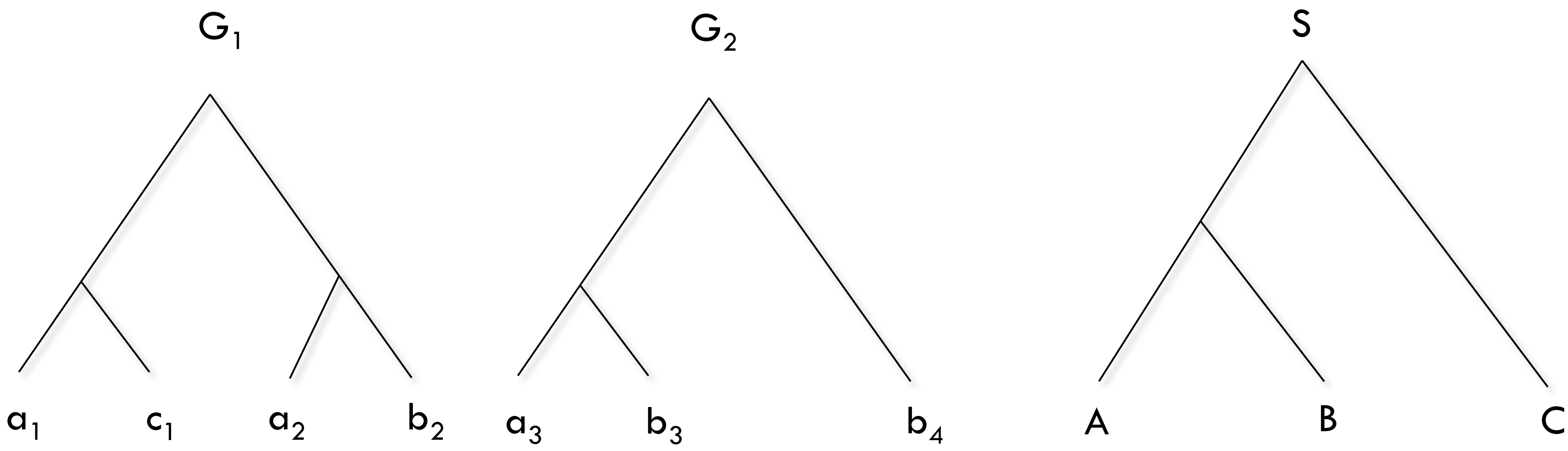
- **Question**: given a fixed mapping, how do we minimize the number of segmental Dups?
  - Any two Dups **unrelated by ancestry** + **mapped to the same species** could potentially be « the same »
  - **# segmental Dups in *f* = height of *f* forest**





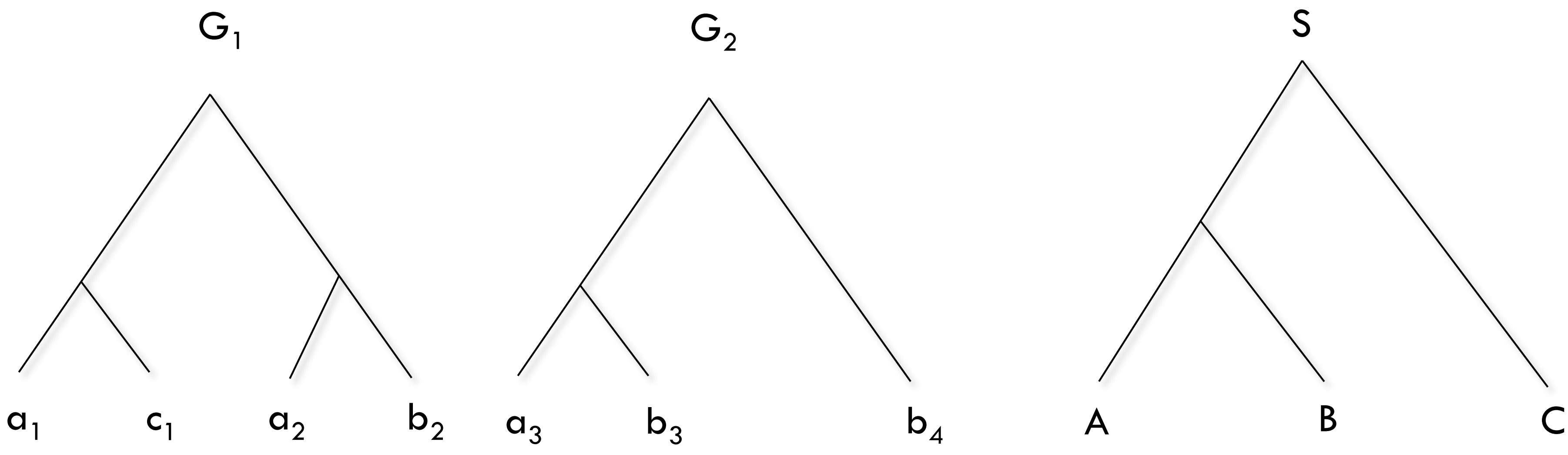
Reconciling with segmental Dups

- **Given:** a set of gene trees $G = \{G_1, \ldots G_k\}$ and a species tree S
- **Find:** a mapping of the nodes of G that minimizes:
  - the sum of Dup heights.
  - the sum of Dup heights + the number of losses.



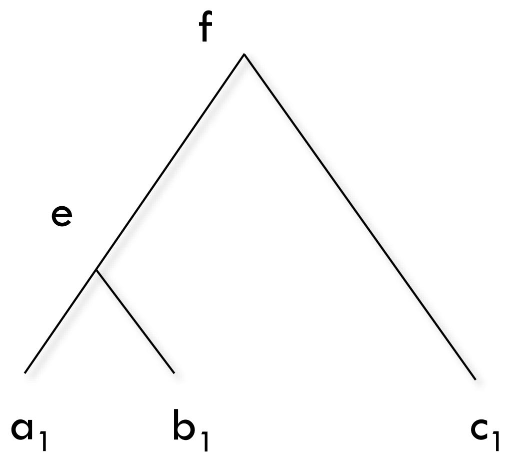
Reconciling with segmental Dups

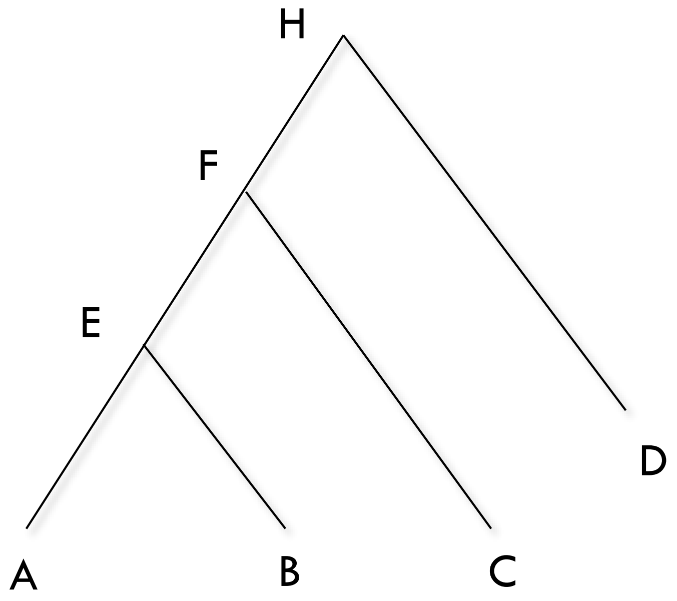
- **Given:** a set of gene trees $G = \{G_1, \ldots G_k\}$ and a species tree S
- **Find:** a mapping of the nodes of G that minimizes:
  - the sum of Dup heights.
  - δ * (sum of Dup heights) + λ * (number of losses)



Reconciling with segmental Dups

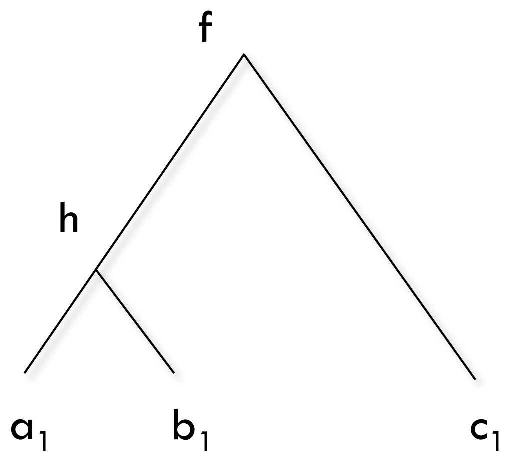
- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.

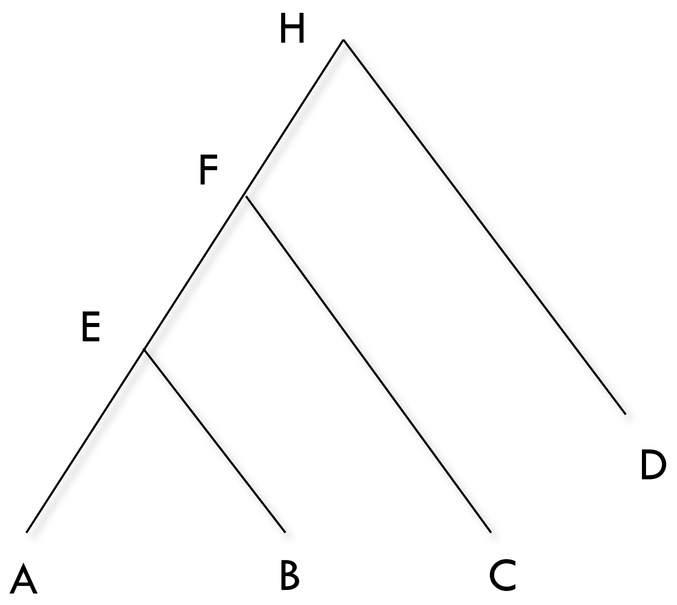




Reconciling with segmental Dups

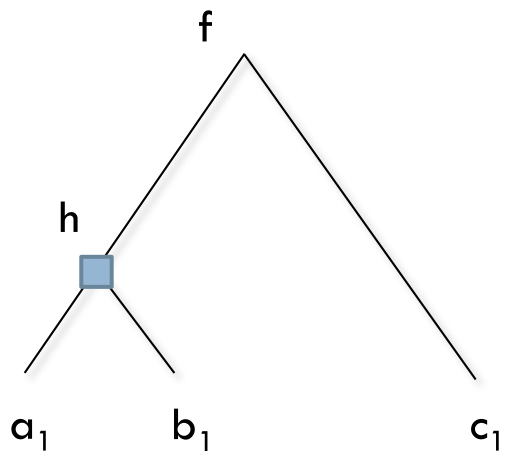
- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.

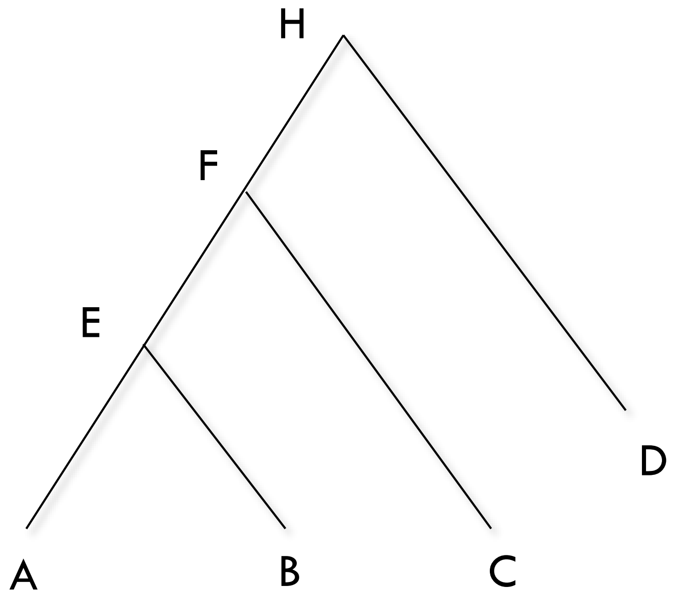




Reconciling with segmental Dups

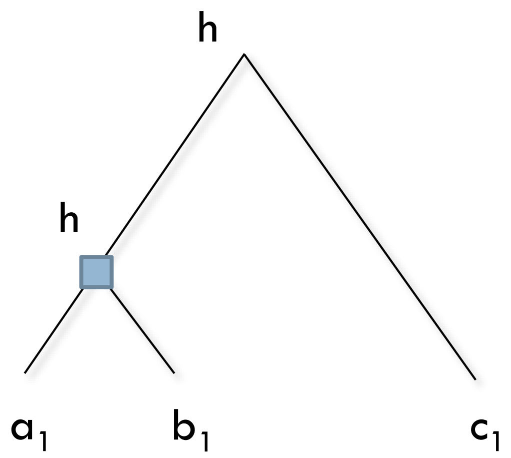
- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.

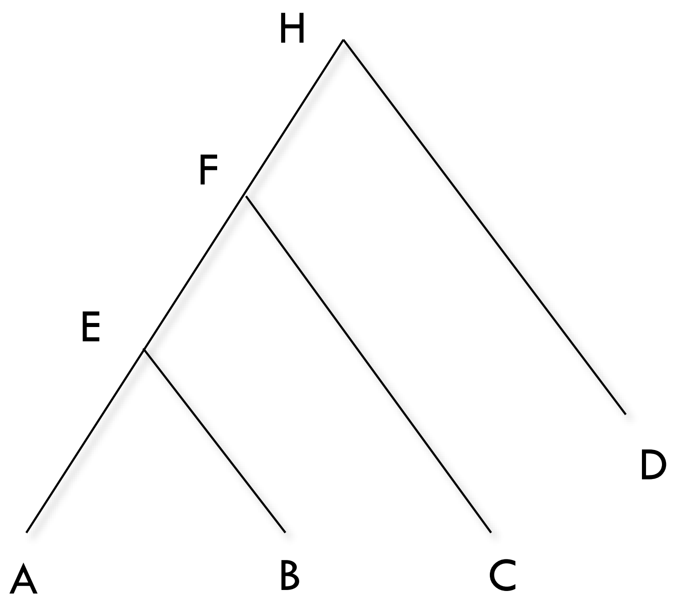




Reconciling with segmental Dups

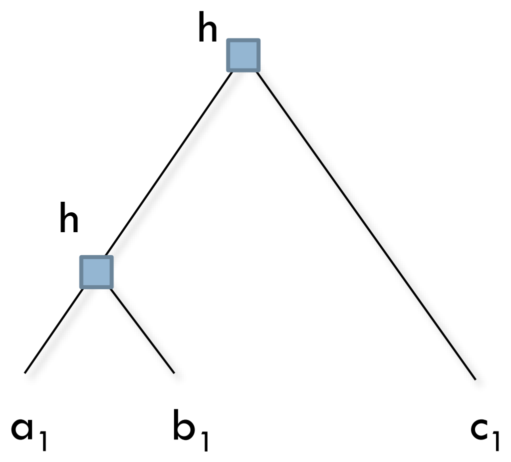
- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.

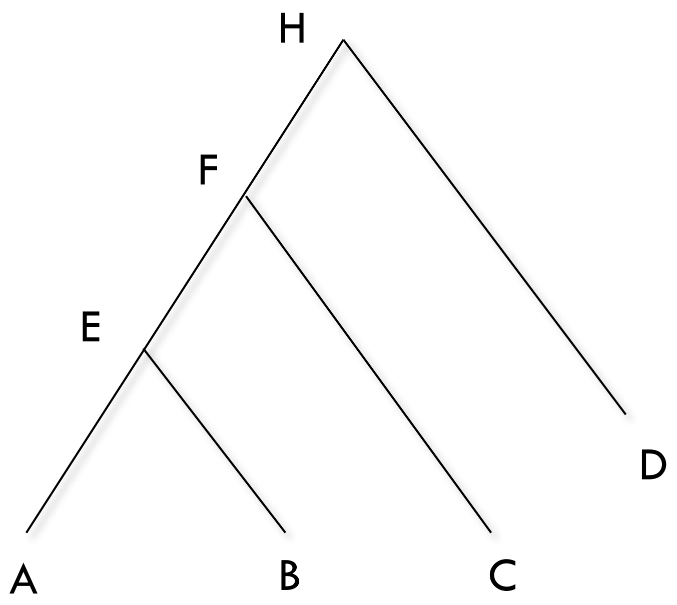




Reconciling with segmental Dups

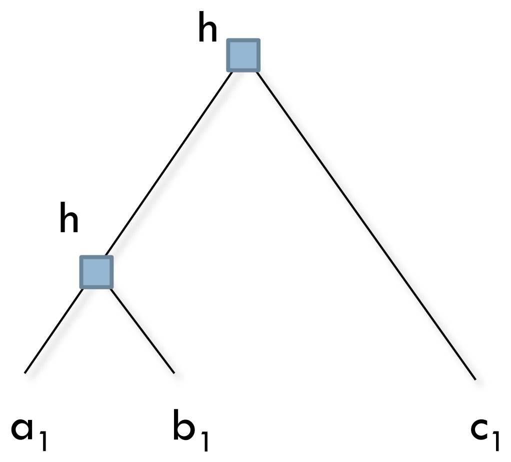
- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.

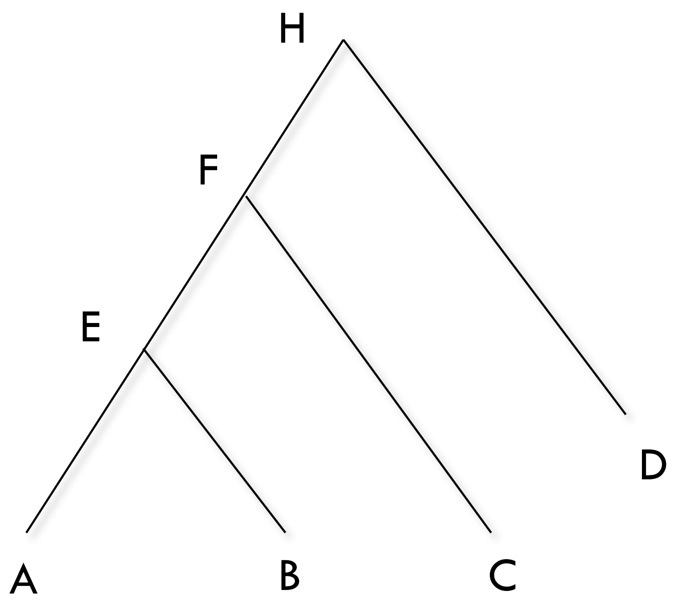




Reconciling with segmental Dups

- A node mapped above its LCA mapping must be a Dup.
- Preserve time-consistency in mapping.
- Remapping a node can create a chain of Dups above it.





Some people worked on this

- Episode Clustering
  - Minimize # of species that underwent Dup, given that **remapping a node cannot force remapping its parent**.
  - Can be solved exactly in poly-time.
  - *[Cotton & Page, Biocomputing 2002], [Burleigh & al., RECOMB 2008]*

- Minimize Dup heights, **under the same constraints**.
  - Heuristics *[Guigó & al., Mol Phylo Evol 1996]*
  - Exact *[Bansal & Eulenstein, Bioinformatics 2008], [Luo & al., TCBB 2011]*
  - Other type of contraints *[Paszek & Gorecki, TCBB 2017]*

- *Our contributions: **get rid of constraints + incorportate losses**.*

The case of $\lambda \geq \delta$

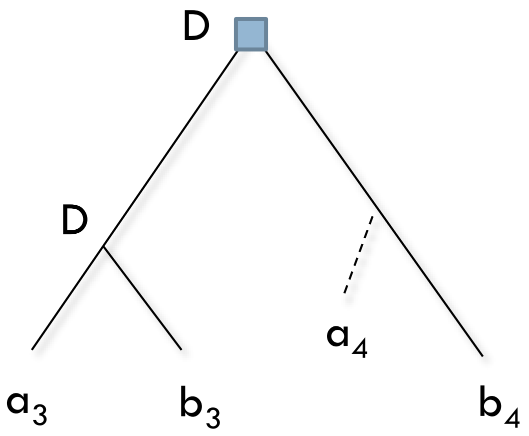
- $\lambda \geq \delta \Rightarrow$ losses are worse than Dups.

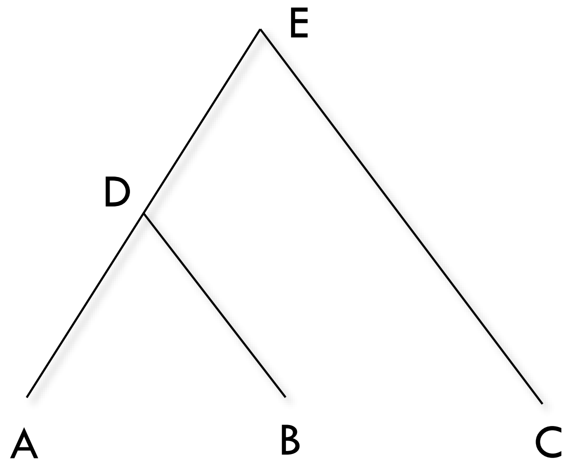
The case of $\lambda \geq \delta$

- $\lambda \geq \delta$ => losses are worse than Dups.
- Remapping an ancestral node to a higher species will **always create additional losses**.

The case of $\lambda \geq \delta$

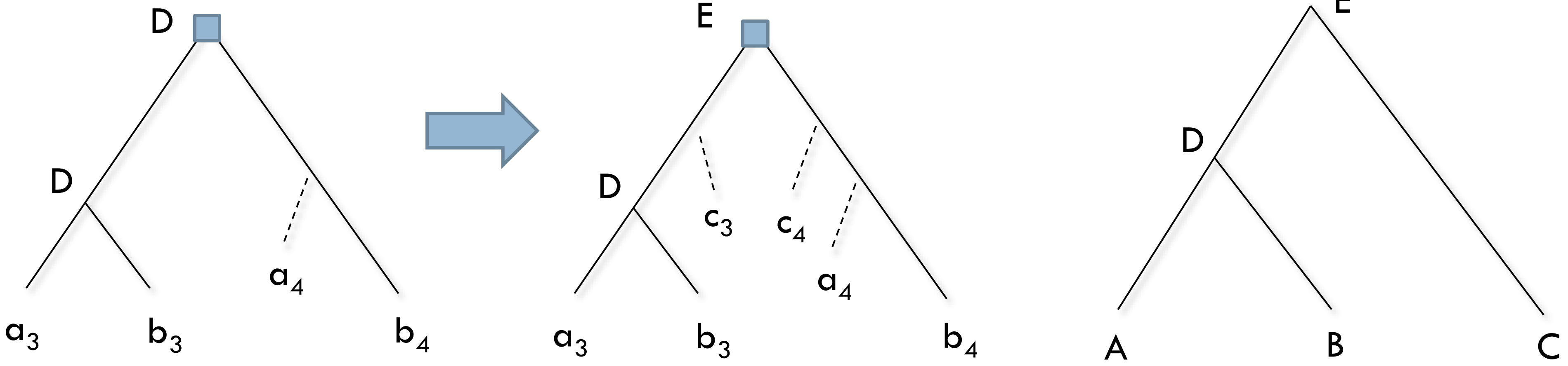
- $\lambda \geq \delta$ => losses are worse than Dups.
- Remapping an ancestral node to a higher species will **always create additional losses**.





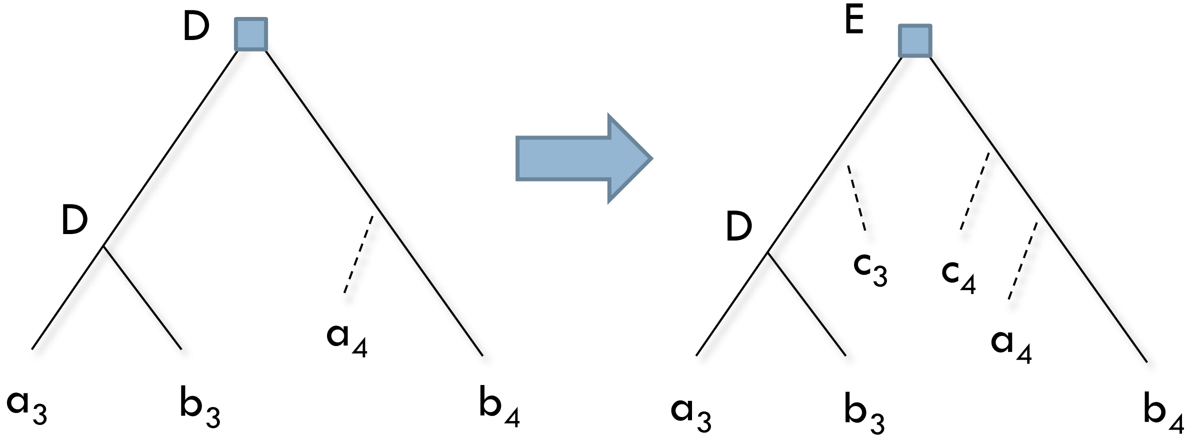
The case of $\lambda \geq \delta$

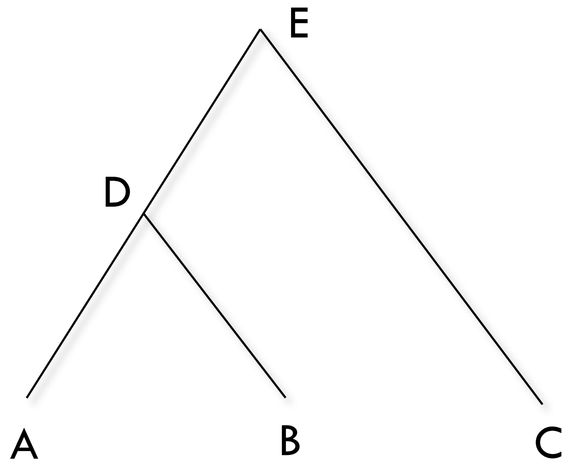
- $\lambda \geq \delta$ => losses are worse than Dups.
- Remapping an ancestral node to a higher species will **always create additional losses**.



The case of $\lambda \geq \delta$

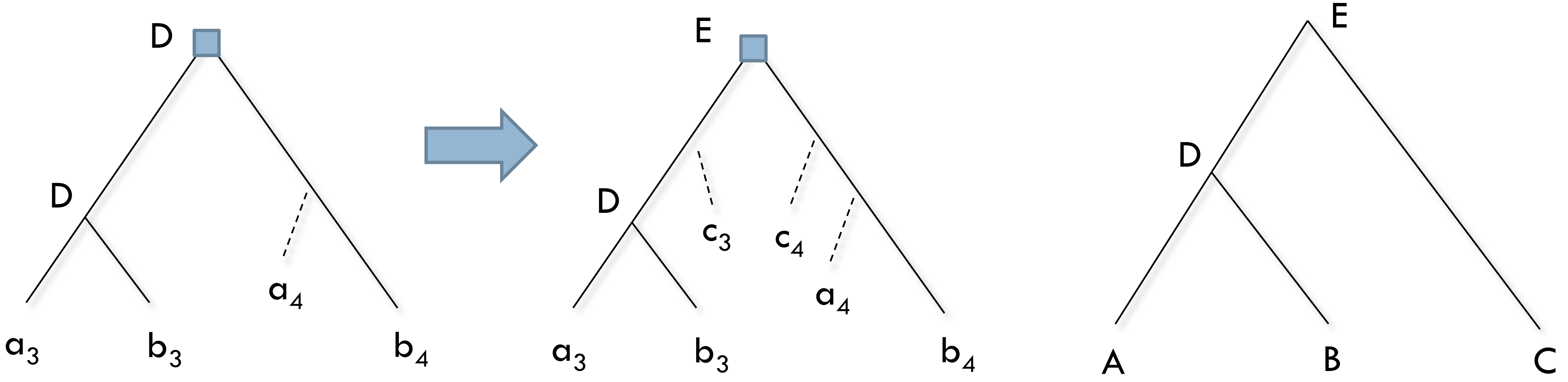
- $\lambda \geq \delta$ => losses are worse than Dups.
- Remapping an ancestral node to a higher species will **always create additional losses**.
- Remapping saves at most one Dup, but creates at least one loss => not really worth it.





The case of $\lambda \geq \delta$

- **Theorem**: when $\lambda \geq \delta$, the usual LCA mapping yields an optimal reconciliation. It is also the unique optimal reconciliation if $\lambda > \delta$.



The case of $\lambda = 0$

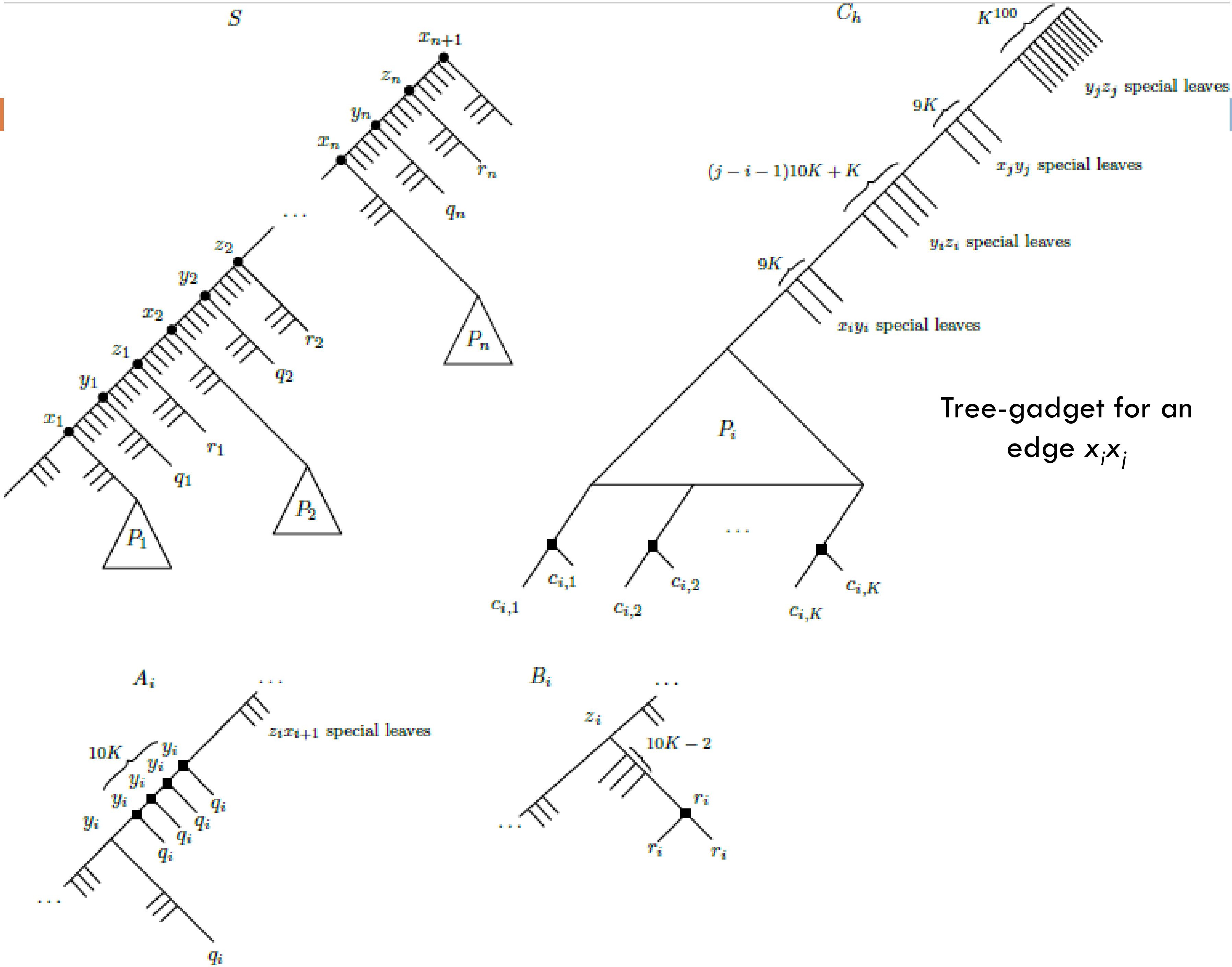
- When λ = 0, we only care about the sum of Dup heights.
- Complexity was left opened by Paszek & Gorecki.

- **Theorem:** Finding an optimal reconciliation with segmental Dups when λ = 0 is NP-hard.

The case of $\lambda = 0$

- When $\lambda = 0$, we only care about the sum of Dup heights.
- Complexity was left opened by Paszek & Gorecki.

- **Theorem:** Finding an optimal reconciliation with segmental Dups when $\lambda = 0$ is NP-hard.
  - Reduction from Vertex Cover
  - 7-page proof, see paper



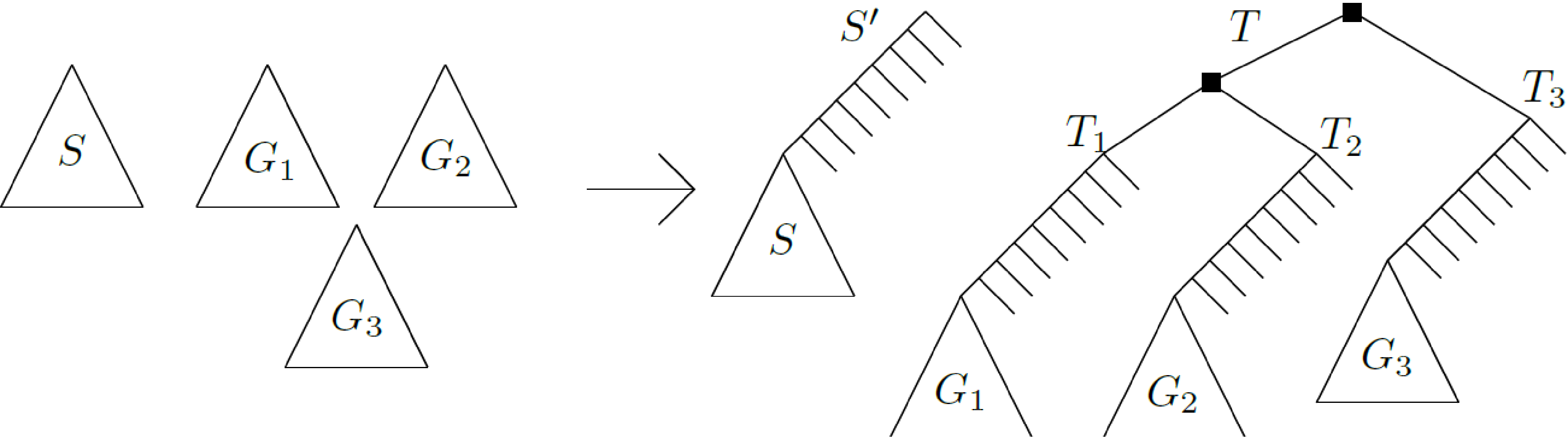
Tree-gadget for an edge x_ix_j

The case of $\lambda = 0$

- **Theorem**: finding an optimal reconciliation with segmental Dups when $\lambda = 0$ is NP-hard, **even if only one gene tree is given in the input.**

The case of $\lambda = 0$

- **Theorem**: finding an optimal reconciliation with segmental Dups when $\lambda = 0$ is NP-hard, **even if only one gene tree is given in the input.**
  - Reduction from reconciliation with many gene trees: just join all the gene trees under many speciations.

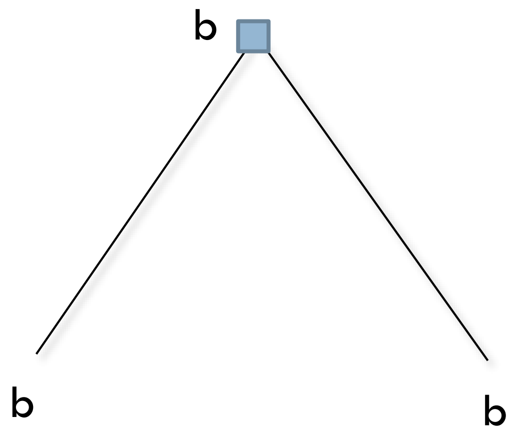


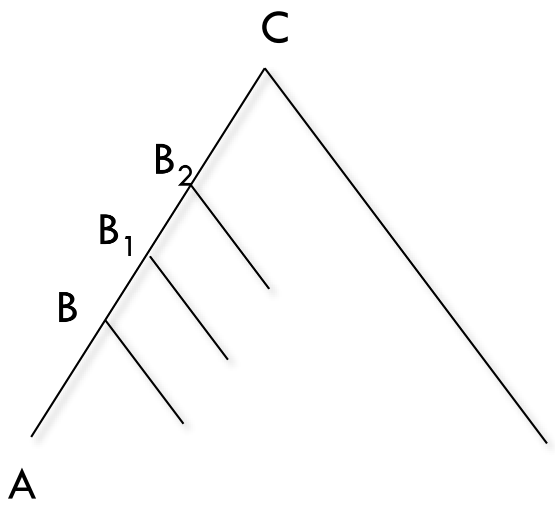
An FPT algorithm for $\lambda < \delta$

- An $O((\delta/\lambda)^{d+1} n)$ time algorithm.
  - *d* is the sum of Dup heights in an optimal solution
  - e.g. when $\delta = 3$, $\lambda = 2$, we get a $O(1.5^{d+1} n)$ algorithm.

An FPT algorithm for $\lambda < \delta$

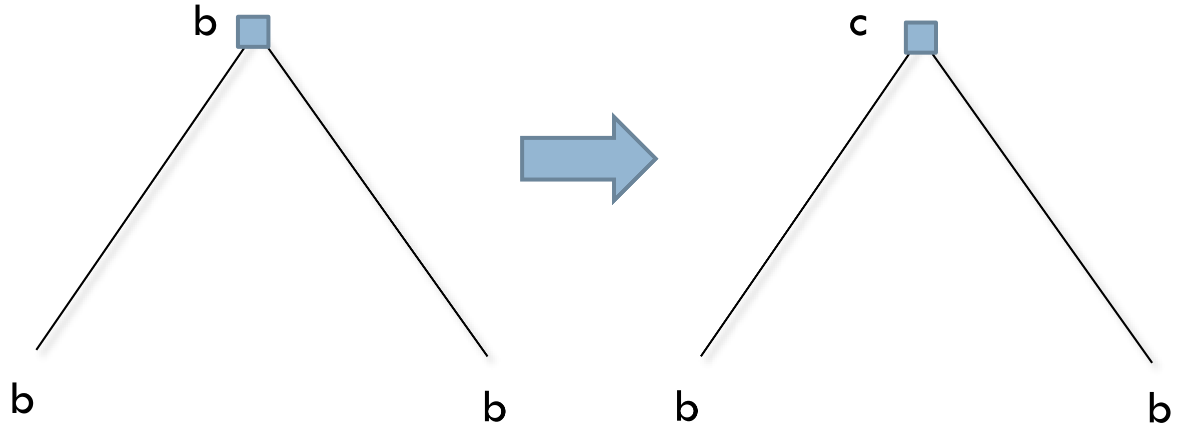
- When we remap a Dup node up by *k* species, we create at least *k* new losses.

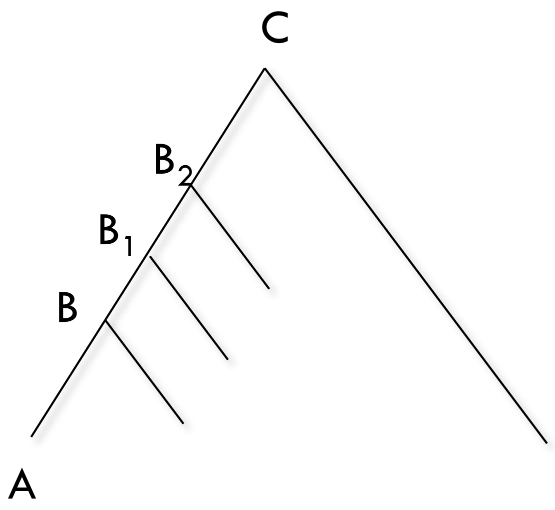




An FPT algorithm for λ < δ

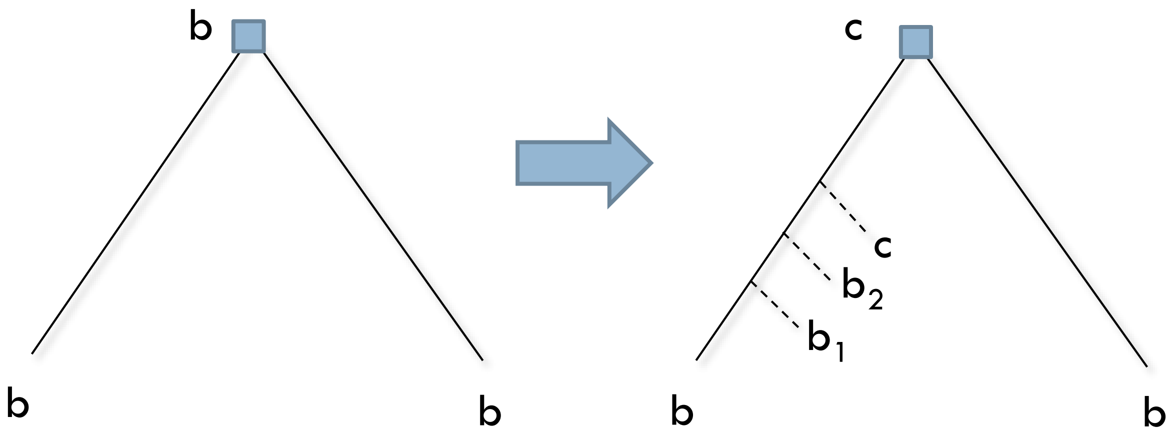
- When we remap a Dup node up by *k* species, we create at least *k* new losses.

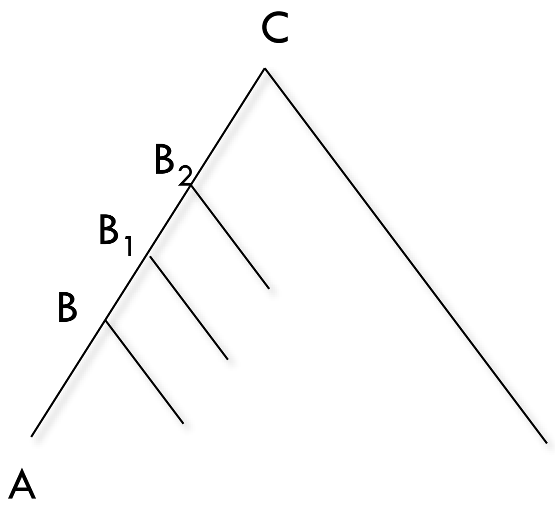




An FPT algorithm for $\lambda < \delta$

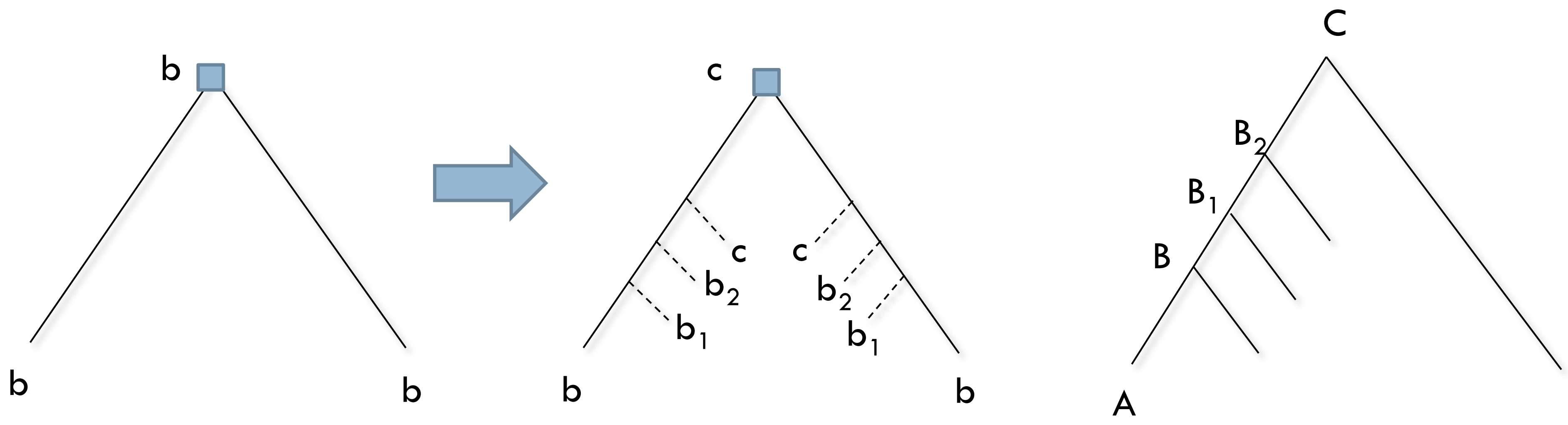
- When we remap a Dup node up by *k* species, we create at least *k* new losses.





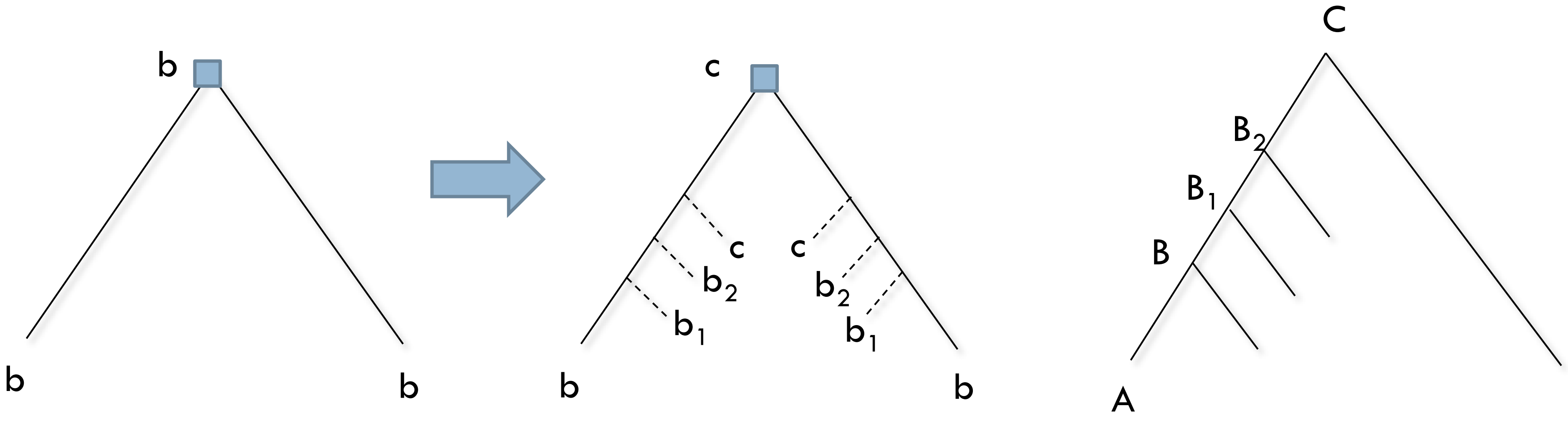
An FPT algorithm for $\lambda < \delta$

- When we remap a Dup node up by *k* species, we create at least *k* new losses.



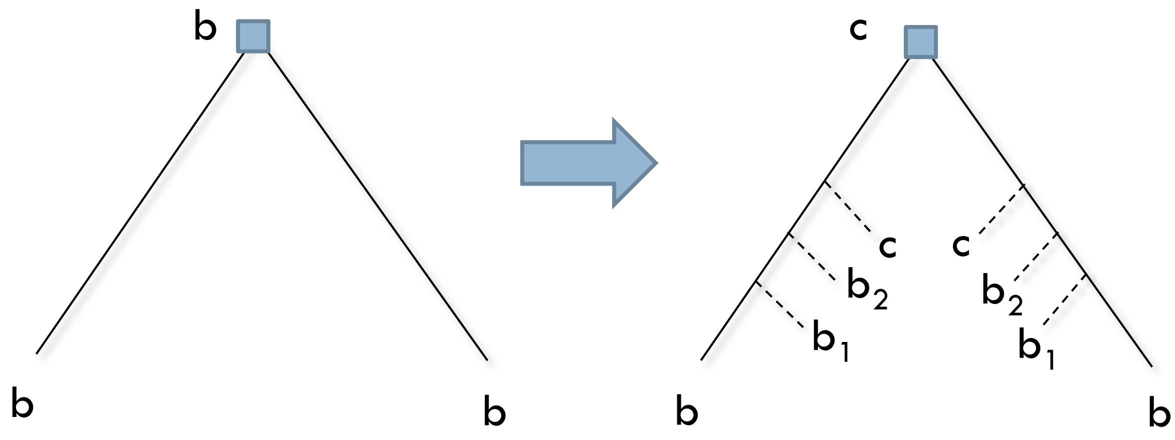
An FPT algorithm for $\lambda < \delta$

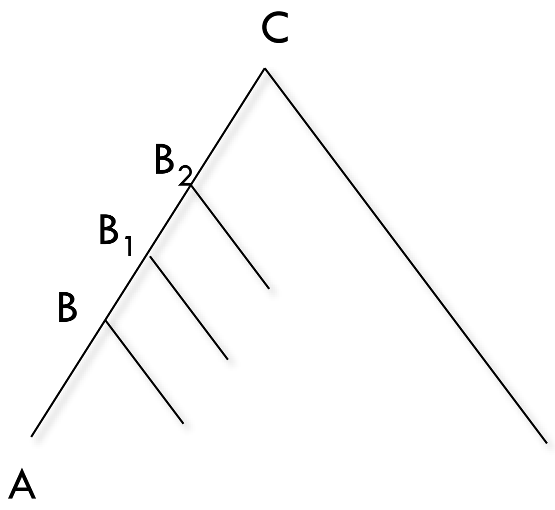
- When we remap a Dup node up by *k* species, we create at least *k* new losses.
- If we remap a Dup node up by more than δ/λ species, we save 1 Dup but create $> \delta/\lambda$ losses.



An FPT algorithm for $\lambda < \delta$

- When we remap a Dup node up by *k* species, we create at least *k* new losses.
- If we remap a Dup node up by more than δ/λ species, we save 1 Dup but create $> \delta/\lambda$ losses.
  - Cost changes by $> -\delta + \lambda * (\delta/\lambda) = 0$.
  - *Not worth it.*





An FPT algorithm for $\lambda < \delta$

- Branching algorithm:
  - Take a Dup node *x* mapped to species *s* under the LCA mapping.
  - Branch into the δ/λ possible ways of remapping *x* to an ancestor *s'* of *s*.
    - *Each time we branch, Dup heights increase by 1.*
    - *Must also remap other nodes who « want » to remap to s'.*

An FPT algorithm for $\lambda < \delta$

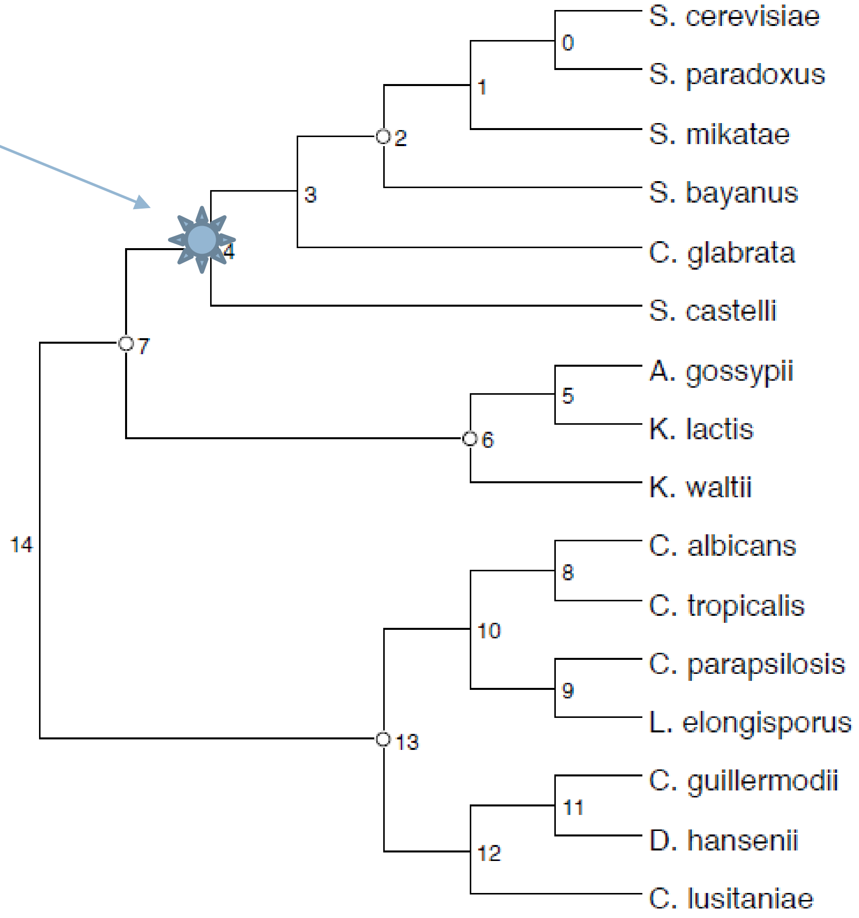
- Branching algorithm:
  - Take a Dup node *x* mapped to species *s* under the LCA mapping.
  - Branch into the δ/λ possible ways of remapping *x* to an ancestor *s'* of *s*.
    - *Each time we branch, Dup heights increase by 1.*
    - *Must also remap other nodes who « want » to remap to s'.*
  - Search tree of degree δ/λ and height at most *d*.
    - $O((\delta/\lambda)^{d+1} n)$ complexity

Experiments

- We implemented the FPT algorithm.
  - https://github.com/manuellafond/Multrec
- We applied it on 2 datasets:
  - Yeast species from [Butler & al., Nature, 2009]
    - 16 species, 2379 gene trees
  - Eukaryotes from [Guigo & al., Mol Phylo Evo, 1996]
    - 16 species, 53 gene trees

Experiments

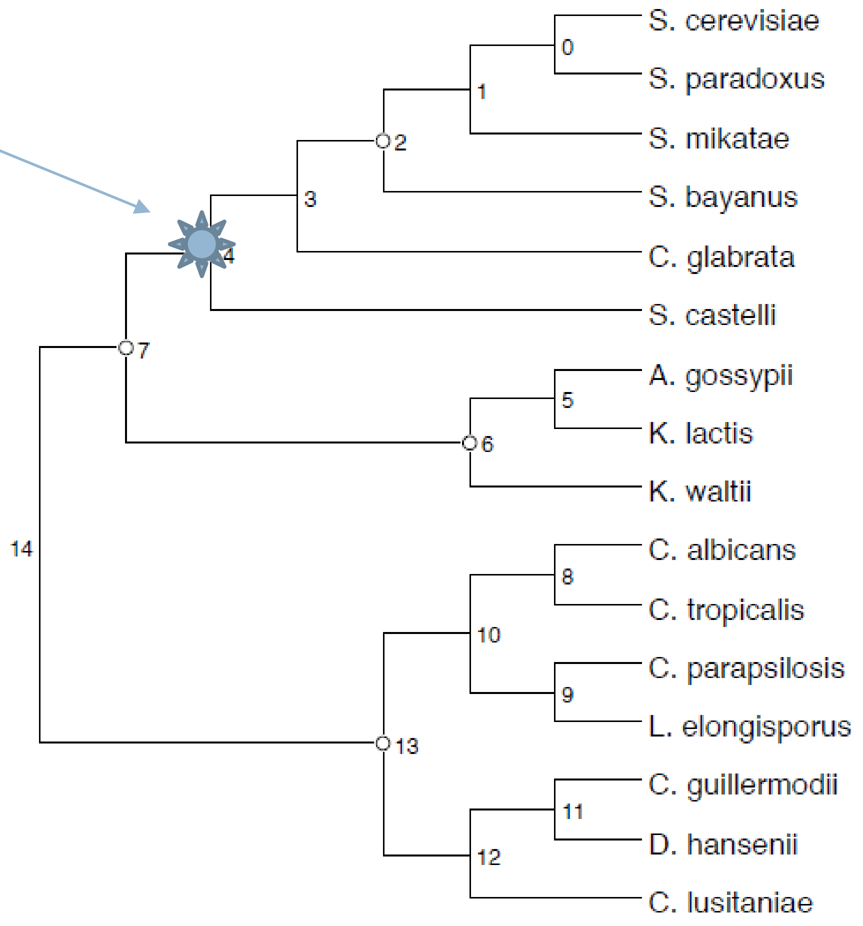
- In the 2379 yeast trees, we infer a segmental Dup with **216 genes** (δ = 3, λ = 2).
  - Located here



Experiments

- In the 2379 yeast trees, we infer a segmental Dup with **216 genes** (δ = 3, λ = 2).
  - Located here
  - Coincides with WGD found using synteny in *[Kellis, Birren & Lander, Nature, 2004]*

Nodes 7,6,13,2 had segmental Dup with 190, 157, 148 and 136 *genes.*



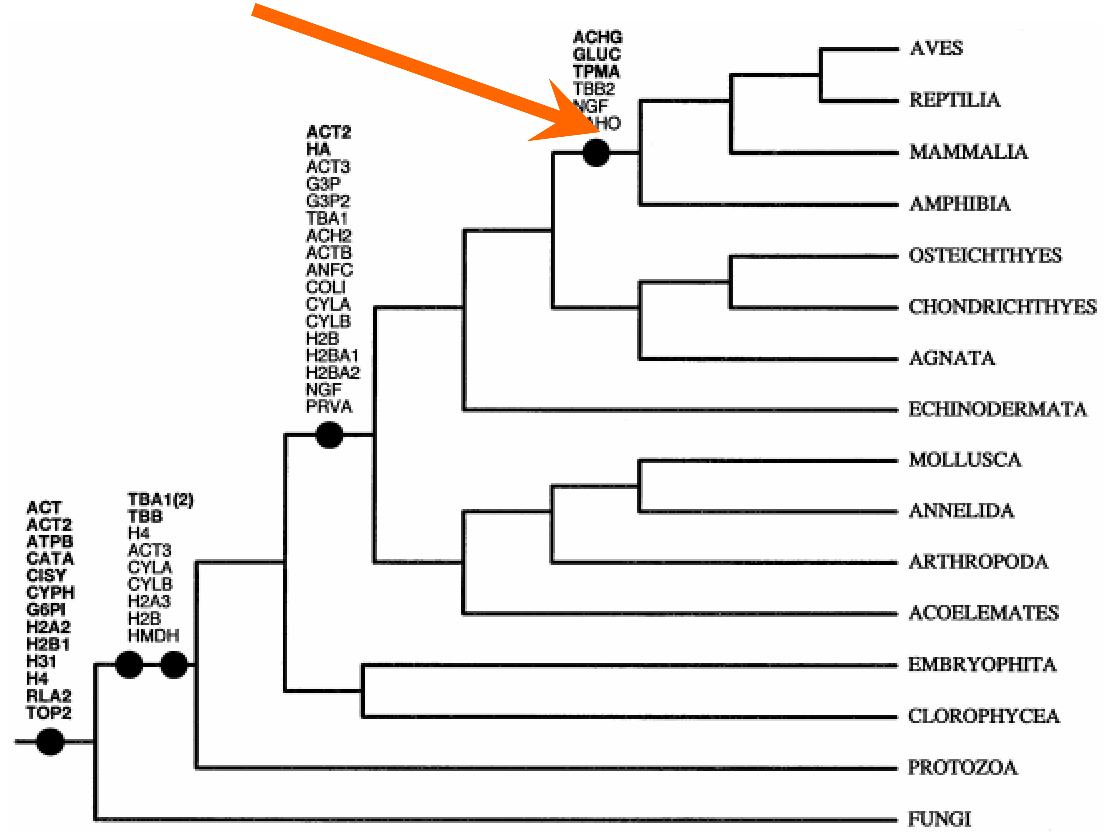
Experiments

- In the 53 Eukaryote gene trees.
  - *ExactMGD [Bansal & Eulenstein, Bioinf, 2008]* finds a solution with **5** segmental Dups
    - Does not allow speciations to become duplications.
  - We find a solution with **4 segmental Dups**
    - By setting $\delta > 61, \lambda = 1$
    - All segmental Dups found in *[Guigo & al., 1996]* are confirmed, **EXCEPT ONE**.

Experiments

- In the 53 Eukaryote gene trees.

In our solutions, no Dup maps here (Tetrapoda)

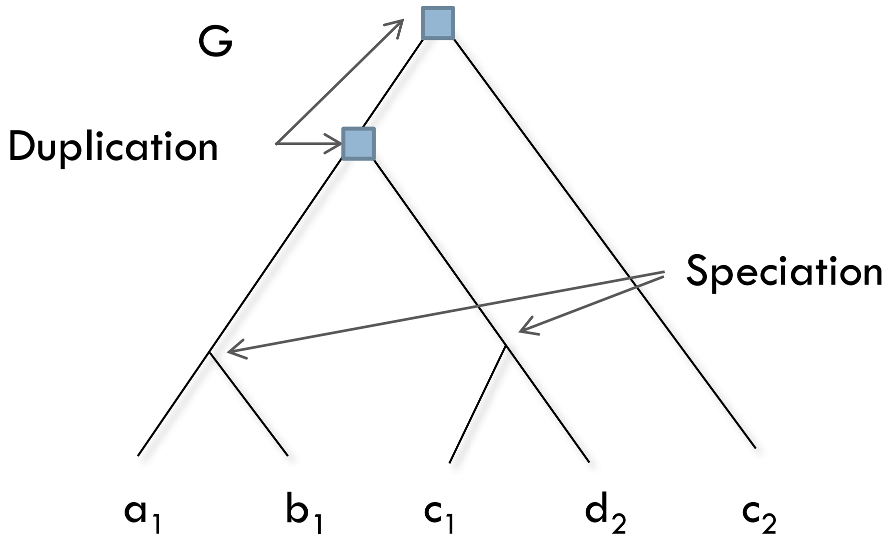


Conclusion

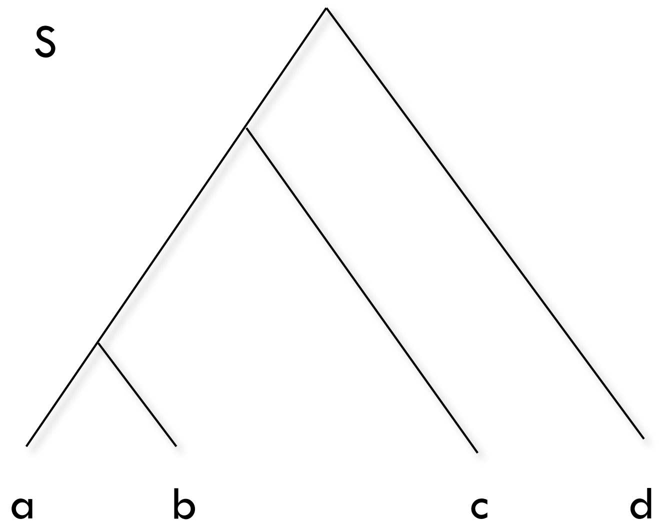
- Open problems
  - Complexity when δ/λ is a constant?
  - Approximation algorithms?
- Modeling segmental losses.
- Incorporate lateral transfer.
- More practical application (e.g. detect WGD in plants)

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



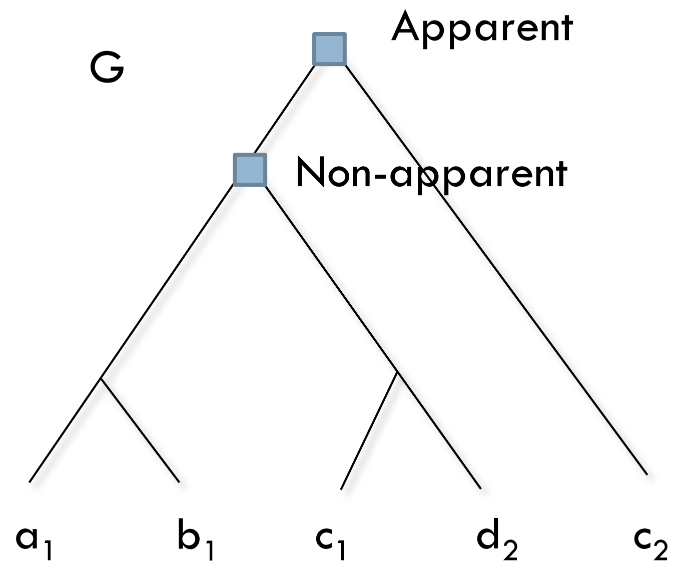
Gene tree



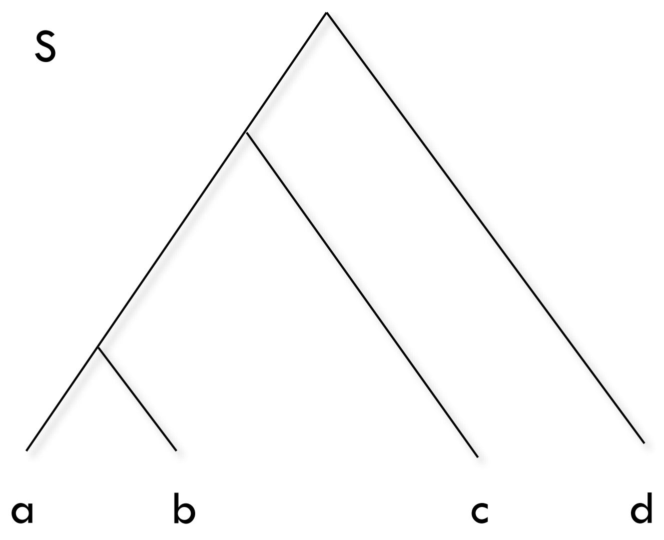
Species tree

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



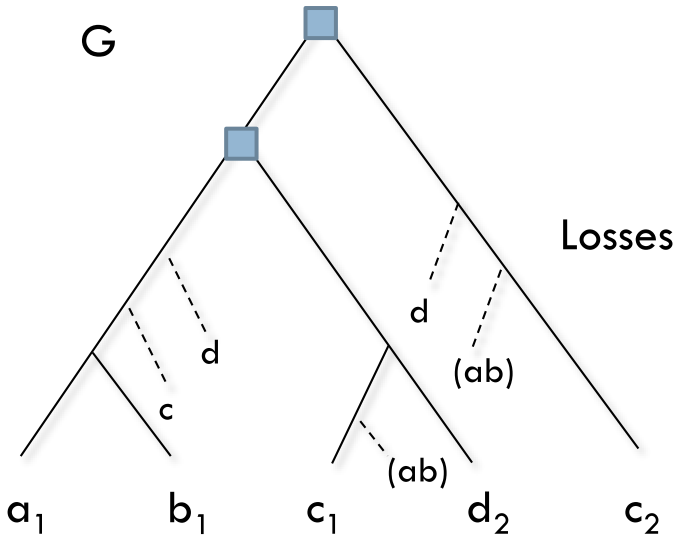
Gene tree



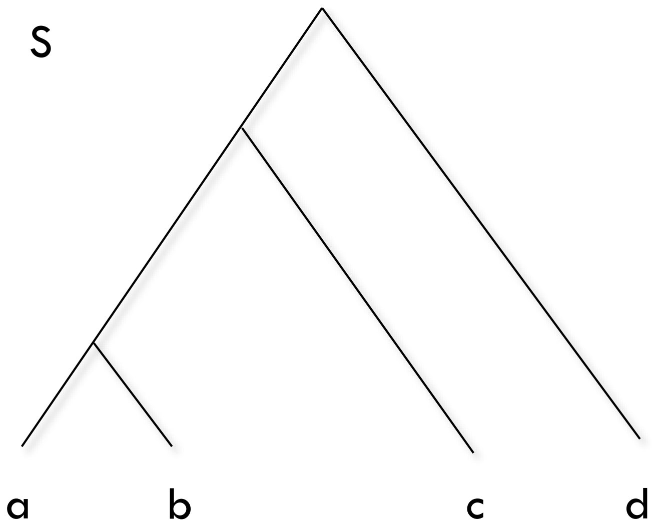
Species tree

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.



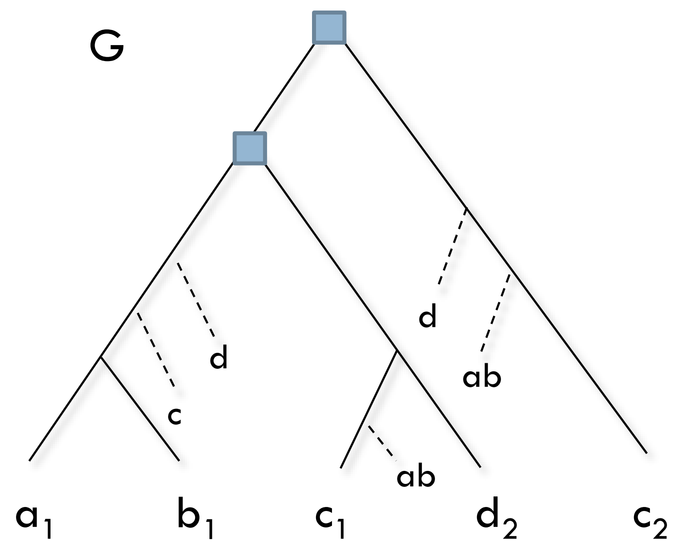
Gene tree

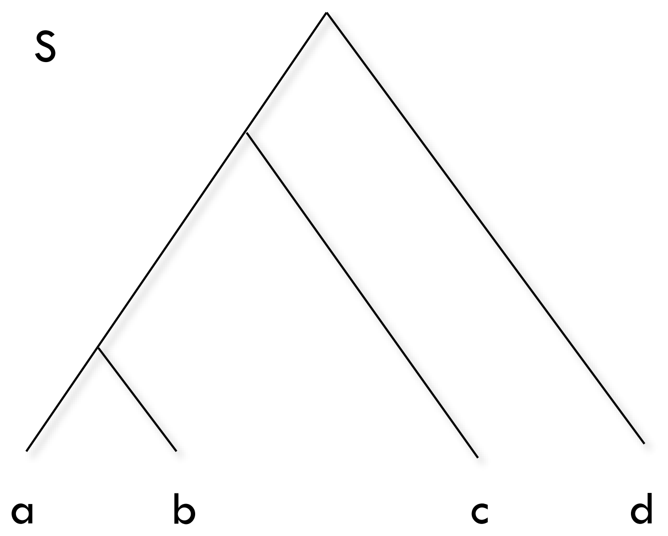


Species tree

Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in a gene tree G.

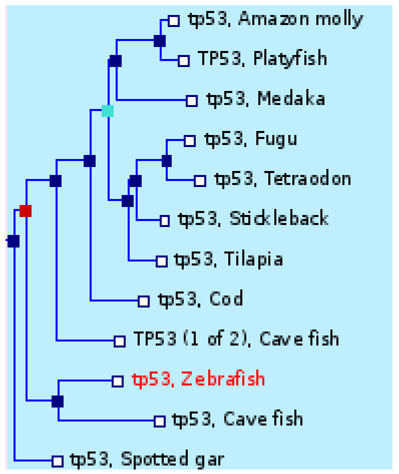




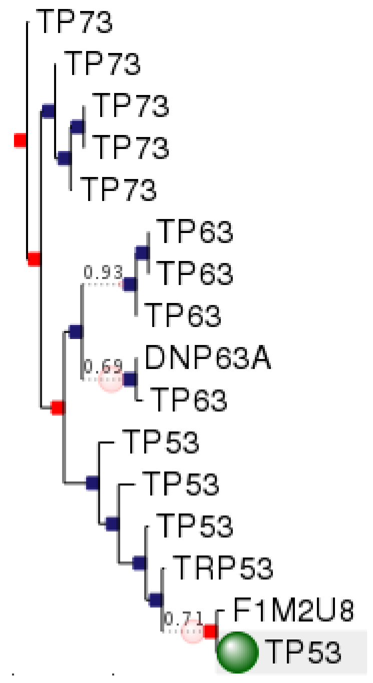
Possible reconciliation costs : #dups, #dups + #losses

TP53 gene tree(s)

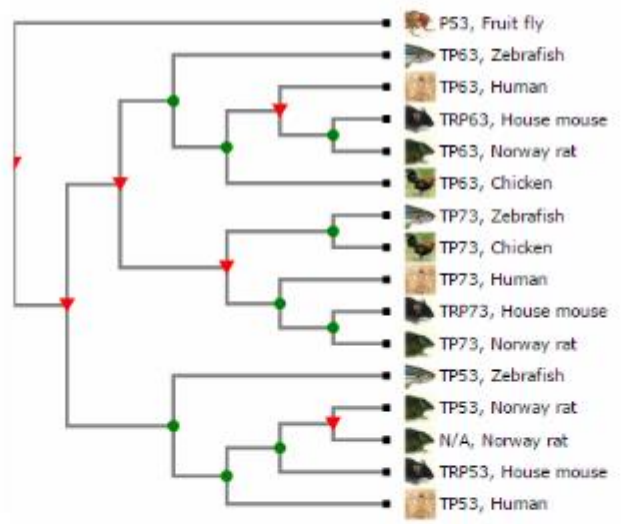
Ensembl

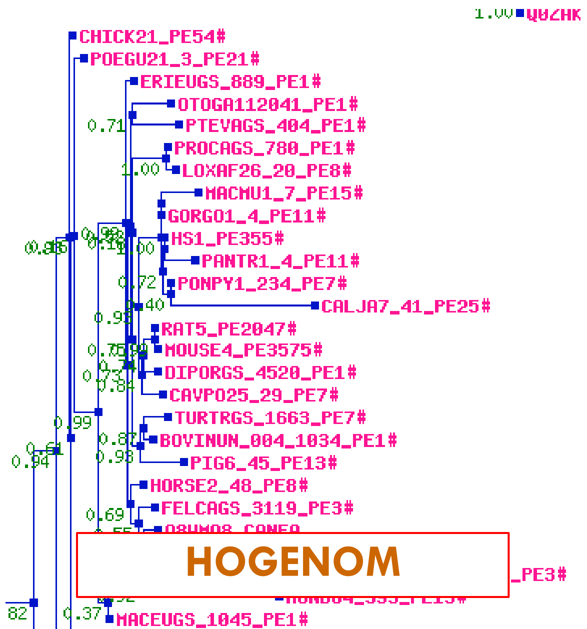


PhylomeDB



TreeFam





HOGENOM

TP53 gene tree(s)

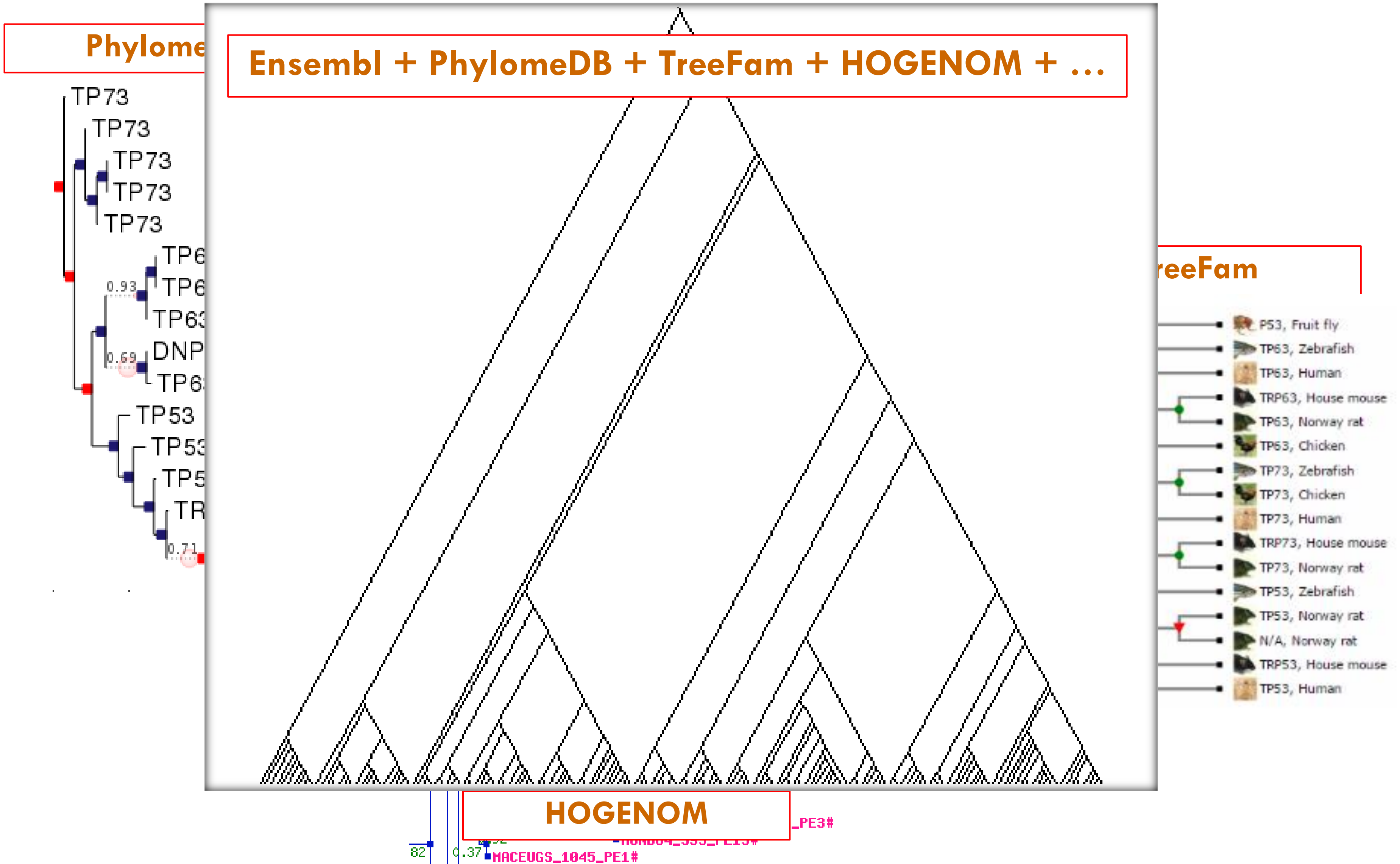
Ensembl

Phylome

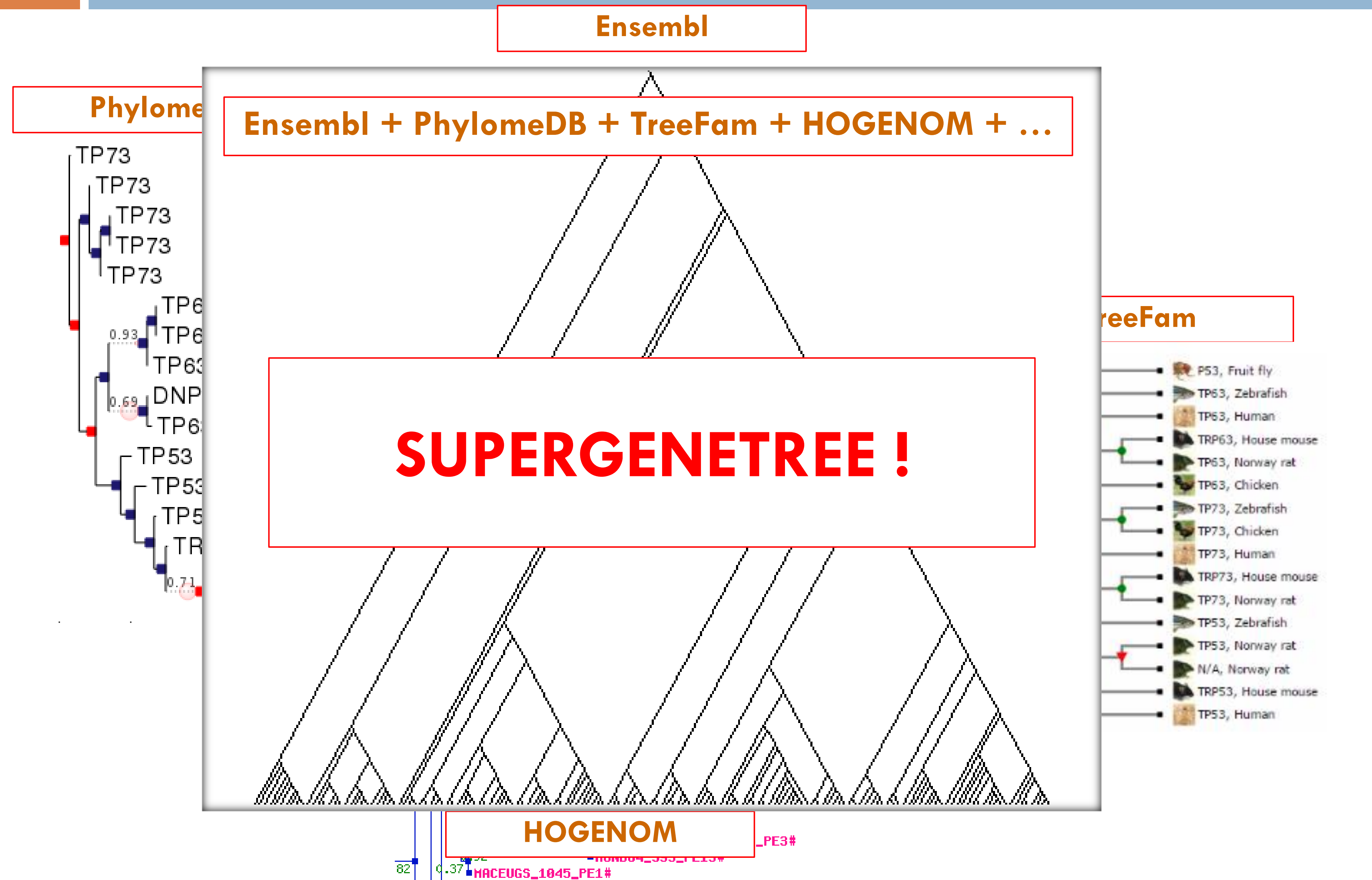
Ensembl + PhylomeDB + TreeFam + HOGENOM + …

reeFam

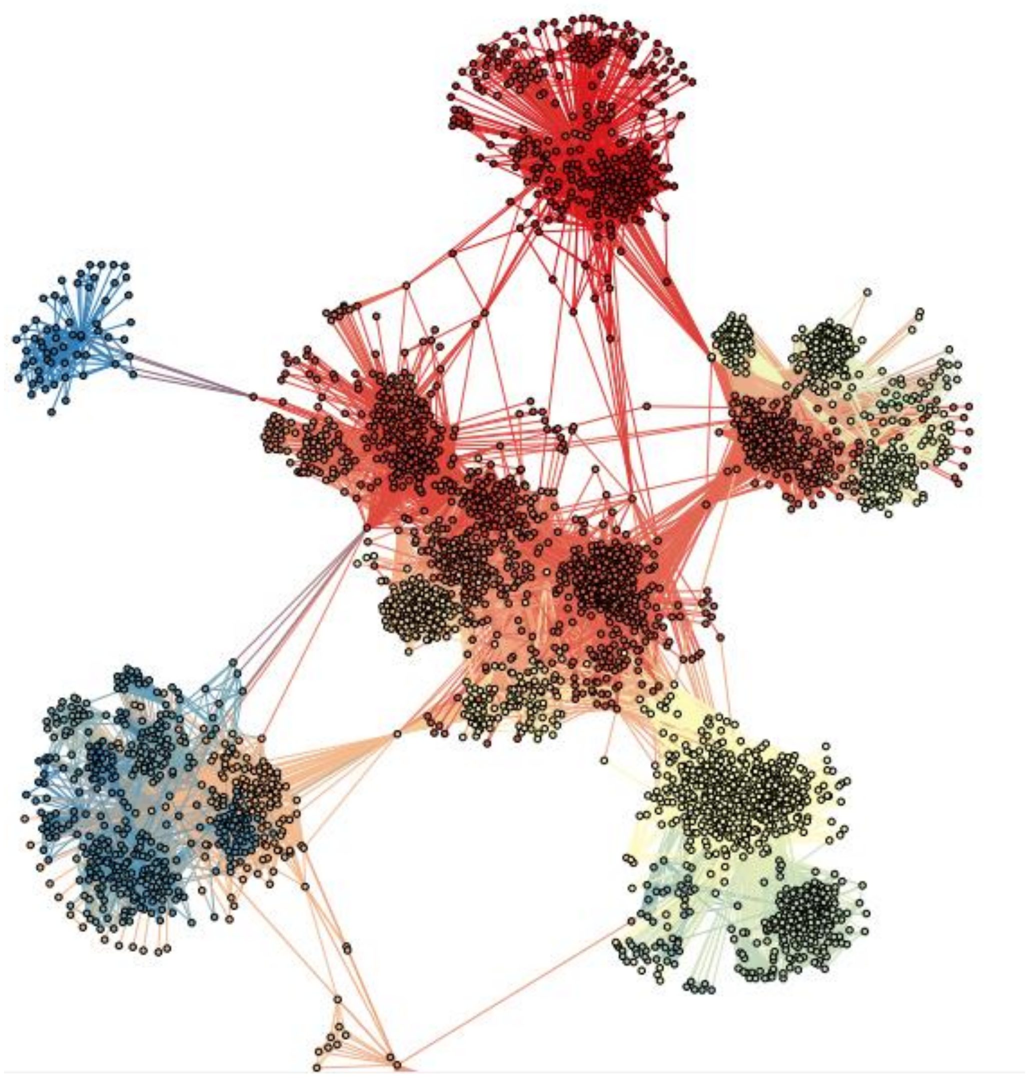
HOGENOM



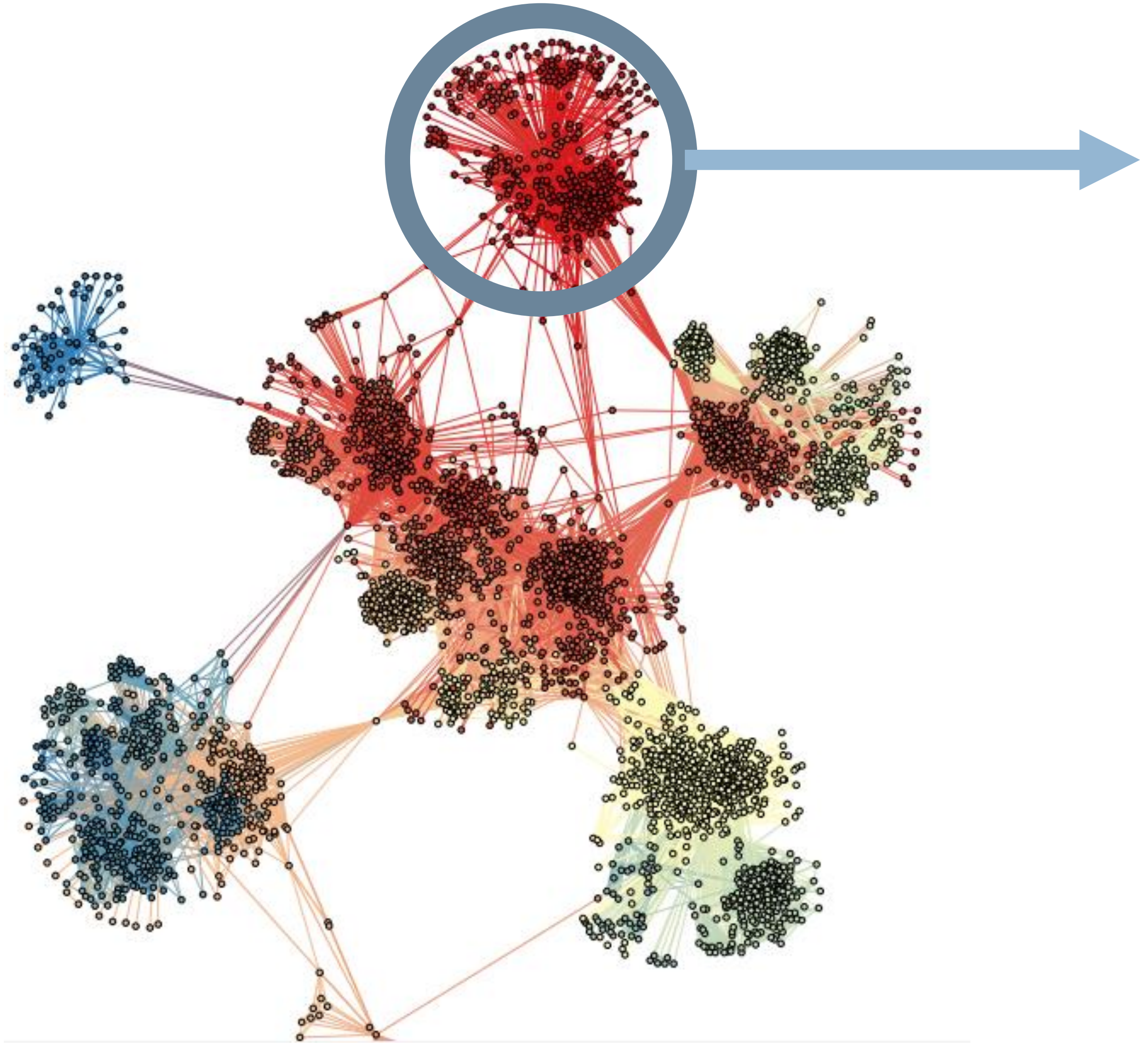
TP53 gene tree(s)

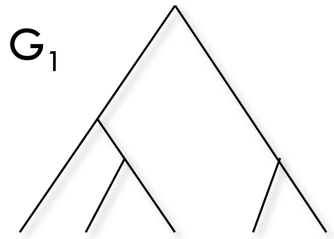


Clusters of orthologous groups

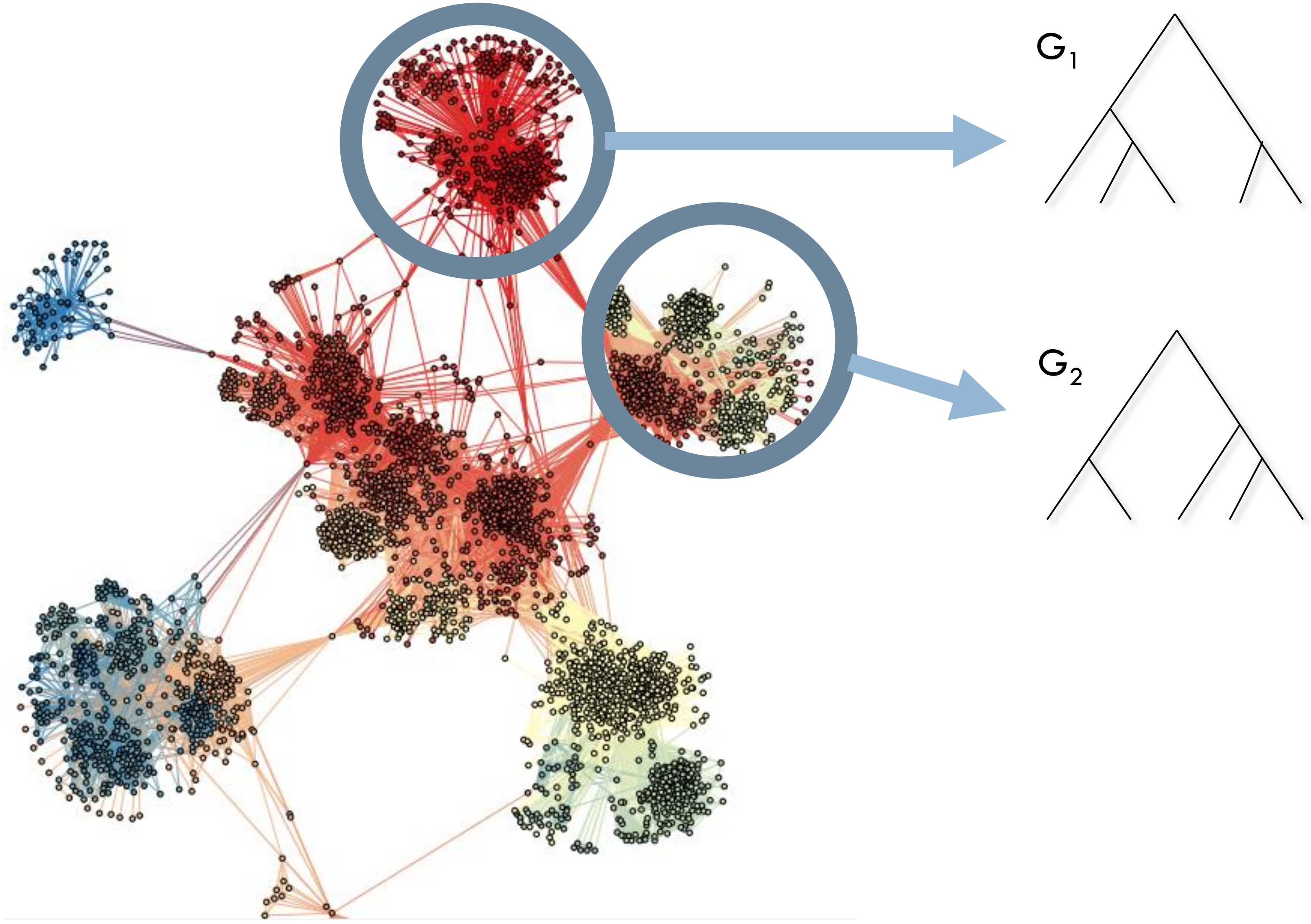


Clusters of orthologous groups

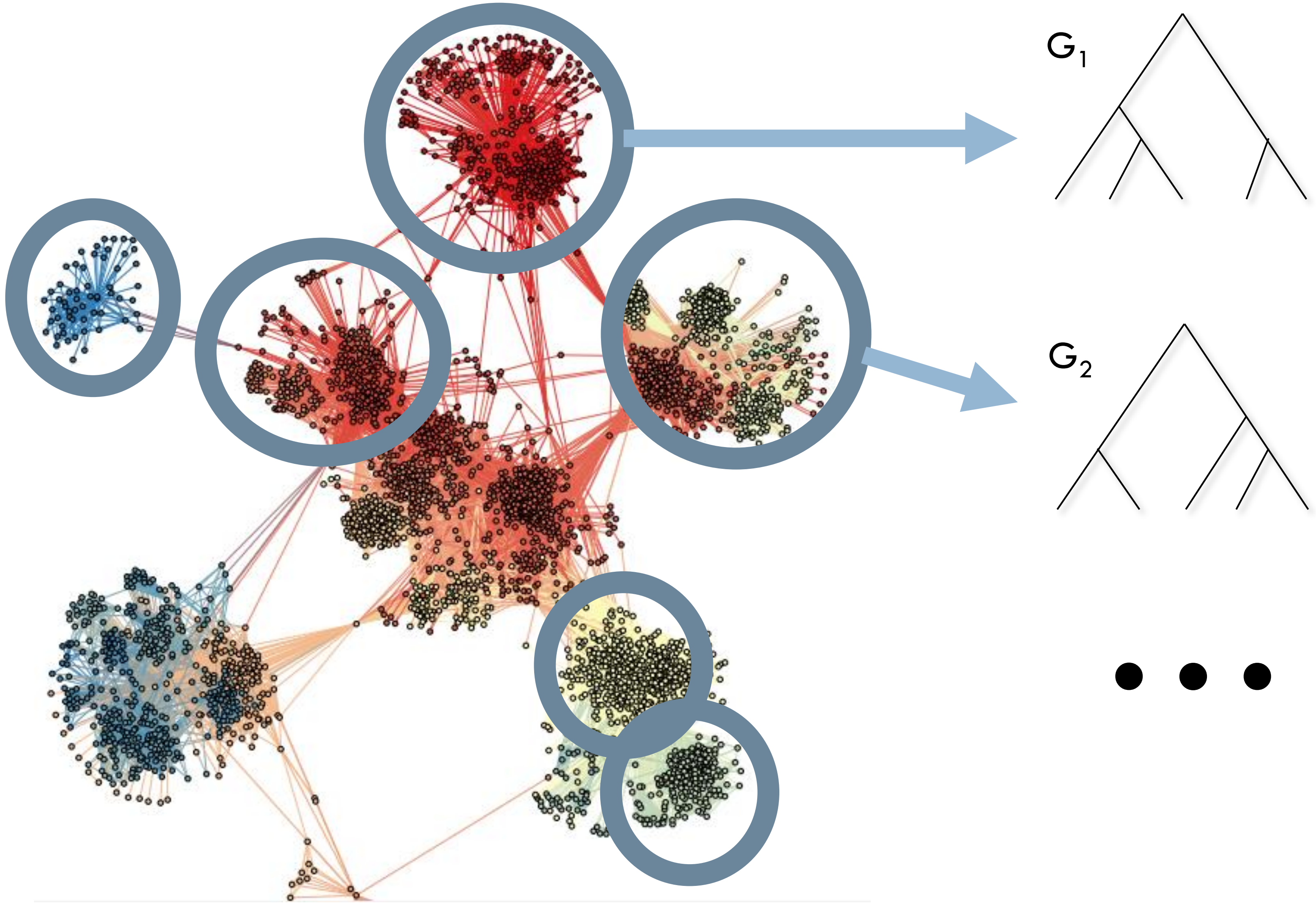




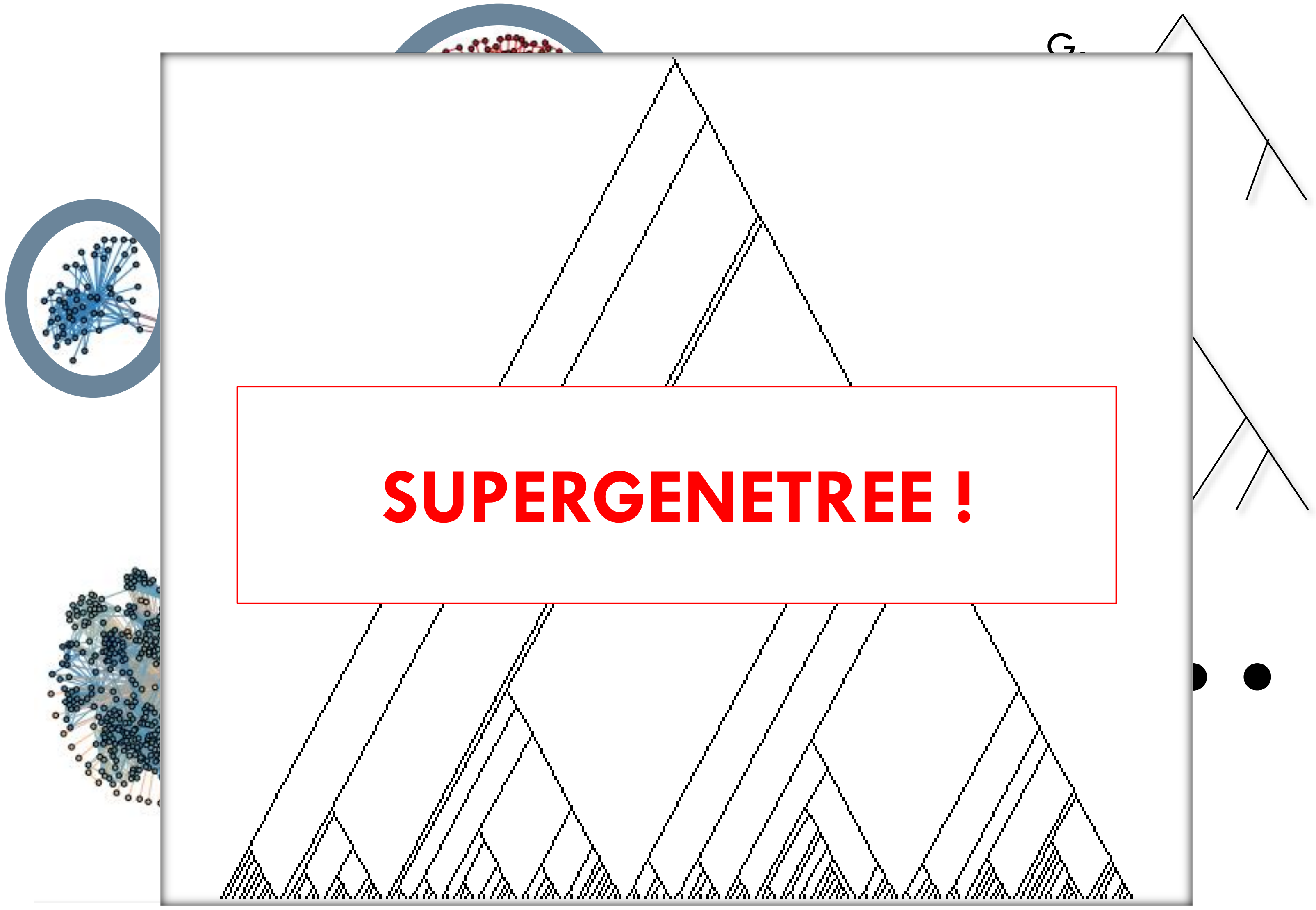
Clusters of orthologous groups



Clusters of orthologous groups

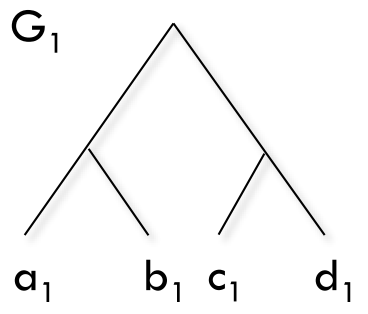


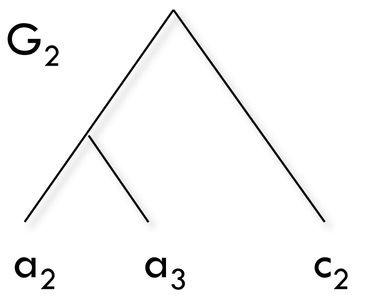
Clusters of orthologous groups

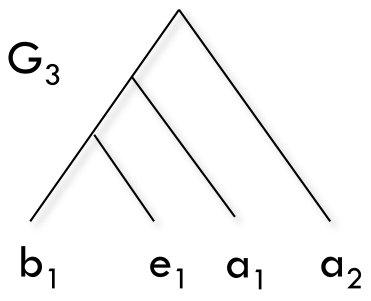


The Supergenetree problem

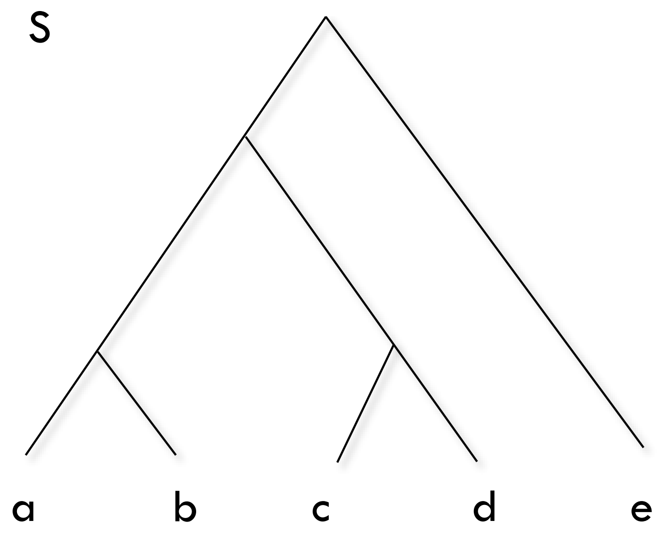
Multiple gene trees







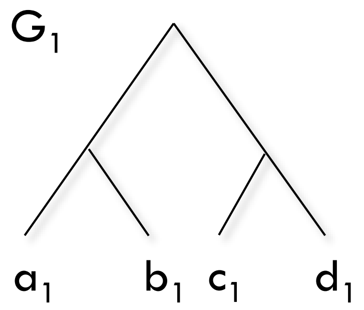
Species tree

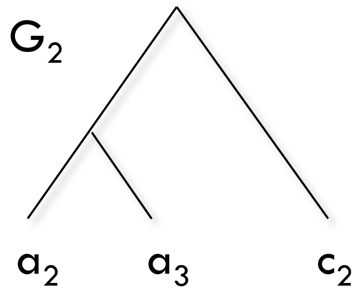


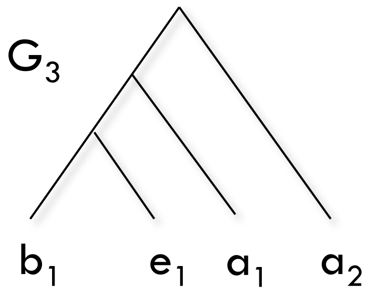
- **Gene tree label = species**
- **Multiple copies (paralogs)**
  - **e.g. a_1, a_2, a_3**
- **Gene trees may be partial + discordant with S (e.g. G_3)**

The Supergenetree problem

Multiple gene trees



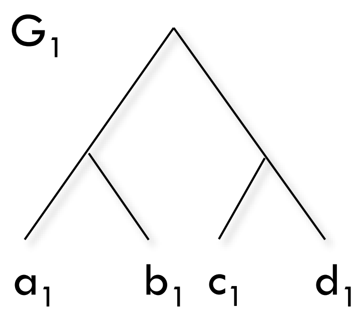


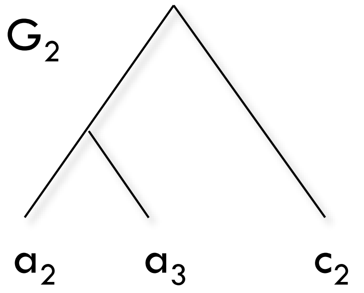


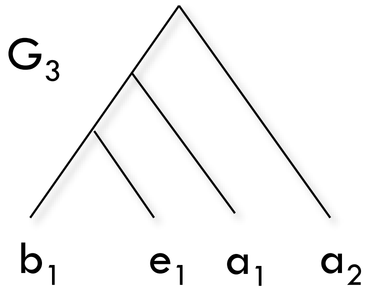
- **Our goal : find a gene tree that displays them all**

The Supergenetree problem

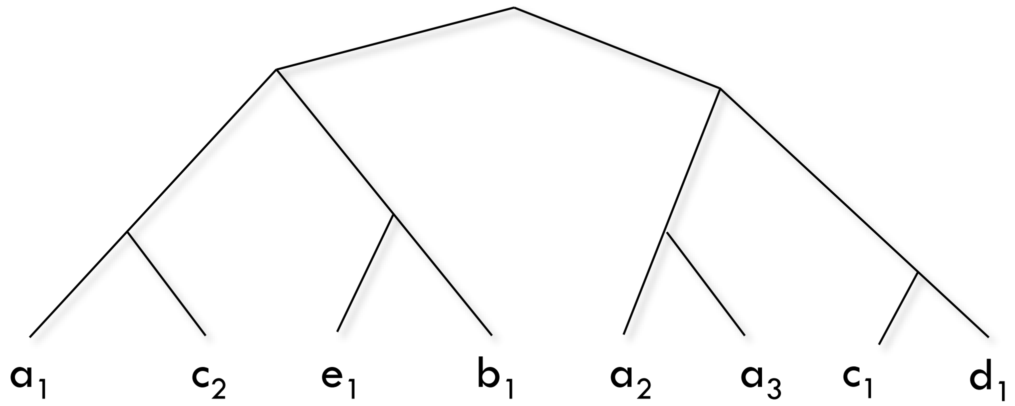
Multiple gene trees





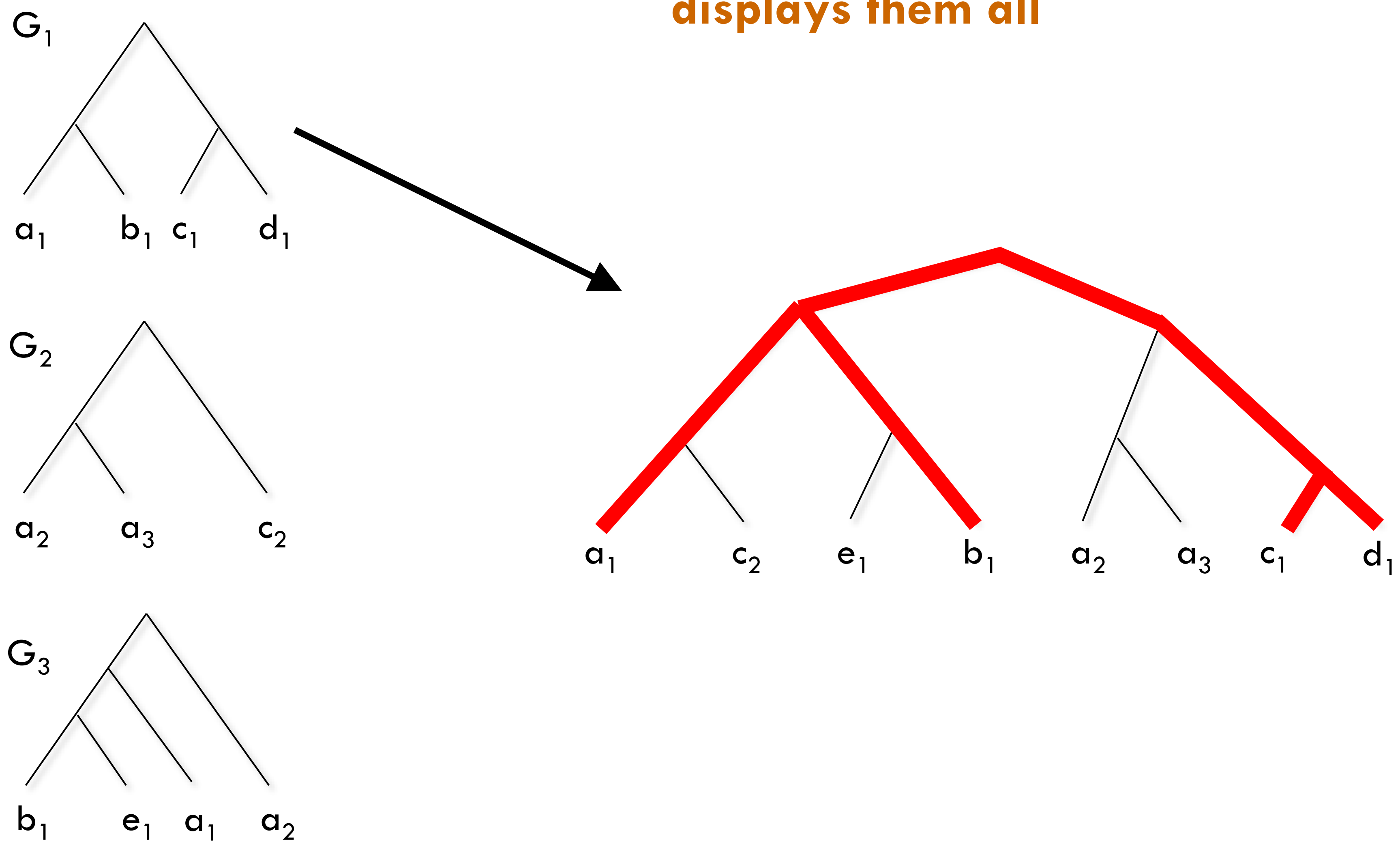


- **Our goal : find a gene tree that displays them all**



The Supergenetree problem

Multiple gene trees

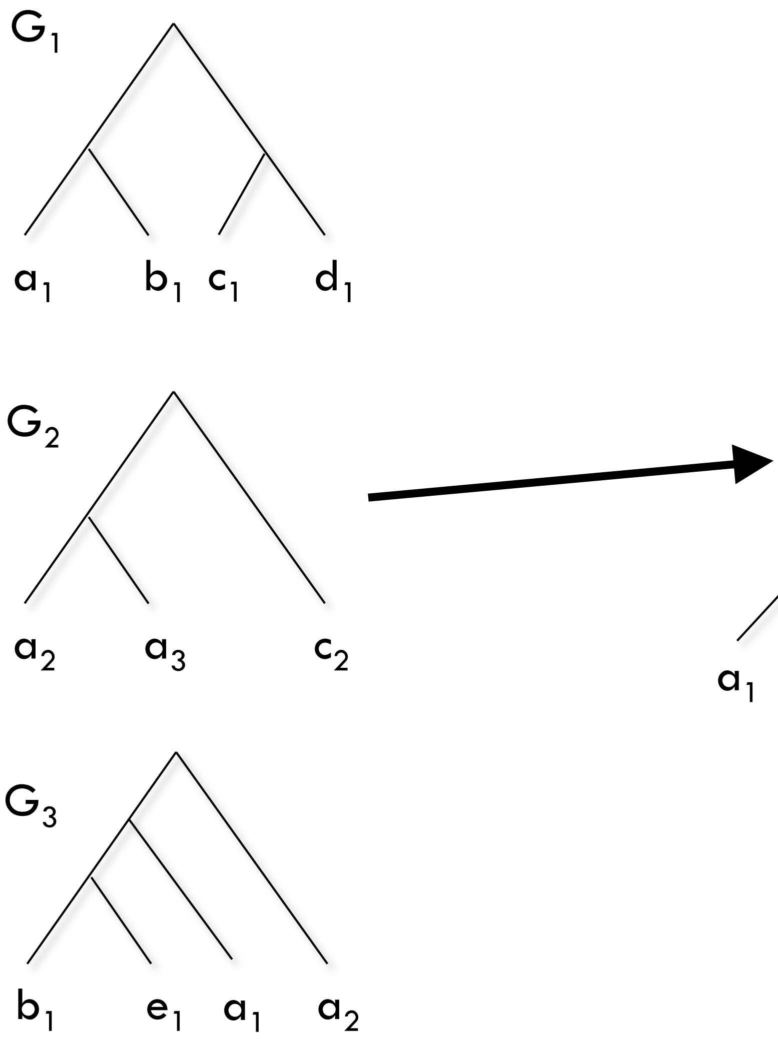


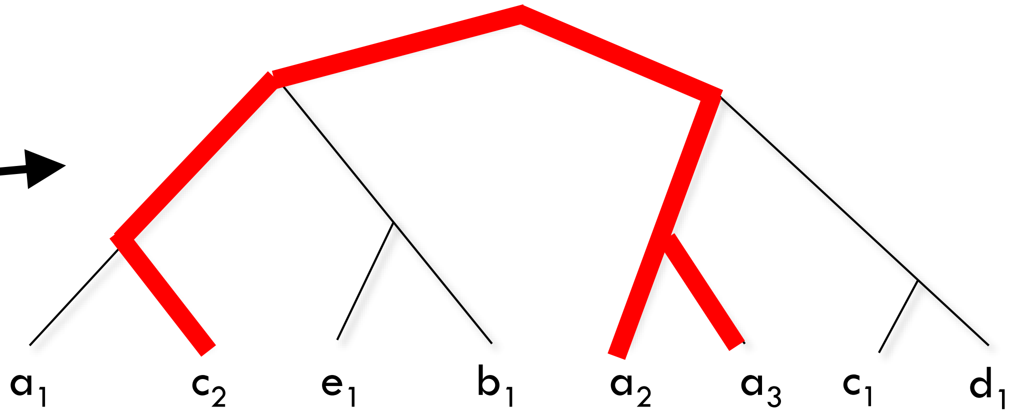
- **Our goal : find a gene tree that displays them all**

The Supergenetree problem

Multiple gene trees

- **Our goal : find a gene tree that displays them all**

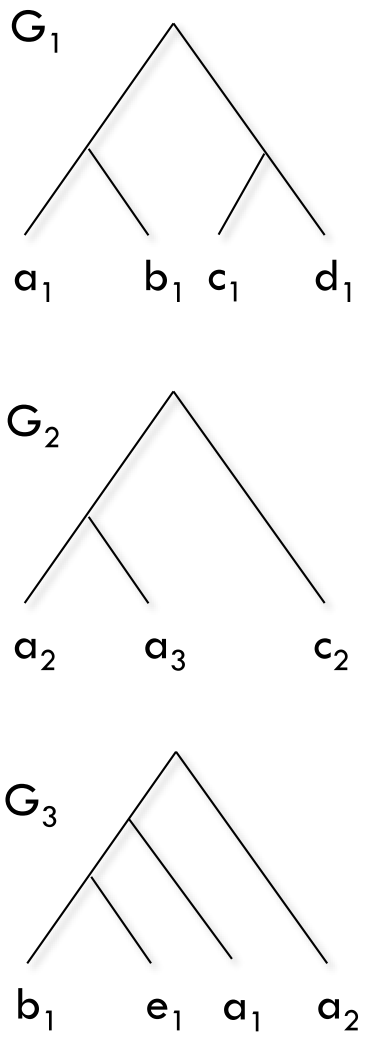


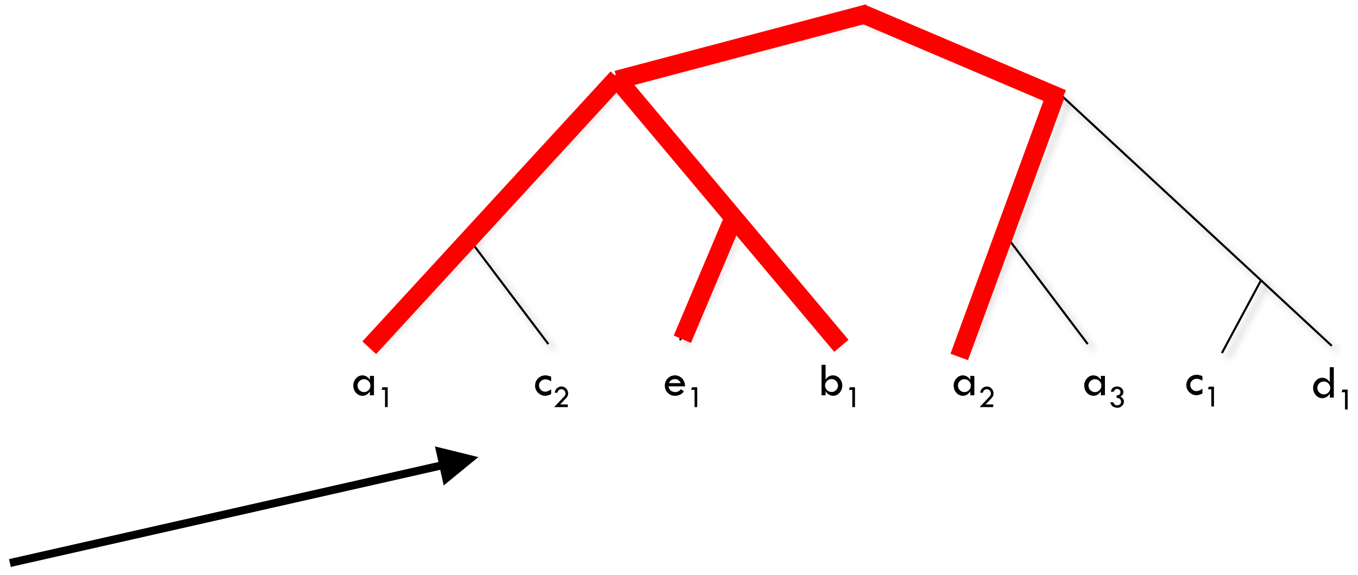


The Supergenetree problem

Multiple gene trees

- **Our goal : find a gene tree that displays them all**





SuperGeneTree

- Our trees are said **compatible** if there is a supertree displaying them all
- Finding a supertree (or determining incompatibility) is an old problem
  - The BUILD algorithm does that *(Aho & al., 1981)*
- What's different about super**gene**trees ?

SuperGeneTree

- Our trees are said **compatible** if there is a supertree displaying them all

- Finding a supertree (or determining incompatibility) is an old problem
  - The BUILD algorithm does that *(Aho & al., 1981)*

- What's different about super**gene**trees ?

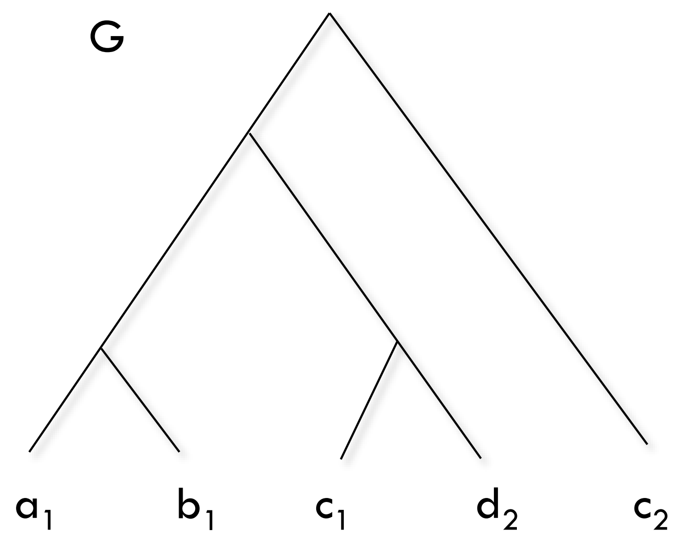
- We have the **species tree**

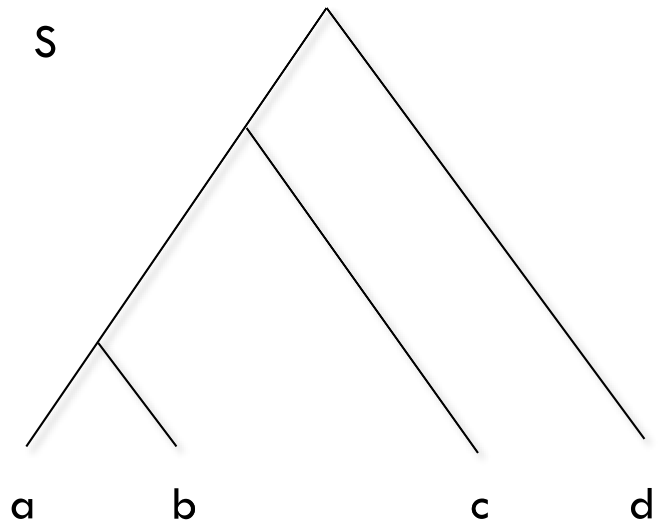
SuperGeneTree

- Often, many supergenetrees exist
- Which one is the best ?
- We **explore ways to choose** using information from the species tree S
- More specifically, we **explore ways to use reconciliation** with S to pick the best supergenetree

Reconciliation

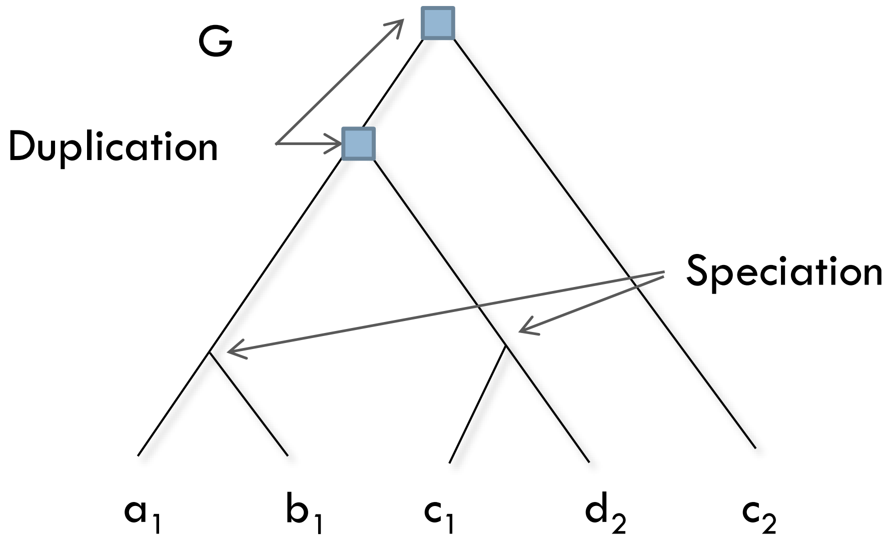
Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

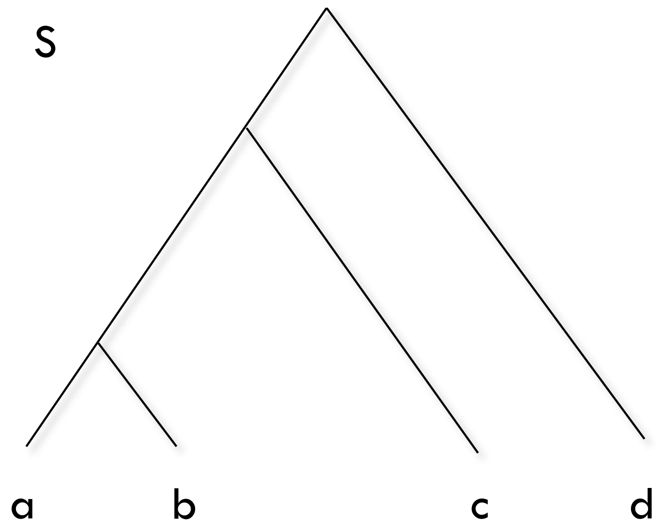




Reconciliation

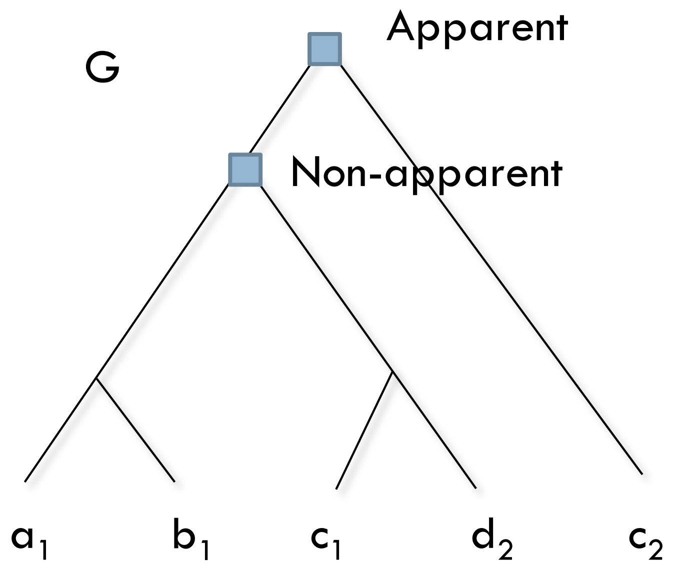
Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

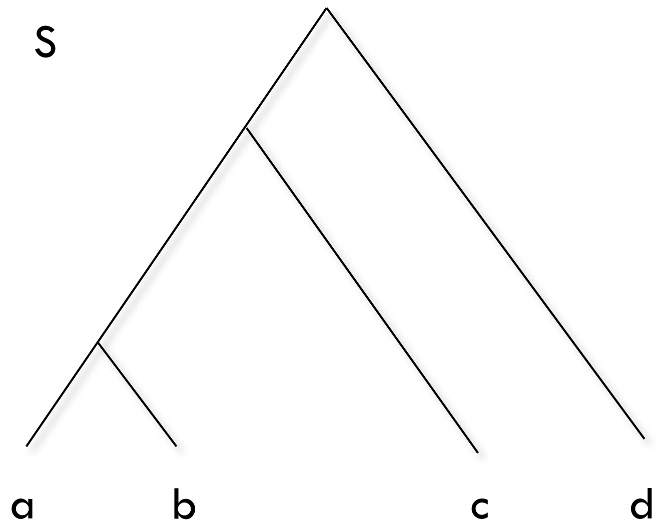




Reconciliation

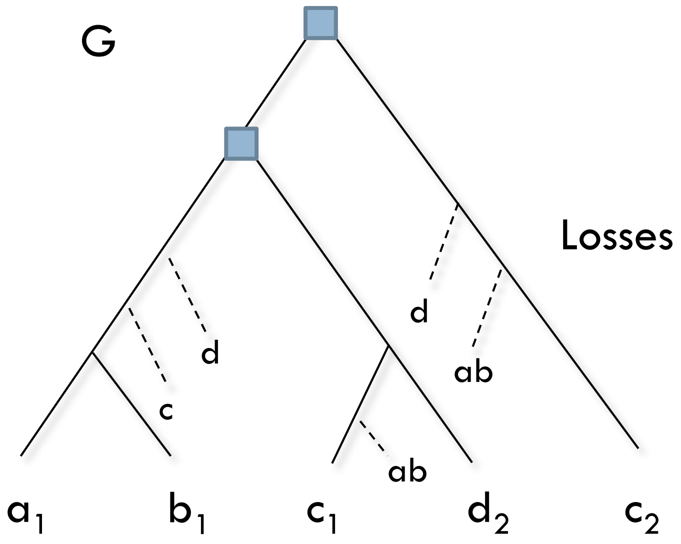
Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

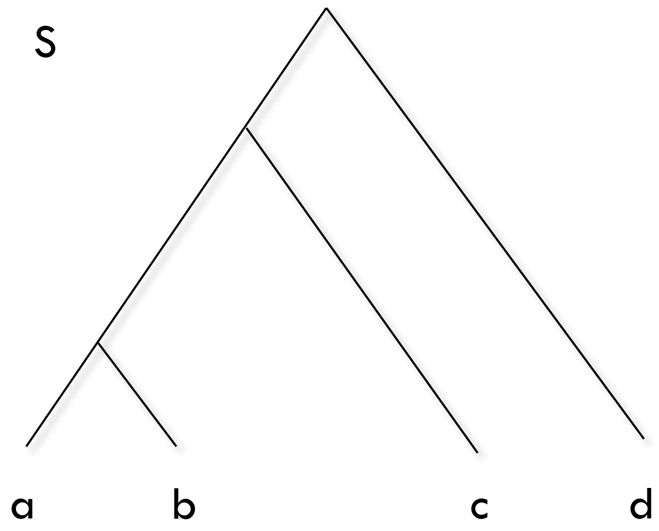




Reconciliation

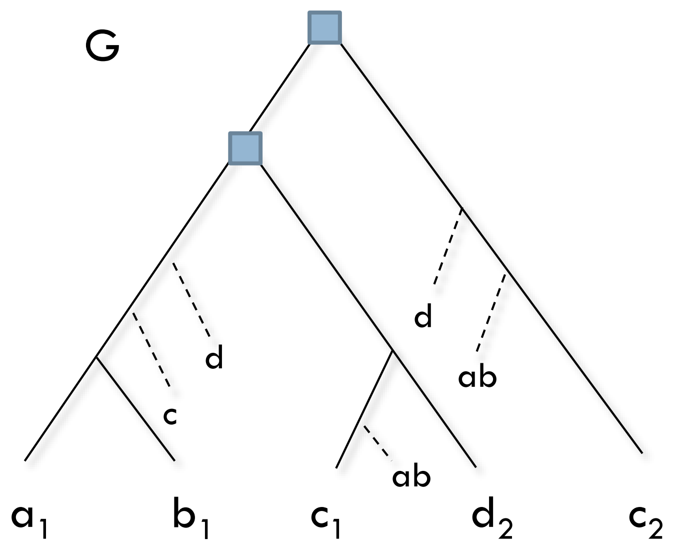
Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

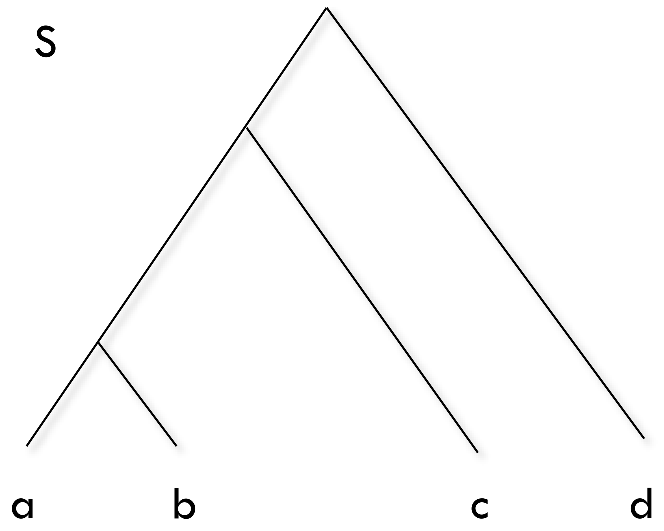




Reconciliation

Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

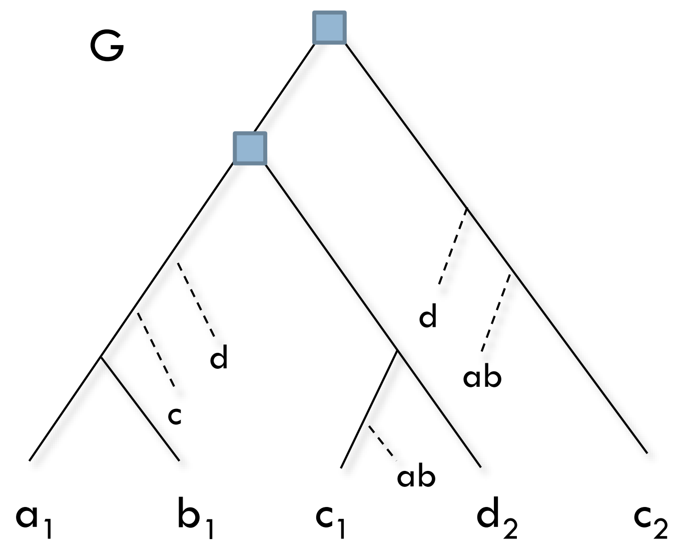


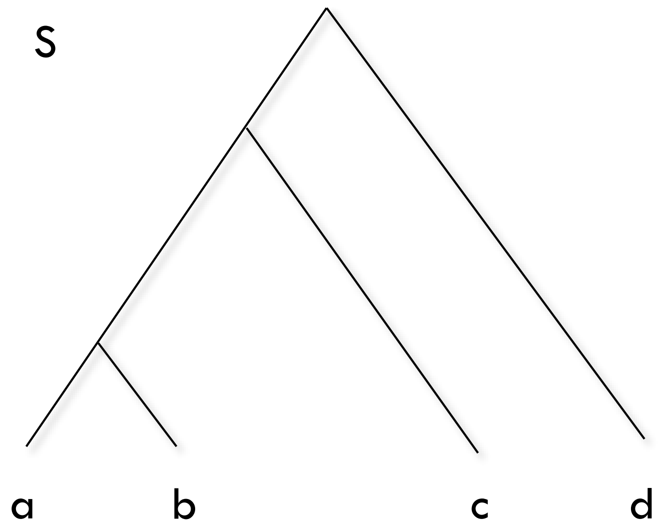


Possible reconciliation costs : #dups, #dups + #losses

Reconciliation

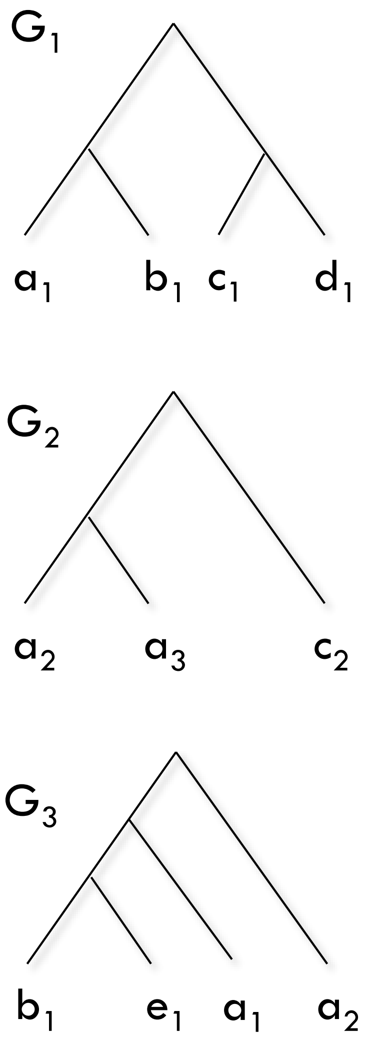
Reconciliation identifies **duplication**, **speciation** and **loss** events in G.

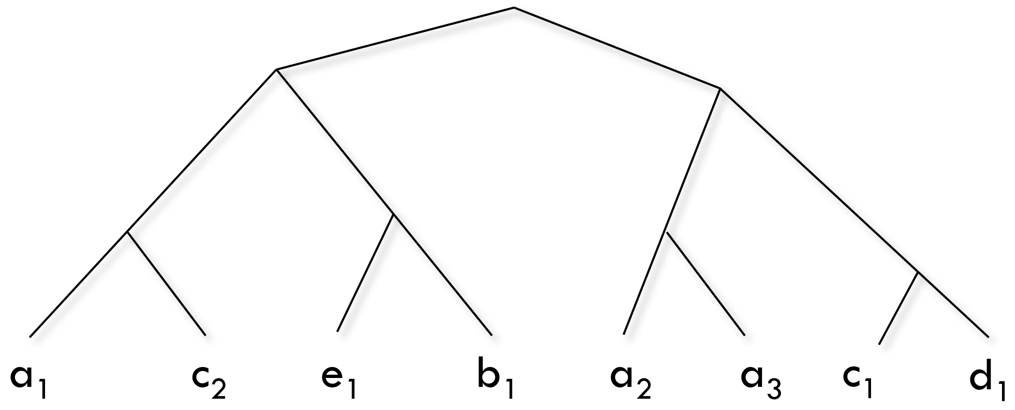




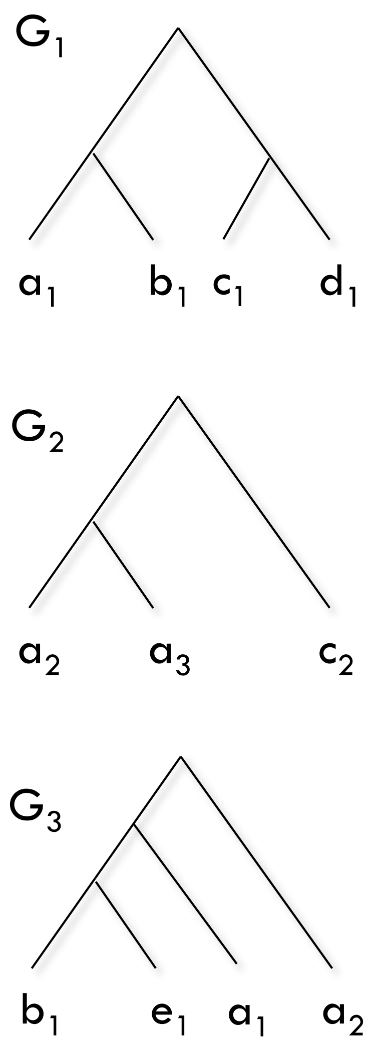
Possible reconciliation costs : **#dups**, #dups + #losses

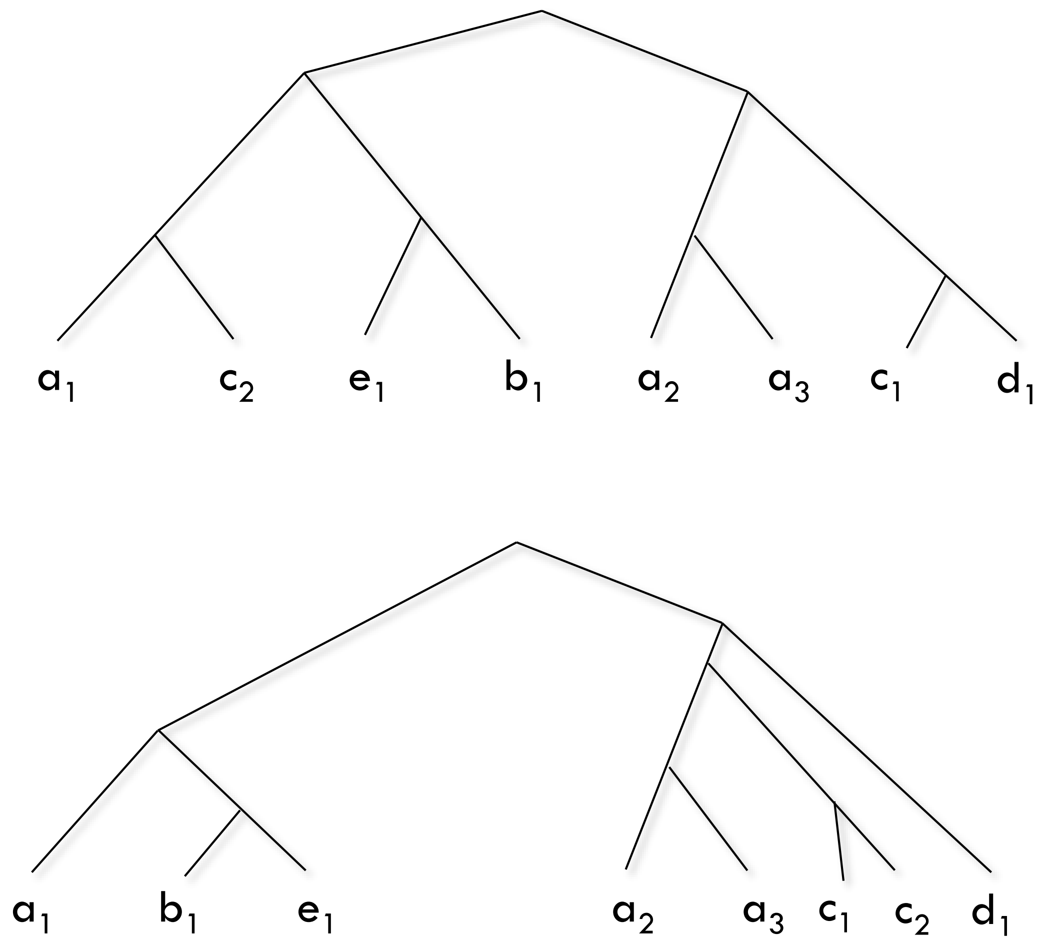
The Supergenetree problem



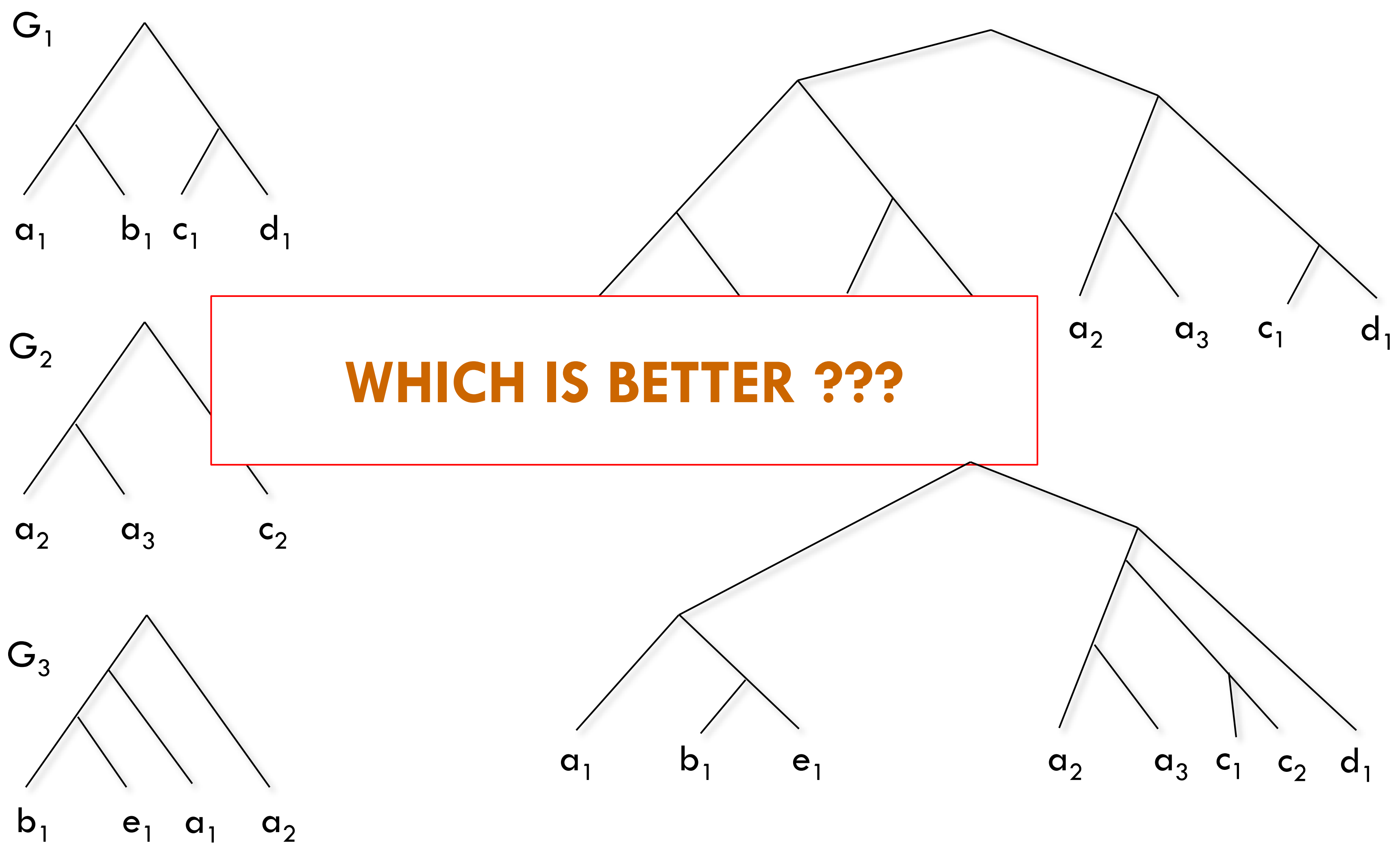


The Supergenetree problem



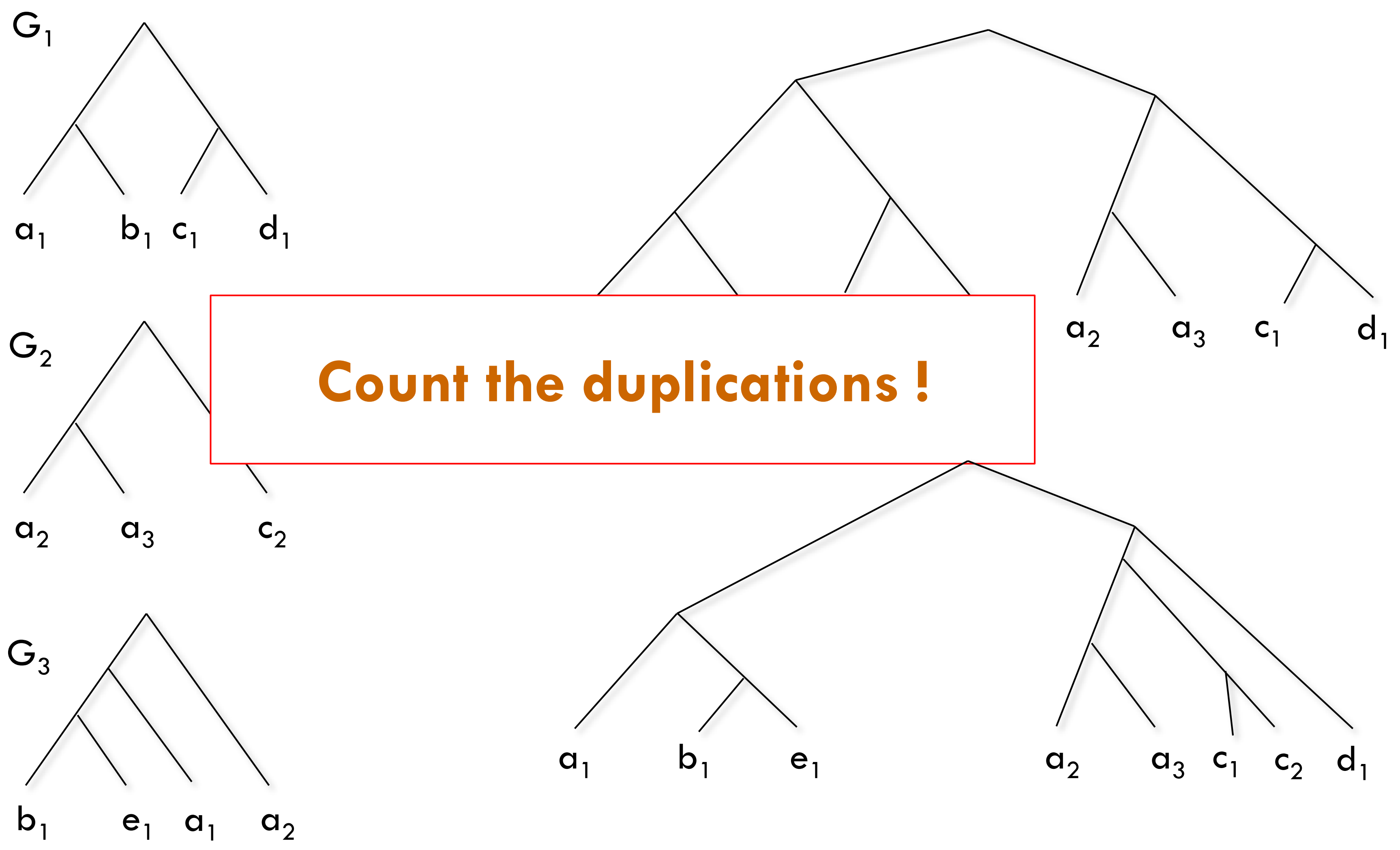


The Supergenetree problem



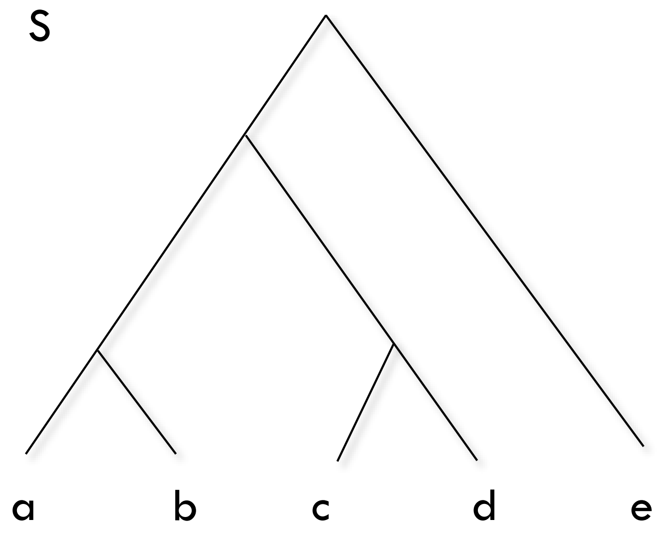
WHICH IS BETTER ???

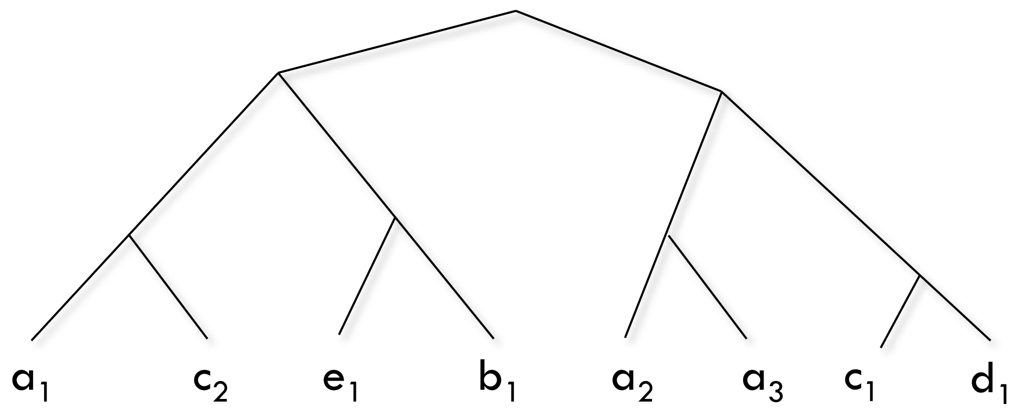
The Supergenetree problem

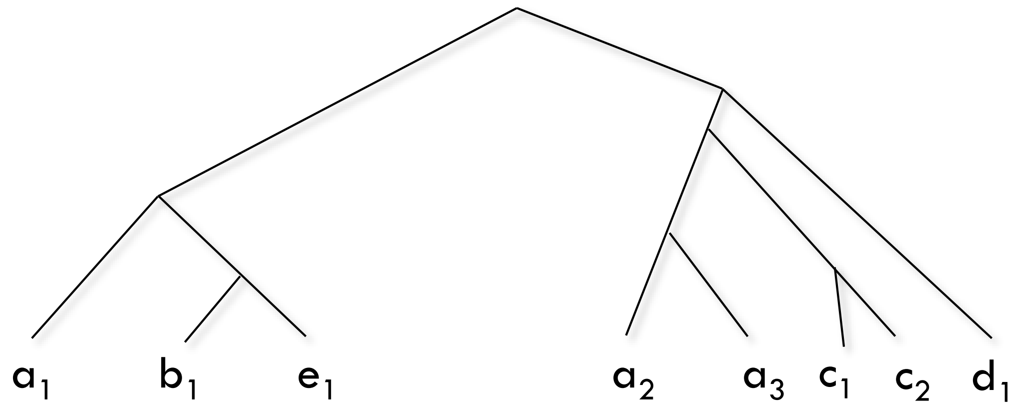


Count the duplications !

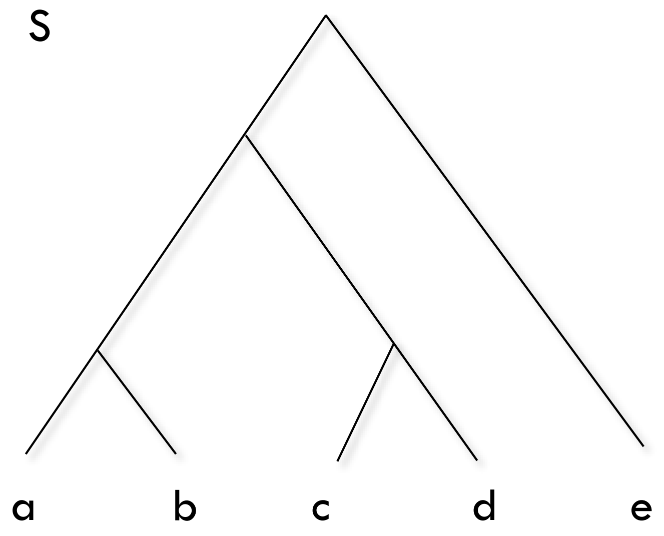
The Supergenetree problem

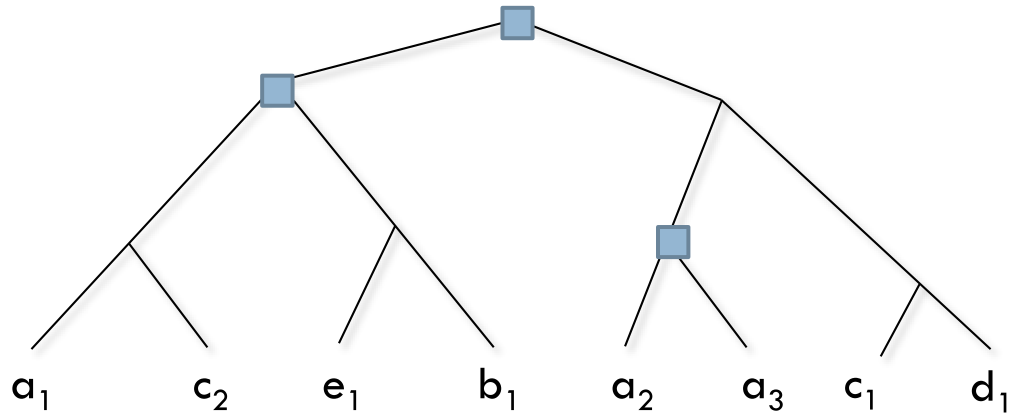


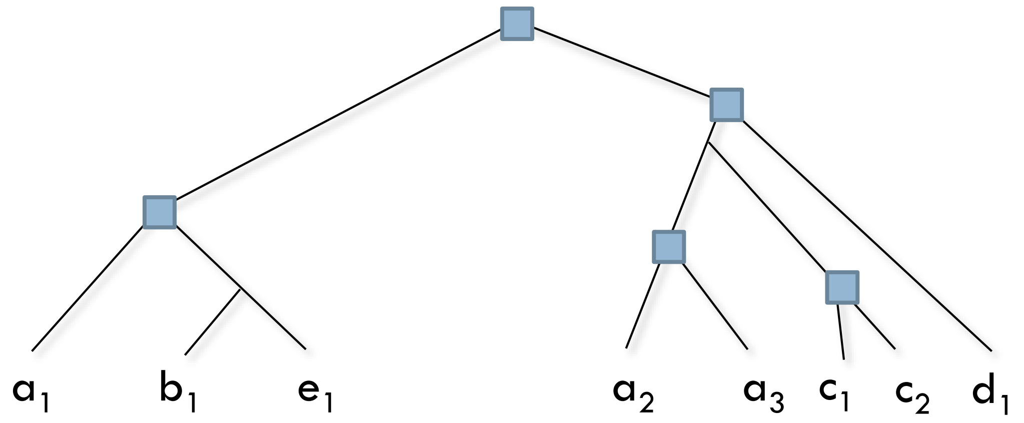




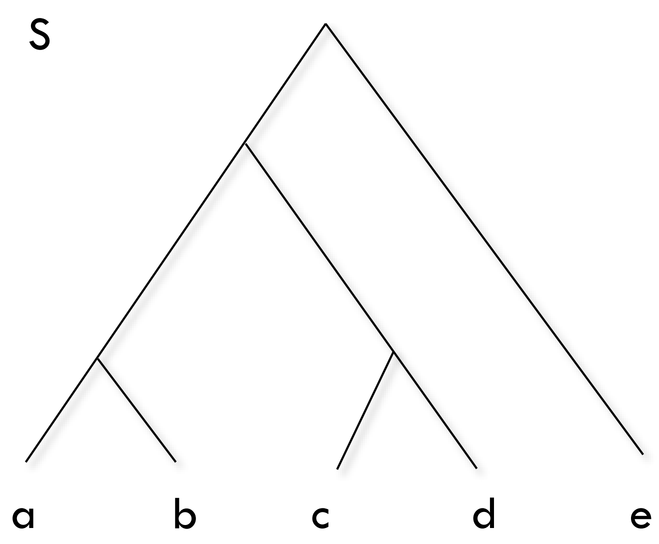
The Supergenetree problem

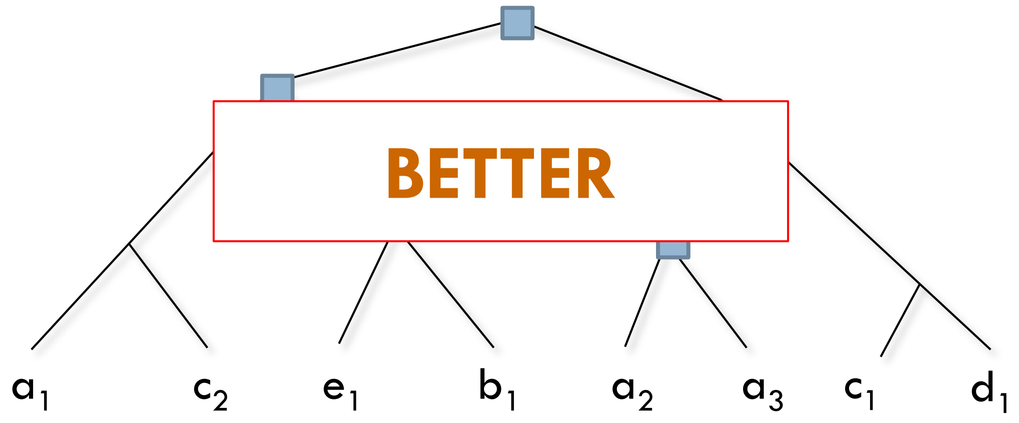


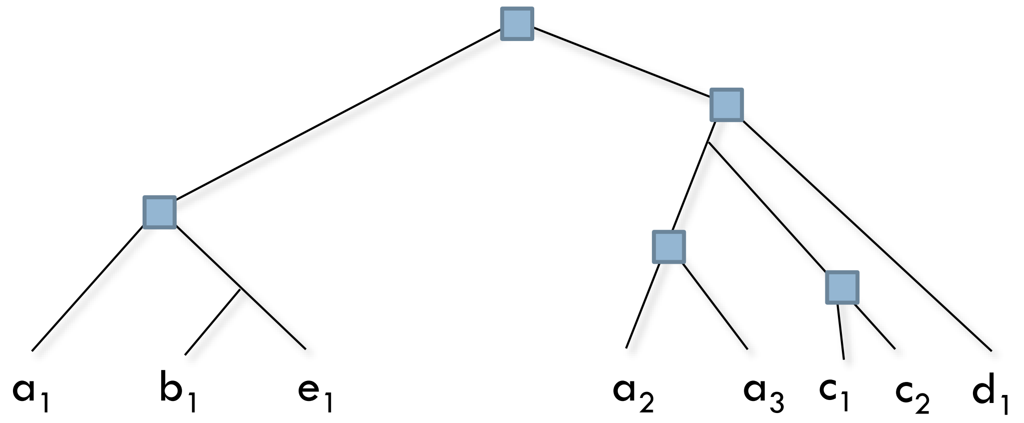




The Supergenetree problem







The plan

In this talk I…

- …come up with supertree problems
  - Finding a supergenetree that minimizes duplications
- …convince you that they're hard
- …try to do something about it
  - Exact, brute-force algorithm
  - A greedy heuristic

SuperGeneTree Problem 1

- **Given:** a set of compatible gene trees G = $\{G_1, \ldots, G_k\}$ and a species tree S
- **Find:** a SuperGeneTree G* that
  - displays every tree of G
  - minimizes #dups(G*, S)

SuperGeneTree Problem 1

- **Given:** a set of compatible gene trees G = $\{G_1, \ldots, G_k\}$ and a species tree S
- **Find:** a SuperGeneTree G* that
  - displays every tree of G
  - minimizes #dups(G*, S)

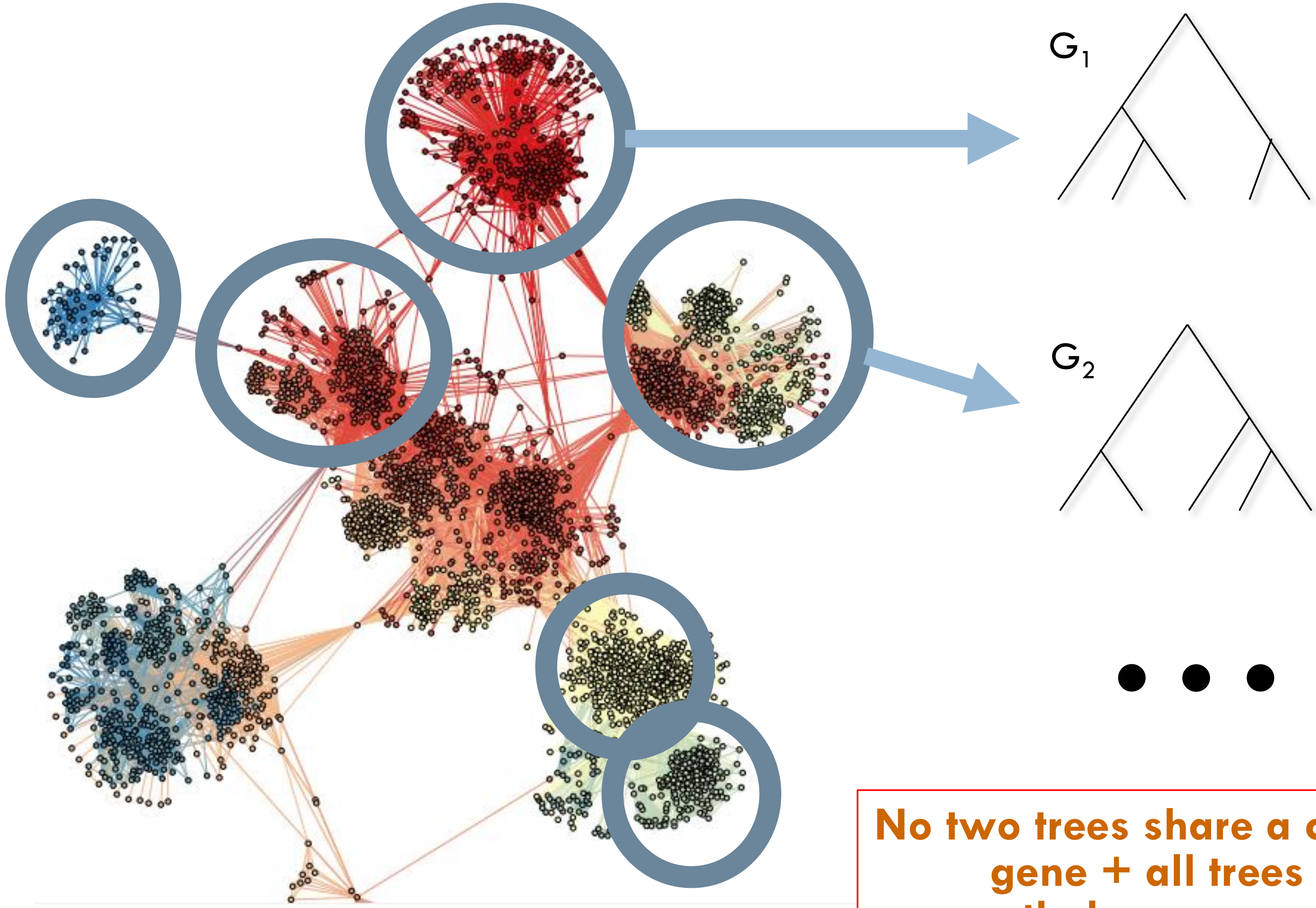
- NP-Complete

SuperGeneTree Problem 1

- **Given:** a set of compatible gene trees G = $\{G_1, \ldots, G_k\}$ and a species tree S
- **Find:** a SuperGeneTree G* that
  - displays every tree of G
  - minimizes #dups(G*, S)

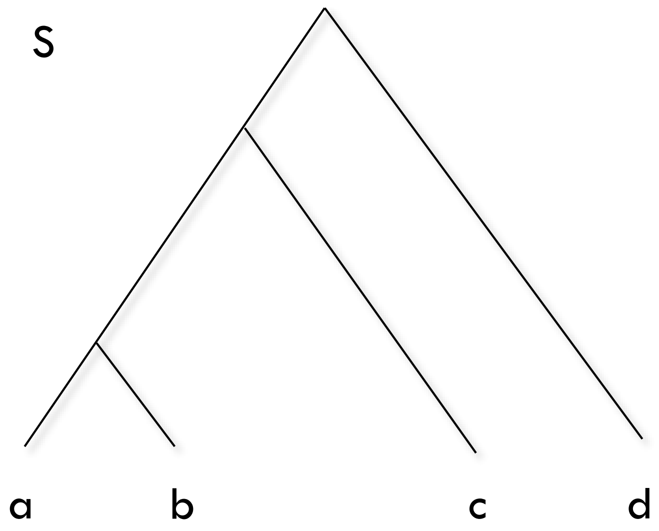
- NP-Complete
- NP-Hard to approximate within a $n^{1-\varepsilon}$ factor

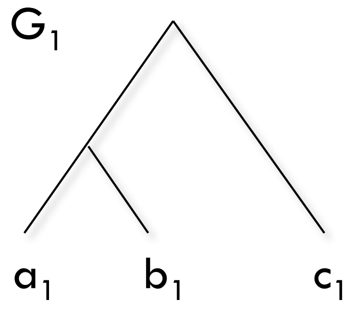
Independent speciation trees

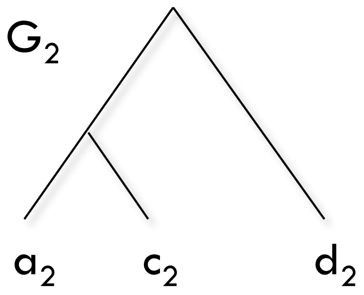


No two trees share a common gene + all trees of orthologous groups

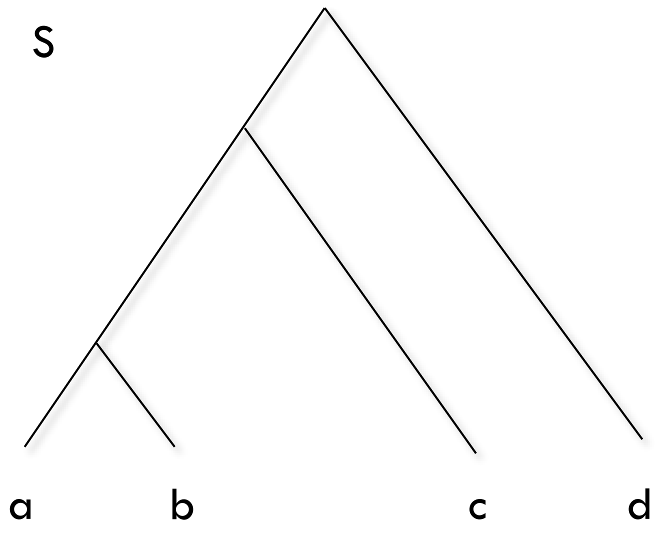
Independent speciation trees

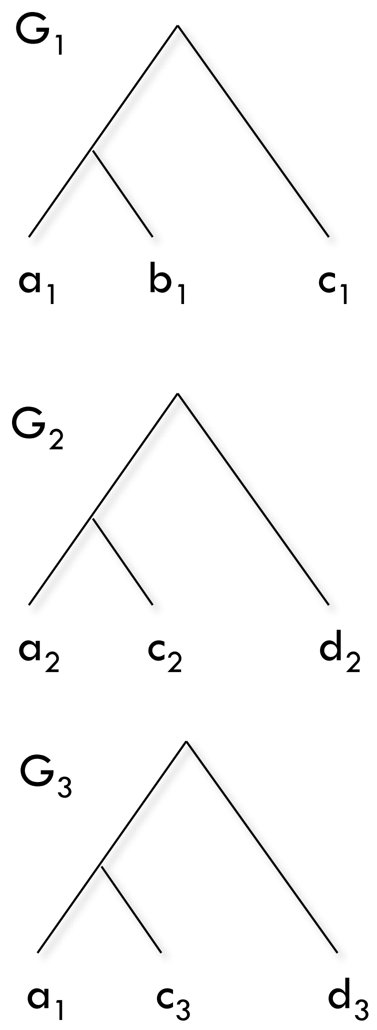




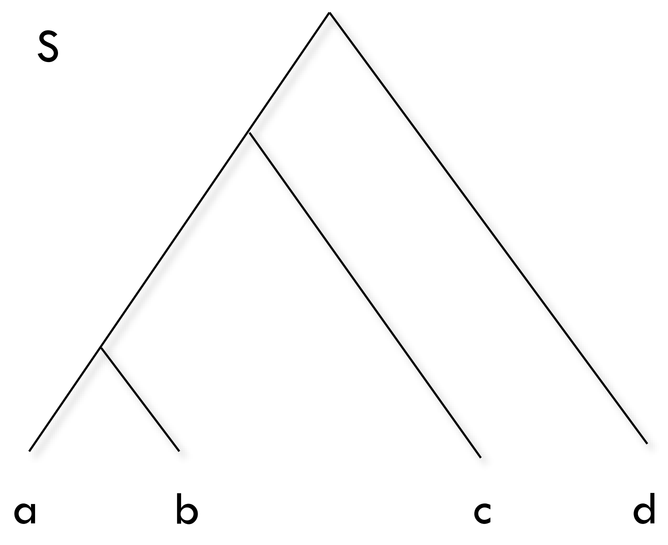


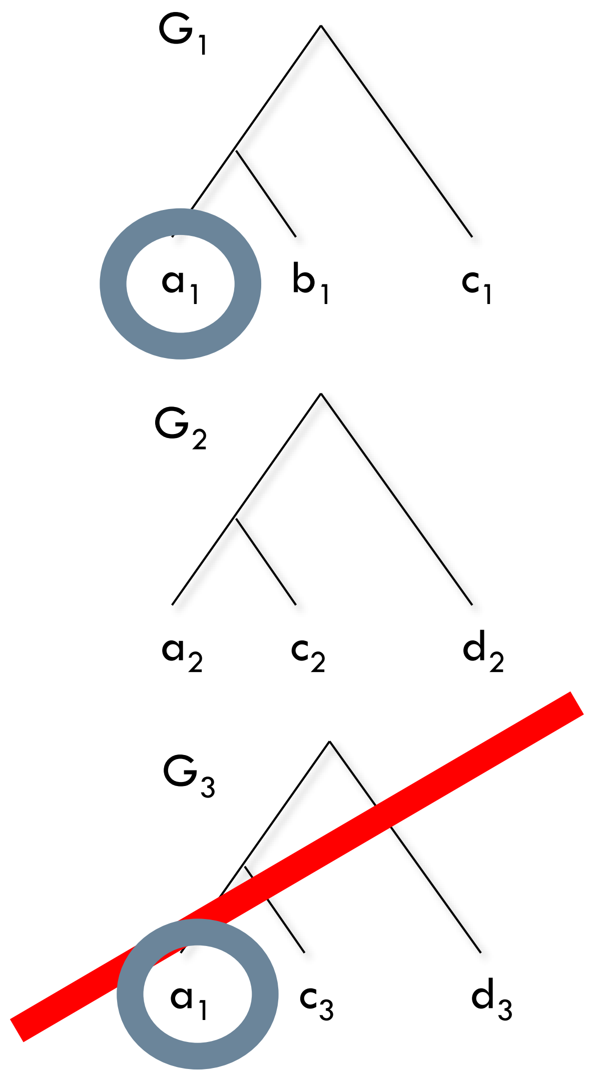
Independent speciation trees





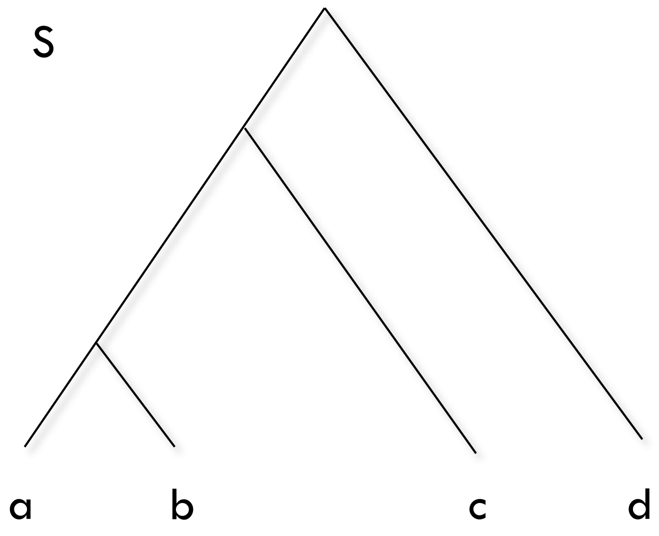
Independent speciation trees

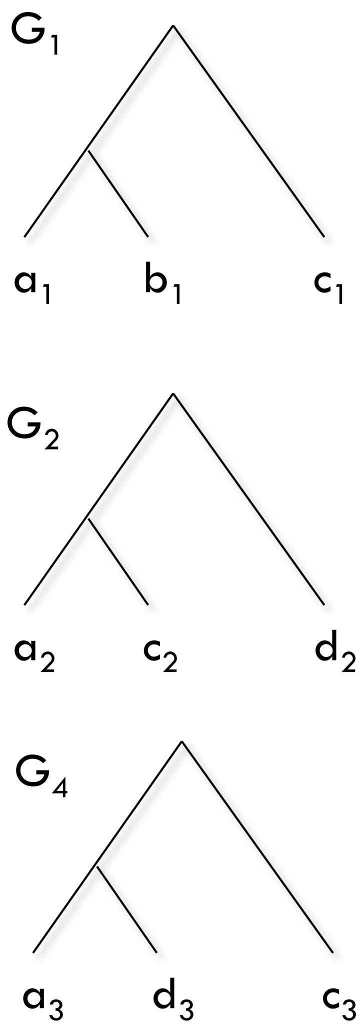




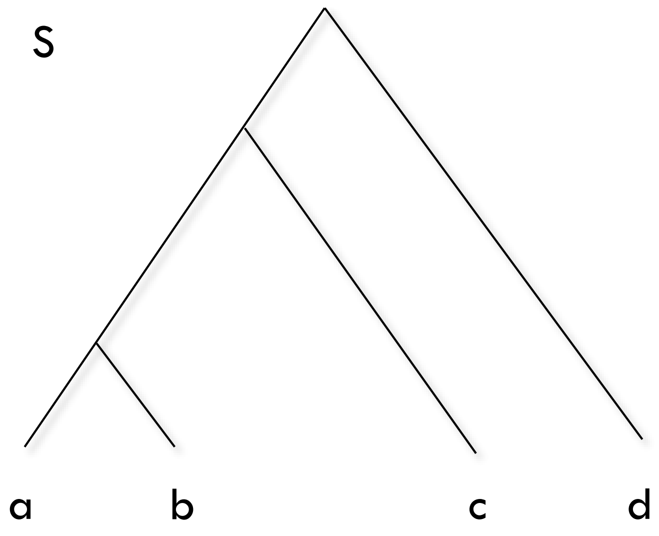
Independent = each gene appears only once

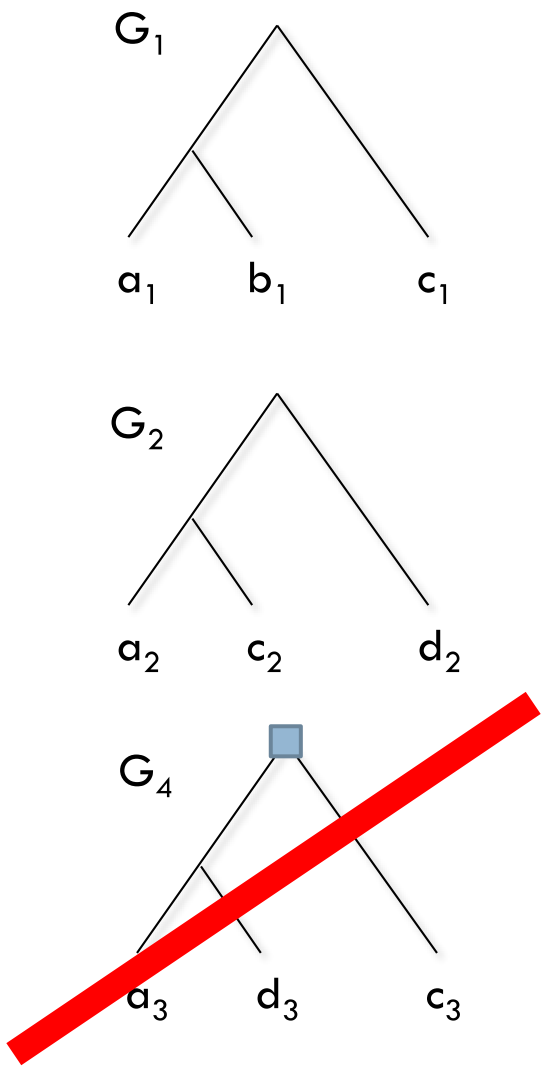
Independent speciation trees





Independent speciation trees





Speciation trees = all speciation (all agree with S)

SuperGeneTree Problem 2

- **Given:** a set of **independent speciation** gene trees $G = \{G_1, \ldots, G_k\}$ and a species tree S
- **Find:** a SuperGeneTree G* that
  - displays every tree of G
  - minimizes #dups(G*, S)

SuperGeneTree Problem 2

- **Given:** a set of **independent speciation** gene trees $G = \{G_1, \ldots, G_k\}$ and a species tree S
- **Find:** a SuperGeneTree G* that
  - displays every tree of G
  - minimizes #dups(G*, S)

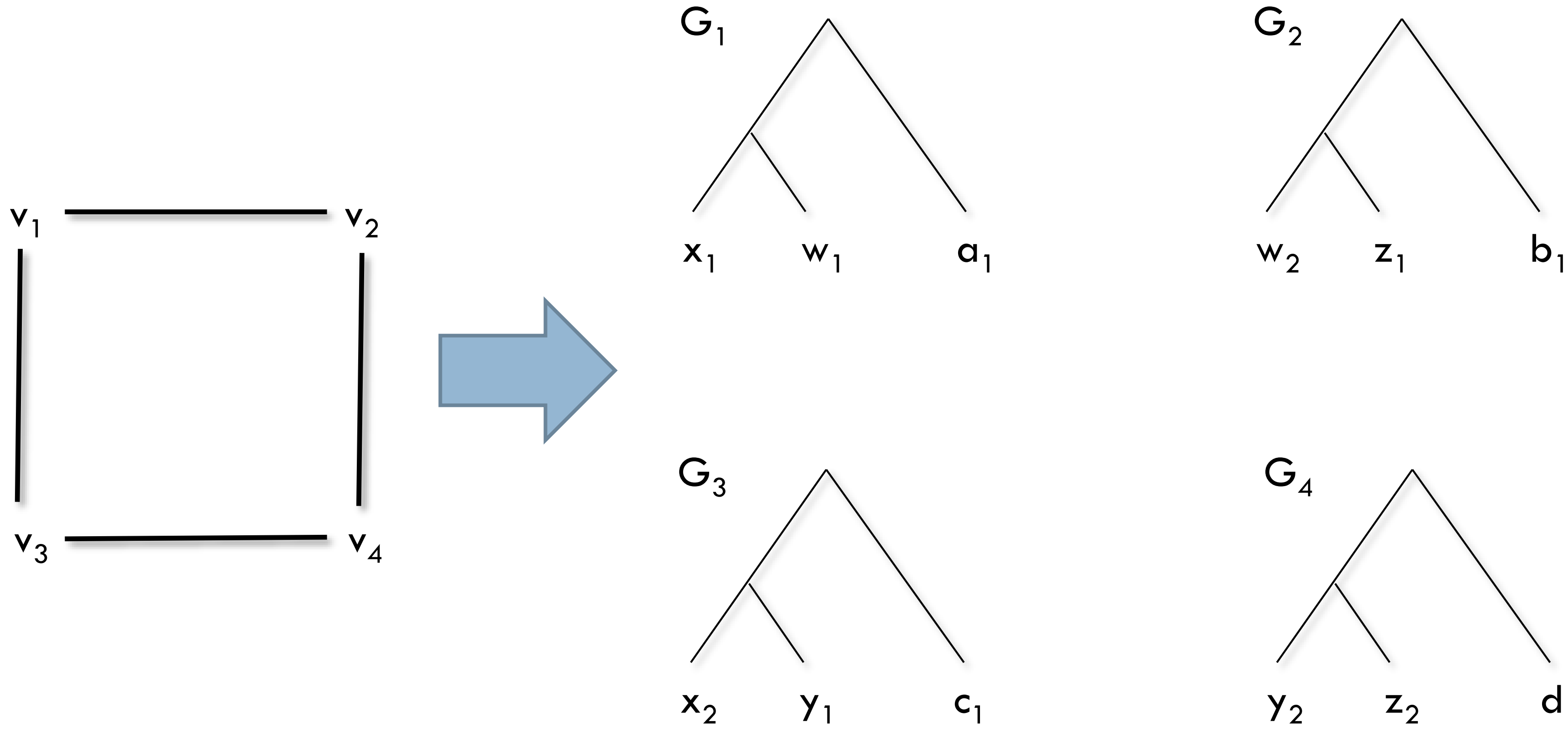
- NP-Complete

The plan

In this talk I…

- …come up with supertree problems
  - Finding a supergenetree that minimizes duplications
- …convince you that they're hard
- …try to do something about it
  - Exact, brute-force algorithm
  - A greedy heuristic

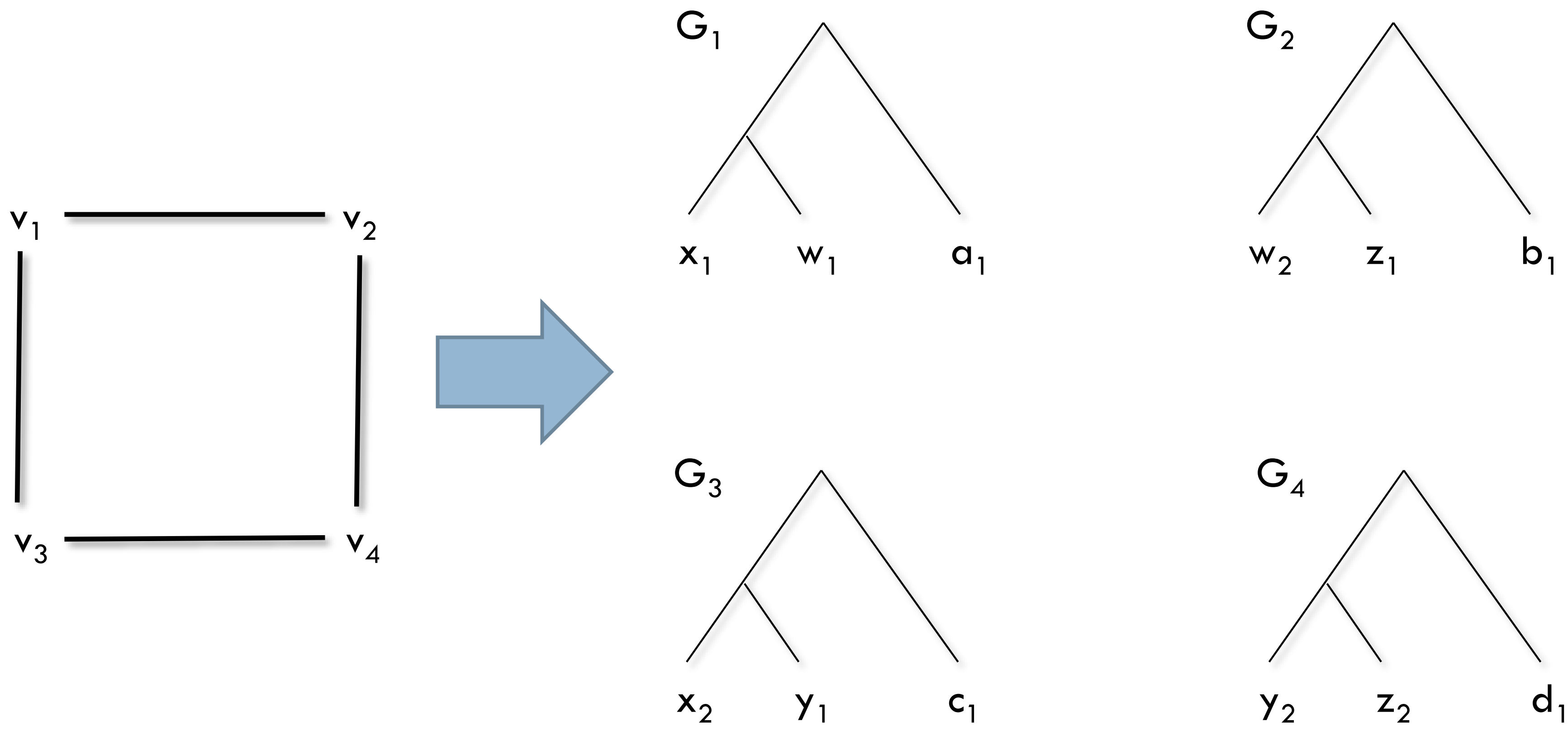
What is so hard about it ?



We will find a vertex-coloring of our graph

(a partition into independent sets)

What is so hard about it ?

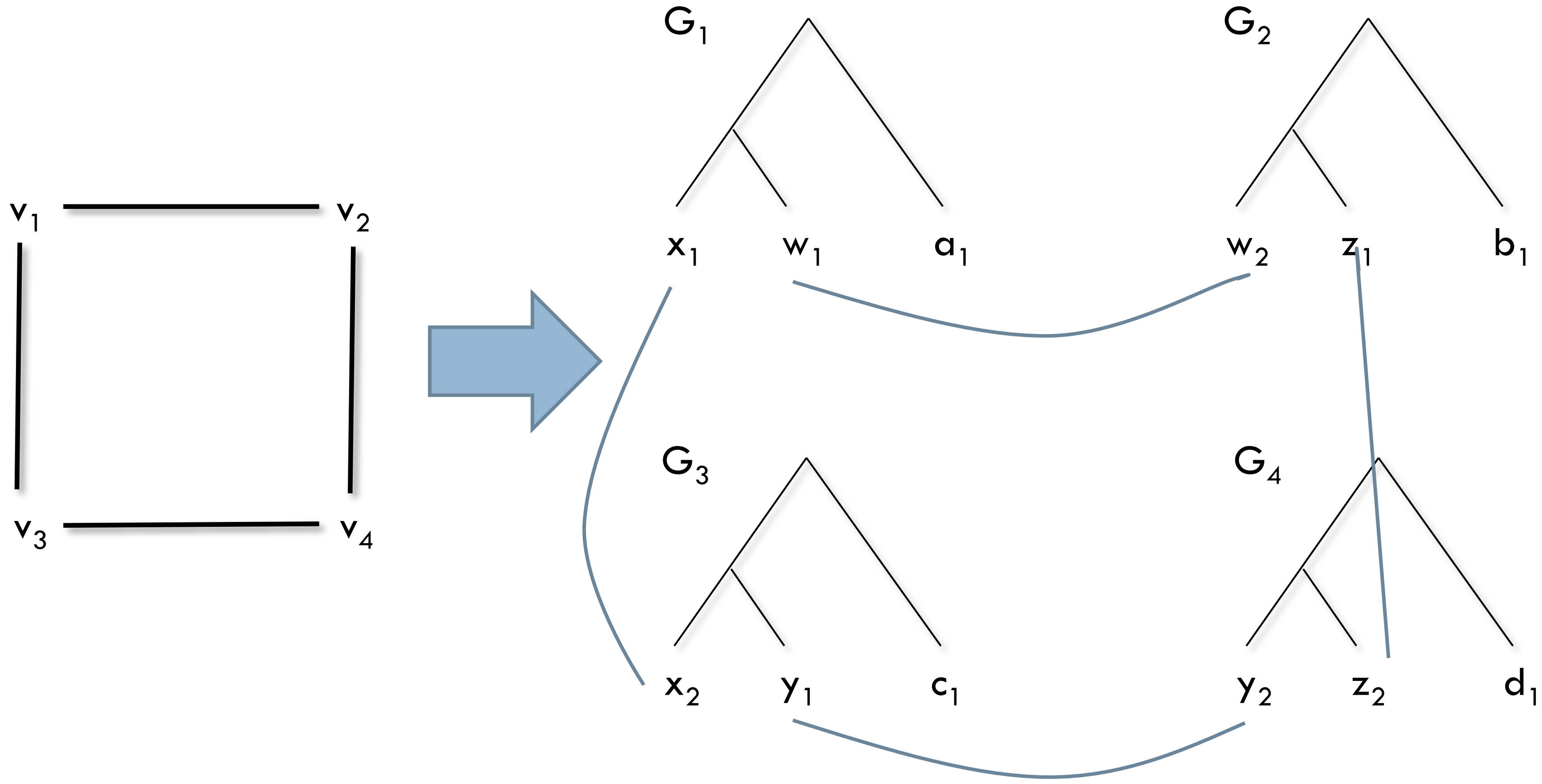


G_i, G_j **share a gene from the same species (i.e. a label)** iff v_i, v_j **share an edge**

$\Leftrightarrow$

G_i, G_j can be merged into a supertree **without duplications** iff v_i, v_j **share no edge**

What is so hard about it ?

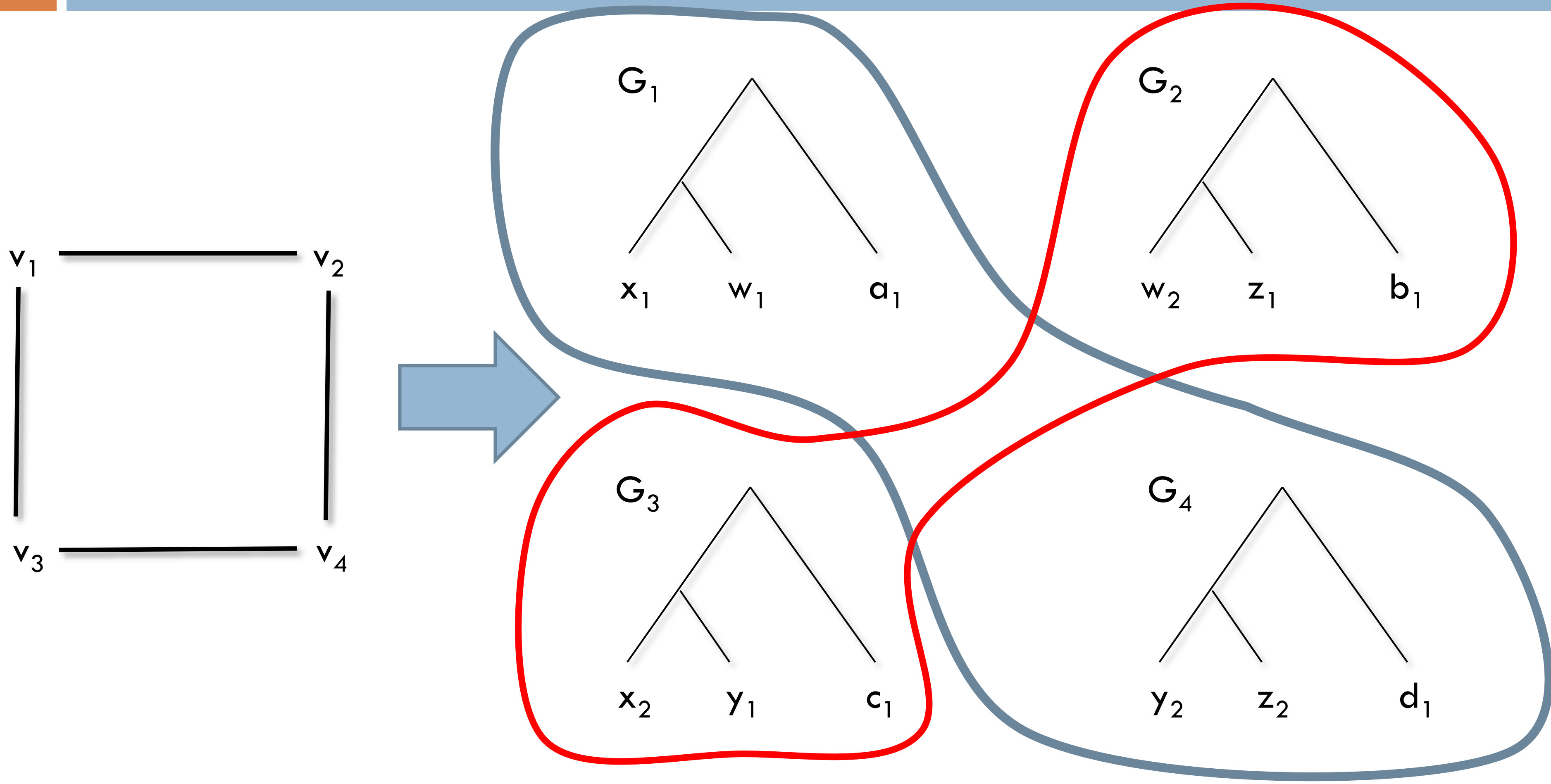


G_i, G_j **share a gene from the same species (i.e. a label)** iff v_i, v_j **share an edge**

$\Leftrightarrow$

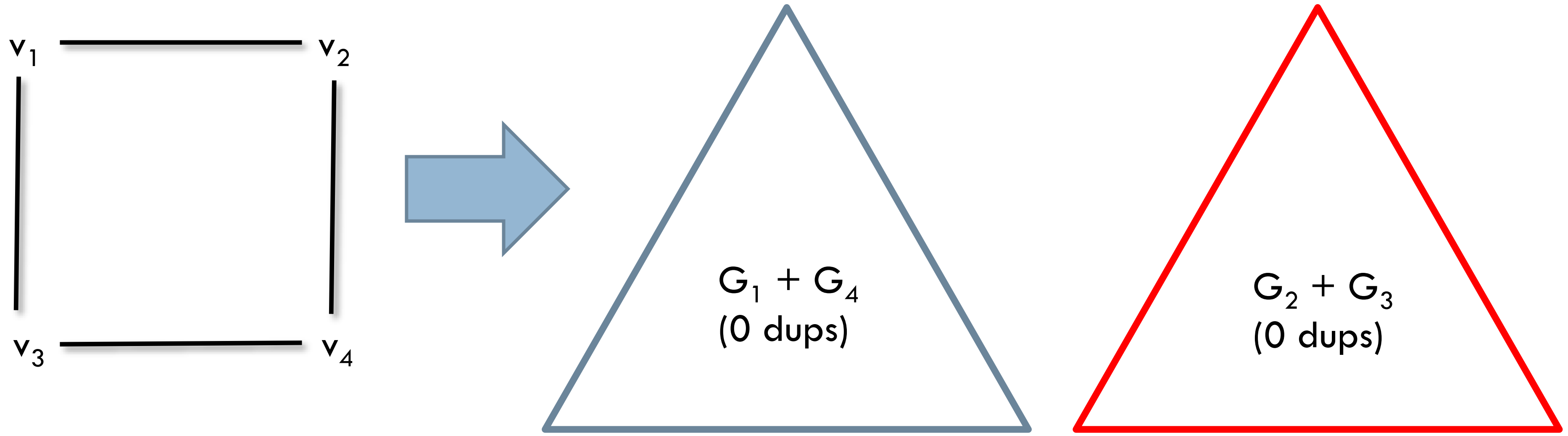
G_i, G_j can be merged into a supertree **without duplications** iff v_i, v_j **share no edge**

What is so hard about it ?



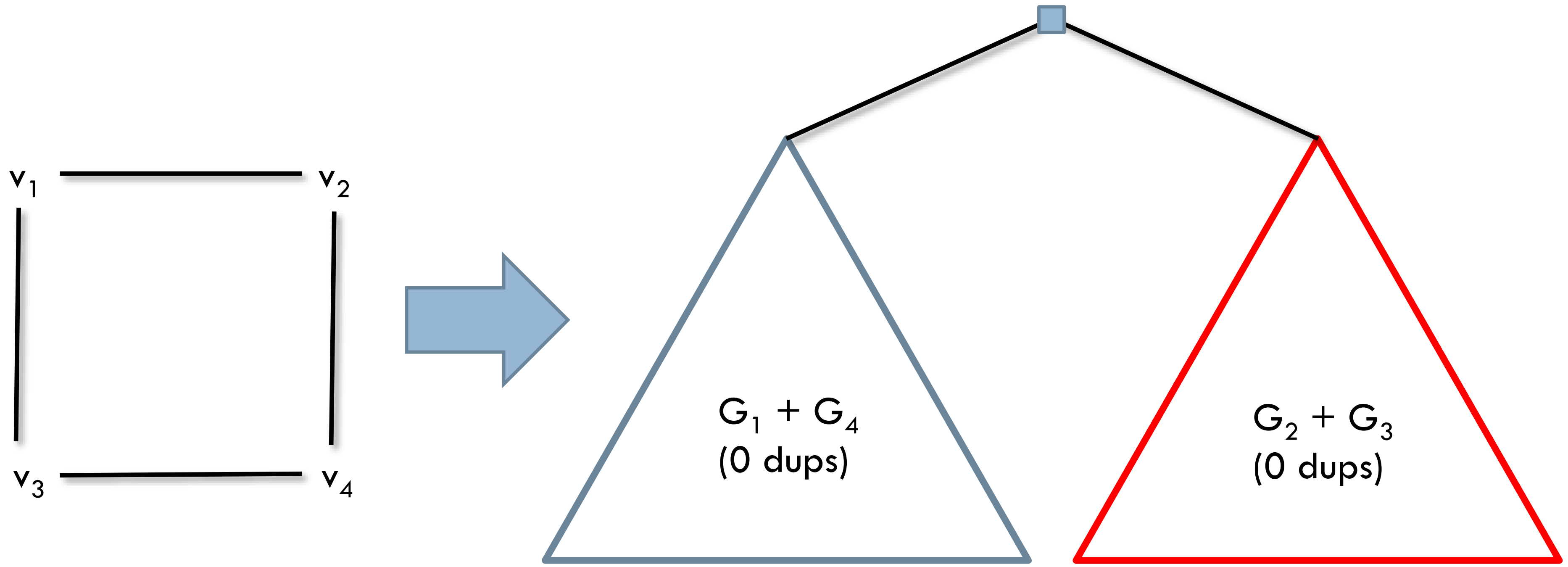
A best solution partitions the trees into **k** sets of trees that **all share no "label"**

What is so hard about it ?



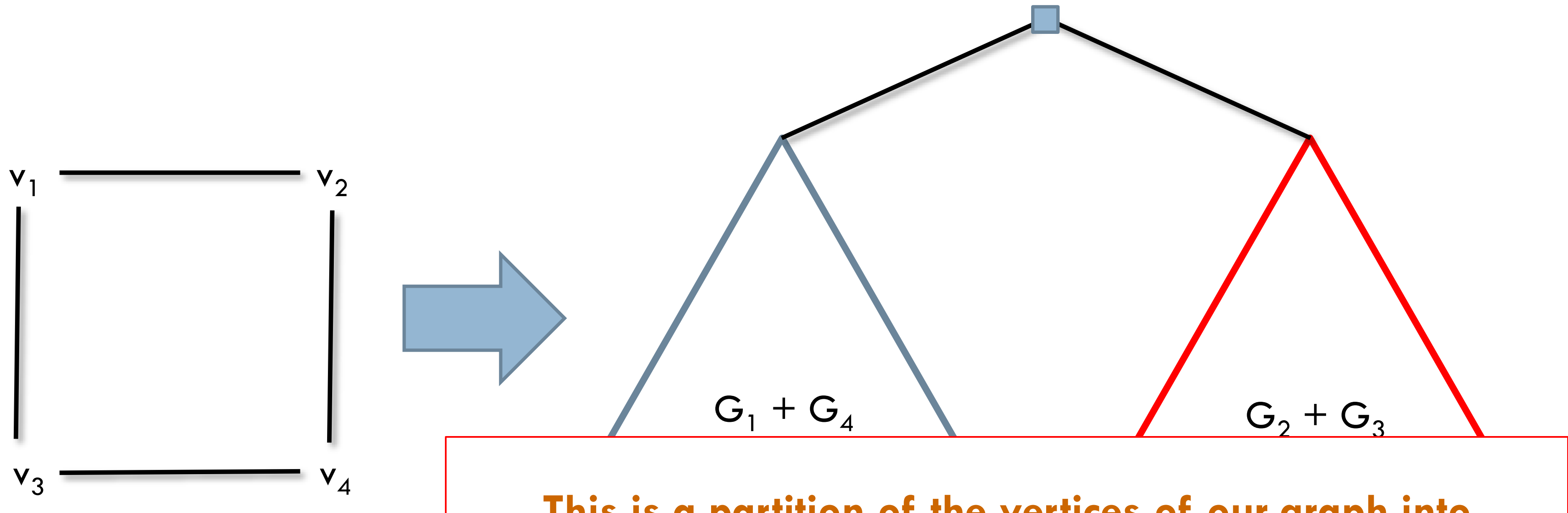
A best solution partitions the trees into **k** sets of trees that **all share no "label"**
Makes one zero-duplication tree for each part.

What is so hard about it ?



A best solution partitions the trees into **k** sets of trees that **all share no "label"**
Makes one zero-duplication tree for each part.
Connects these k subtrees with at most k – 1 duplications.

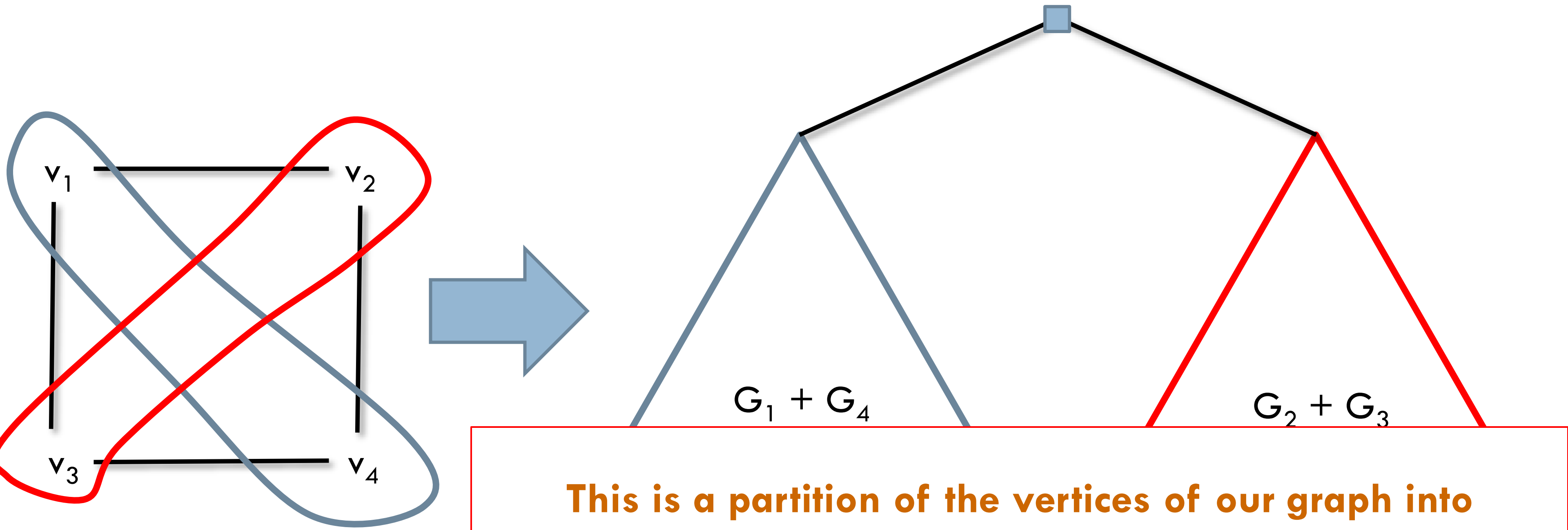
What is so hard about it ?



This is a partition of the vertices of our graph into independent sets, i.e. a vertex-coloring !

A best solution partitions the trees into **k** sets of trees that **all share no "label"**
Makes one zero-duplication tree for each part.
Connects these k subtrees with at most k – 1 duplications.

What is so hard about it ?



This is a partition of the vertices of our graph into independent sets, i.e. a vertex-coloring !

A best solution partitions the trees into **k** sets of trees that **all share no "label"**
Makes one zero-duplication tree for each part.
Connects these k subtrees with at most k – 1 duplications.

The plan

In this talk I…

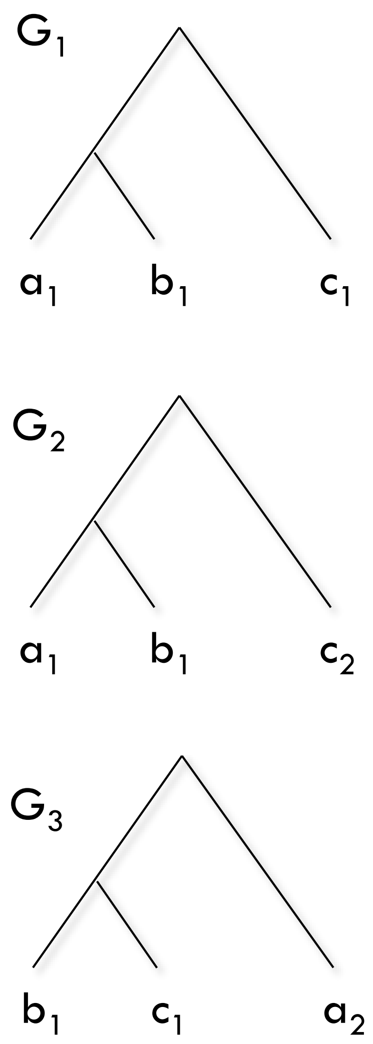
- …come up with supertree problems
  - Finding a supergenetree that minimizes duplications
- …convince you that they're hard
- …try to do something about it
  - Exact, brute-force algorithm
  - A greedy heuristic

Extending the BUILD algorithm

- Given a set of trees G, the BUILD algorithm outputs, if it exists, a supertree T displaying every tree of G
  - T might be partially resolved (non-binary)
  - Every binary resolution of T displays G

- BUILD can be extended to output **every** supertree displaying G + every minimally resolved *(Constantinescu & Sankoff, 1995, Ng & Wormald, 1996, Semple, 2003)*

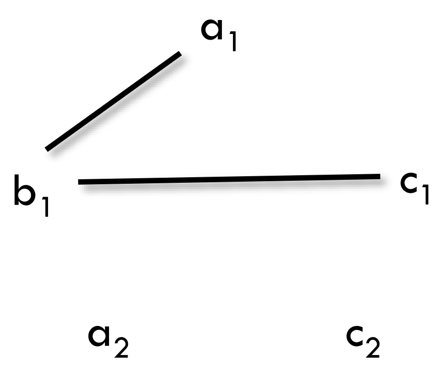
Extending the BUILD algorithm



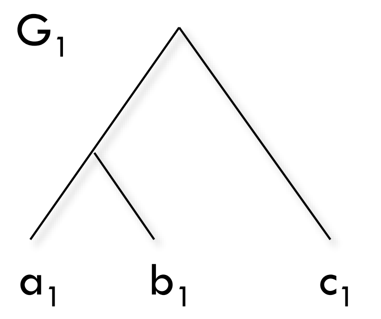
BUILD graph

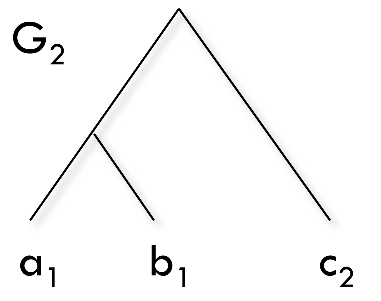
vertices = genes

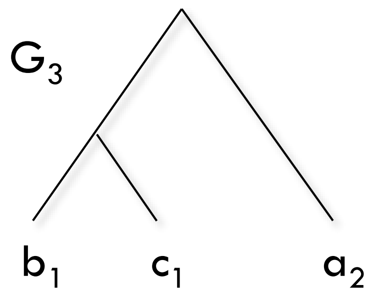
edges = genes together in some triplet



Extending the BUILD algorithm



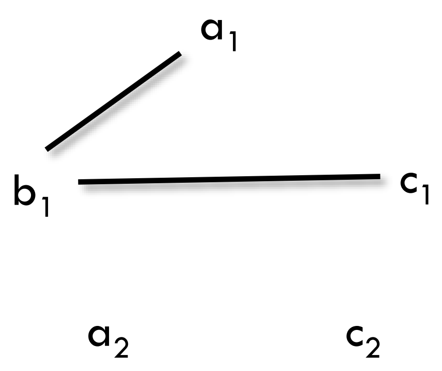




BUILD graph

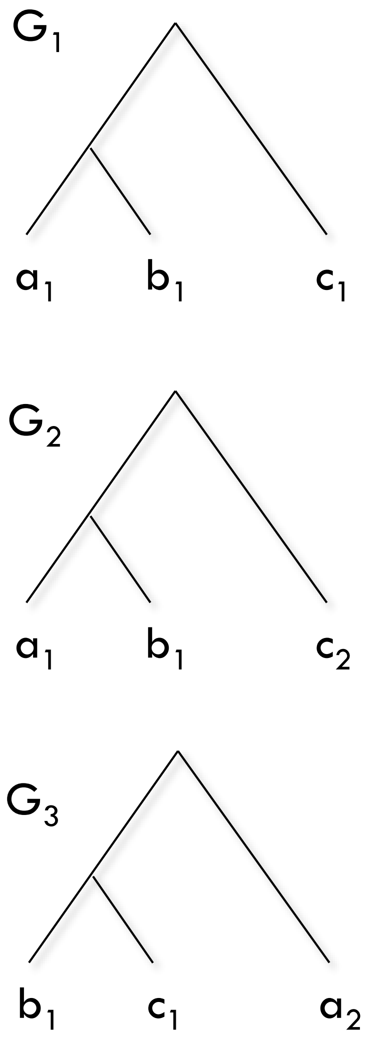
vertices = genes

edges = genes together in some triplet



Partition of connected components = possible splits at the root

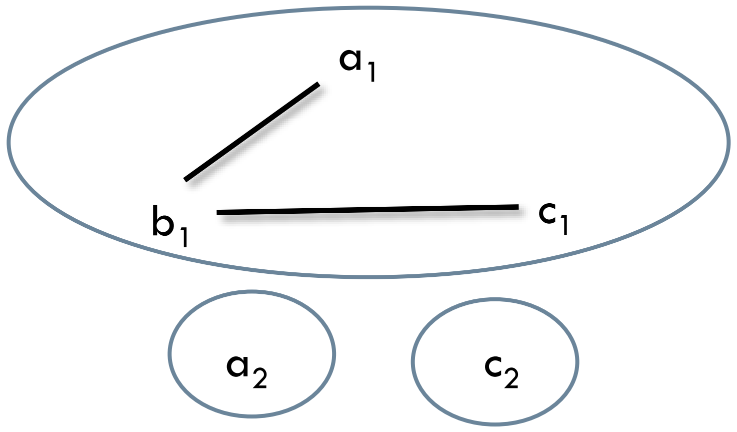
Extending the BUILD algorithm



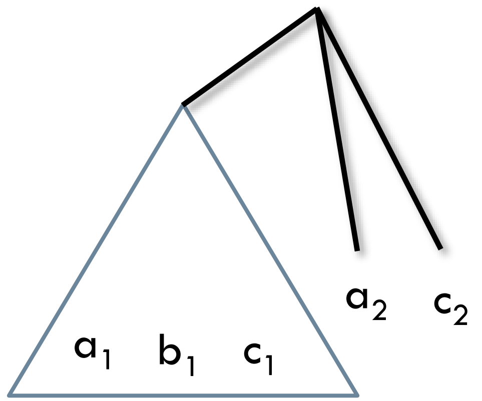
BUILD graph

vertices = genes

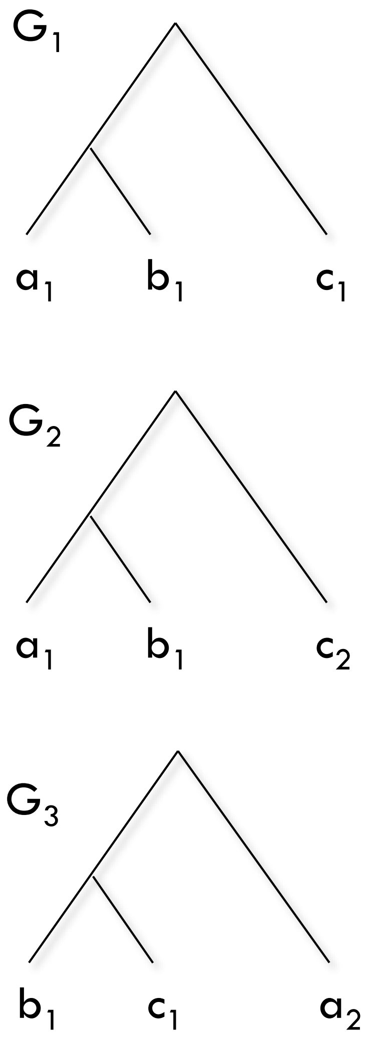
edges = genes together in some triplet



Partition of connected components = possible splits at the root



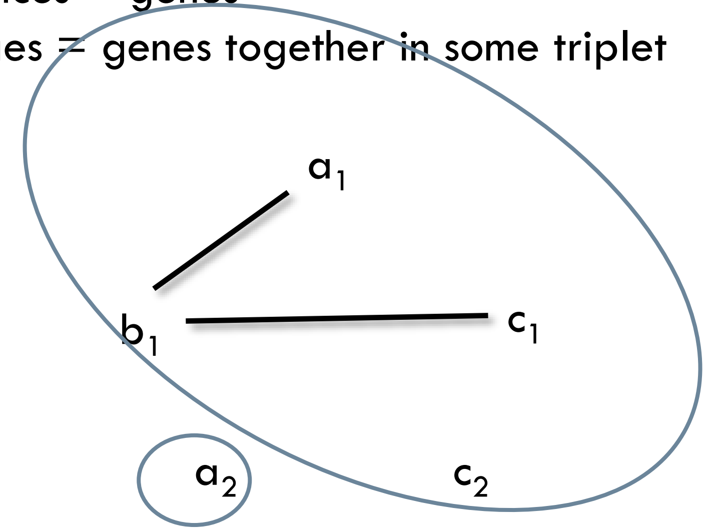
Extending the BUILD algorithm



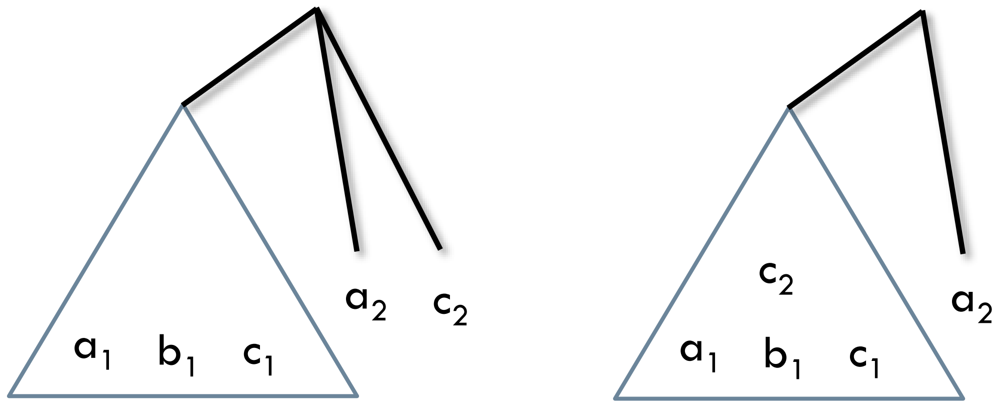
BUILD graph

vertices = genes

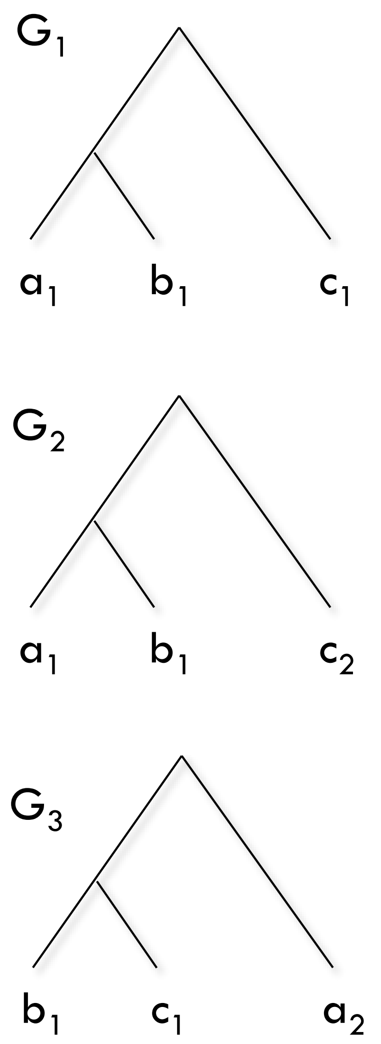
edges = genes together in some triplet



Partition of connected components = possible splits at the root



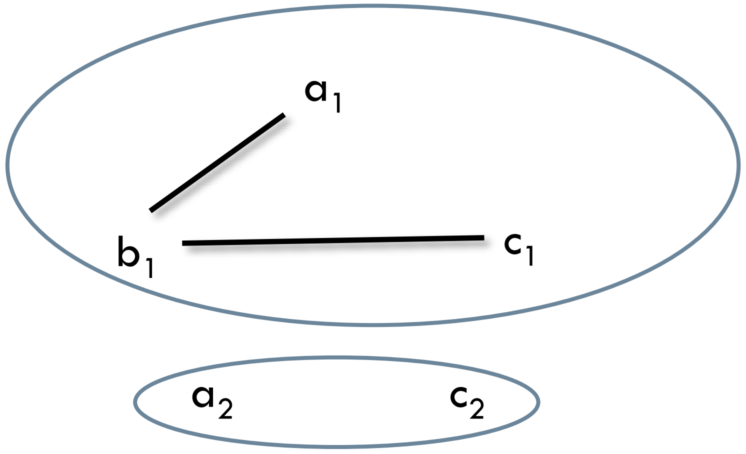
Extending the BUILD algorithm



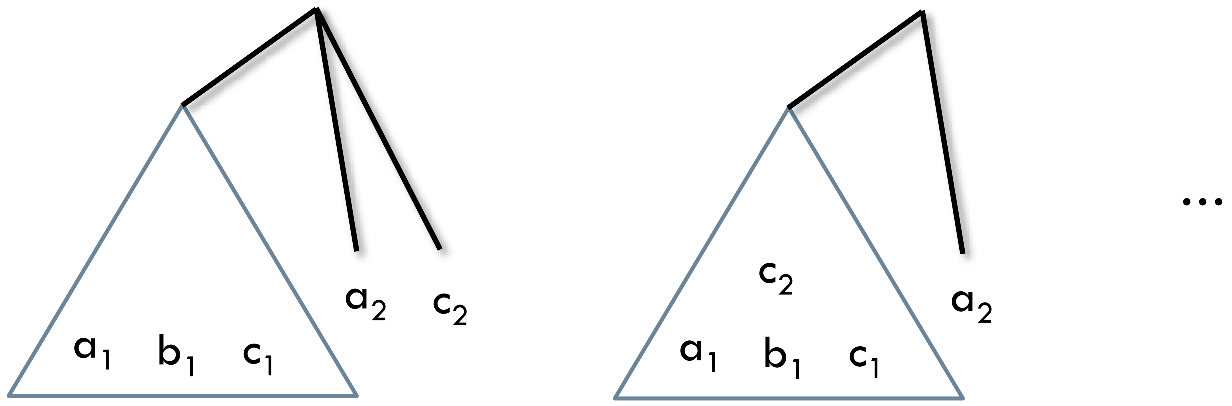
BUILD graph

vertices = genes

edges = genes together in some triplet



Partition of connected components = possible splits at the root



Extending the BUILD algorithm

- For every partially unresolved tree T obtained in this fashion :
  - Find a resolution that minimizes the number of duplications *(linear time, Lafond & al. 2012)*

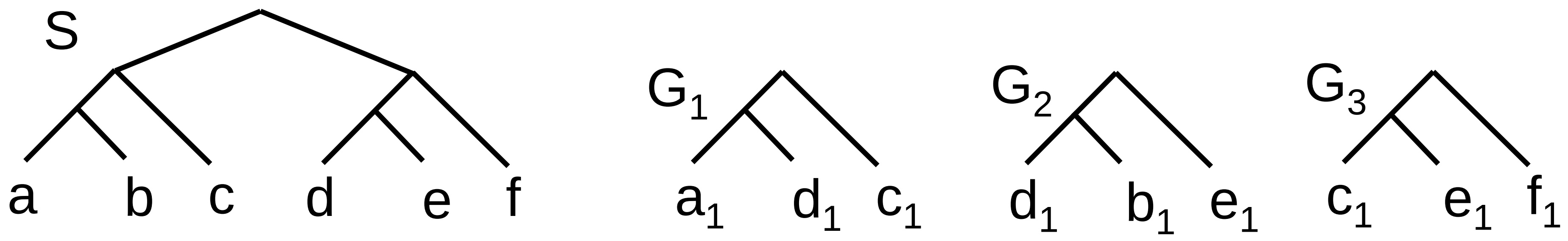
- In the worst case, there are $\Omega(n^{n/2})$ trees to resolve *(Jansson, Lemence, Lingas, 2012)*.
  - Total time : $\Omega(n * n^{n/2})$

- Worst case in practice : ?

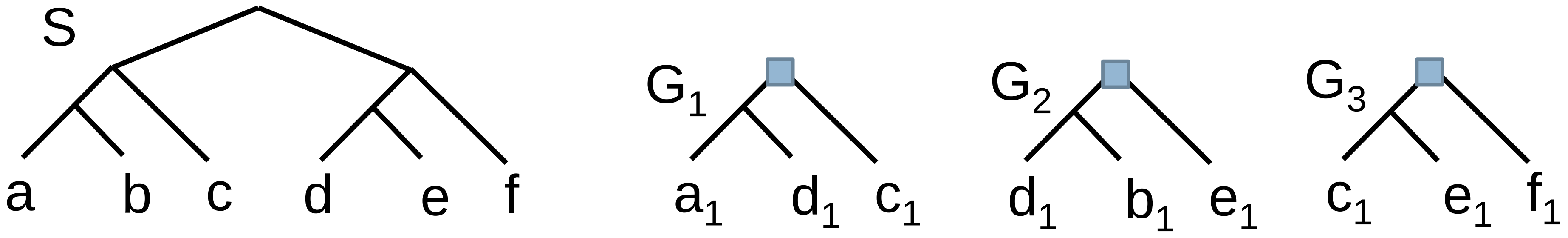
Extending the BUILD algorithm

- Trying every partition of the components can take some time.
- Instead, let’s find a way to choose a partition that "looks good".

A greedy approach

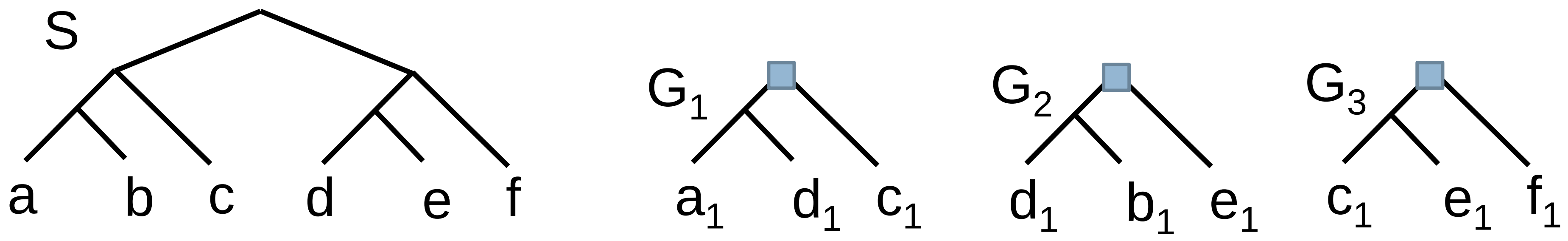


A greedy approach



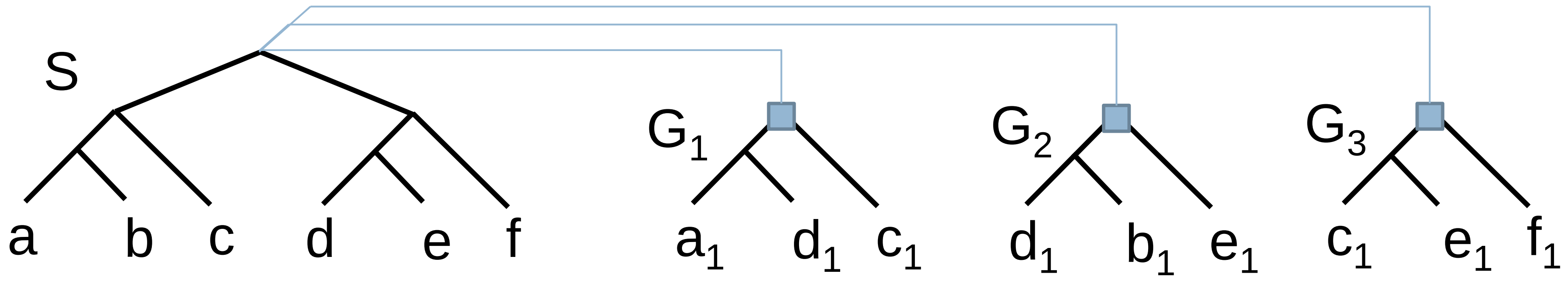
We already know that some duplications will be required.

A greedy approach



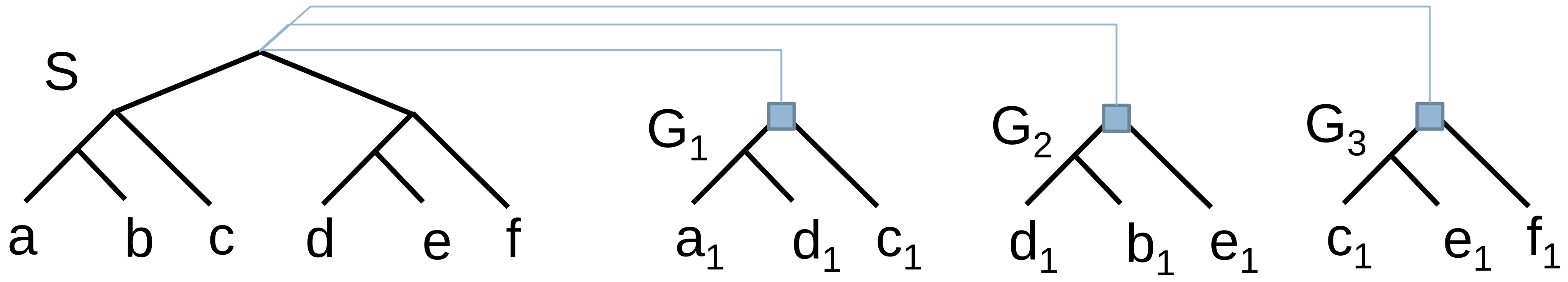
We already know that some duplications will be required.
Focus on the "highest" ones, i.e. those that occur before the first speciation in S.

A greedy approach



We already know that some duplications will be required.
Focus on the "highest" ones, i.e. those that occur before the first speciation in S.

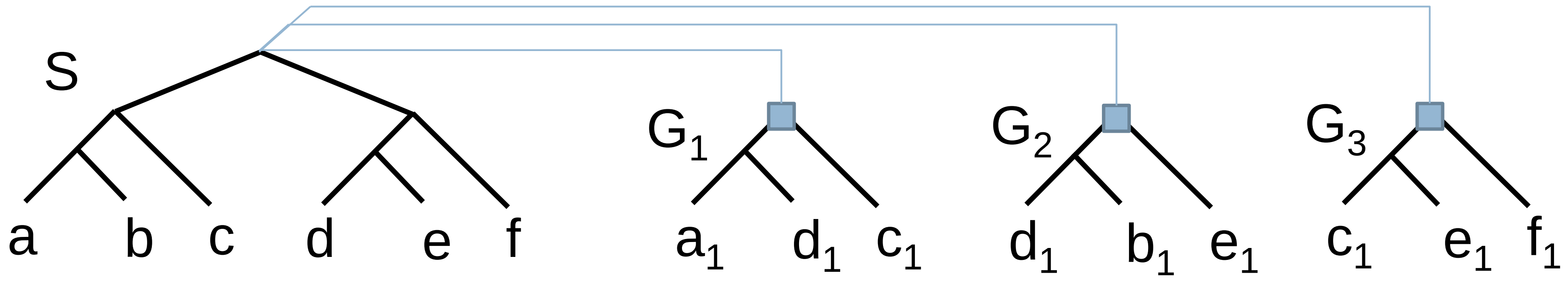
A greedy approach



We already know that some duplications will be required.
Focus on the "highest" ones, i.e. those that occur before the first speciation in S.

We call those duplication Pre Speciation Duplications (PreSpecDups).

A greedy approach

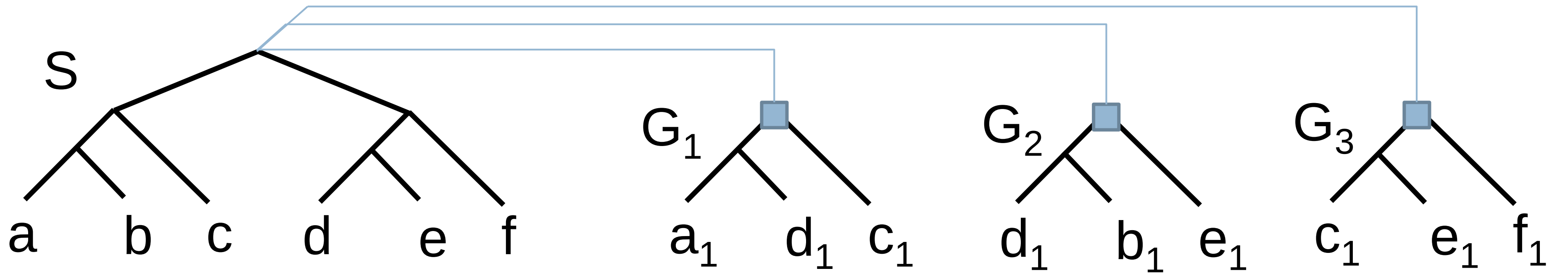


We already know that some duplications will be required.
Focus on the "highest" ones, i.e. those that occur before the first speciation in S.

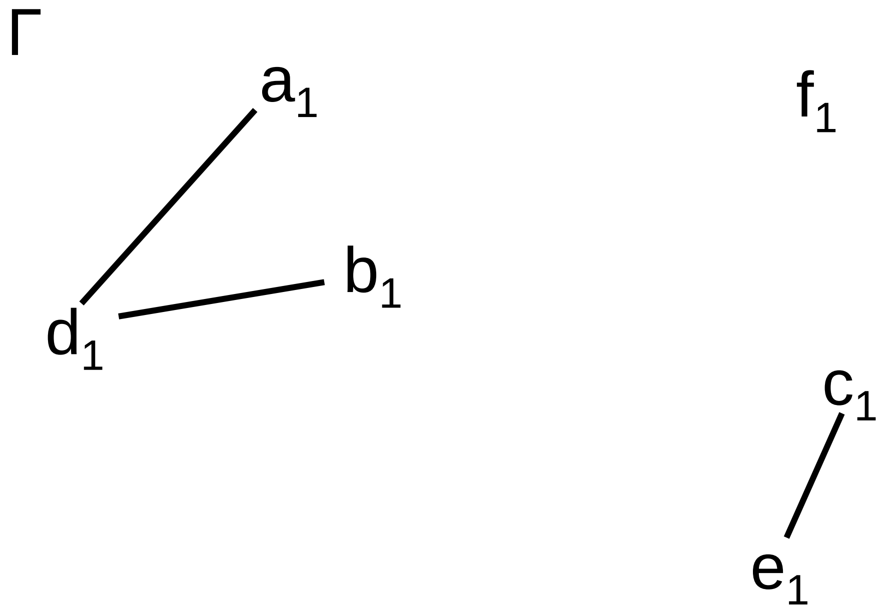
We call those duplication Pre Speciation Duplications (PreSpecDups).

New subproblem : minimize only these PreSpecDups

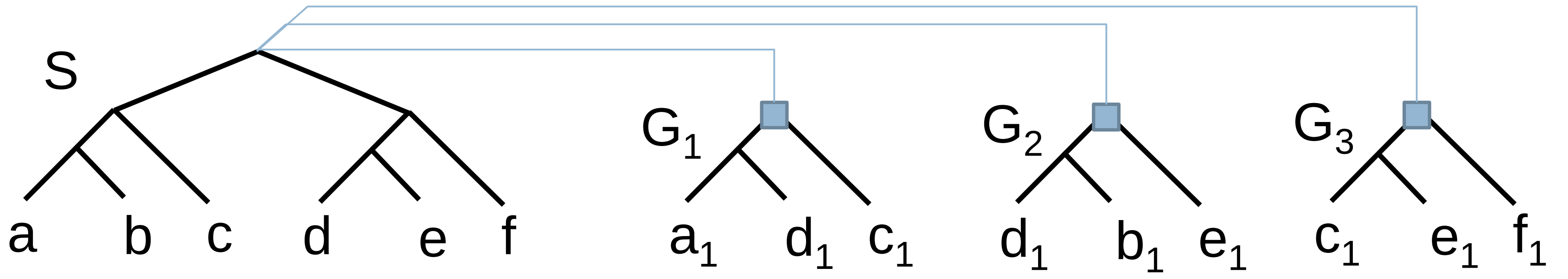
A greedy approach

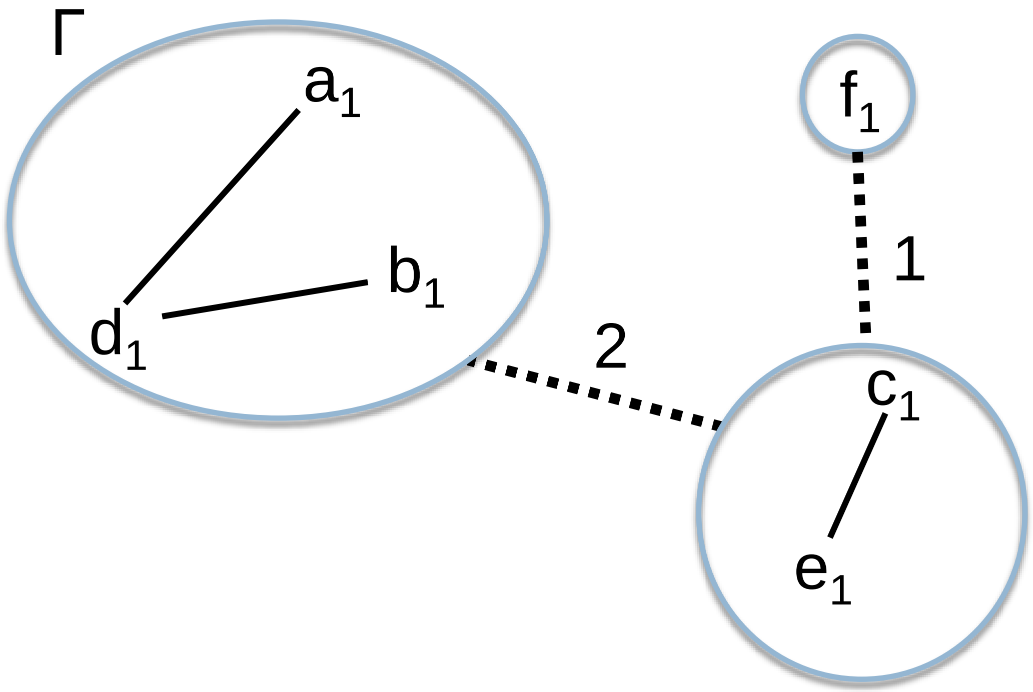


- Make the BUILD graph and identify the components.



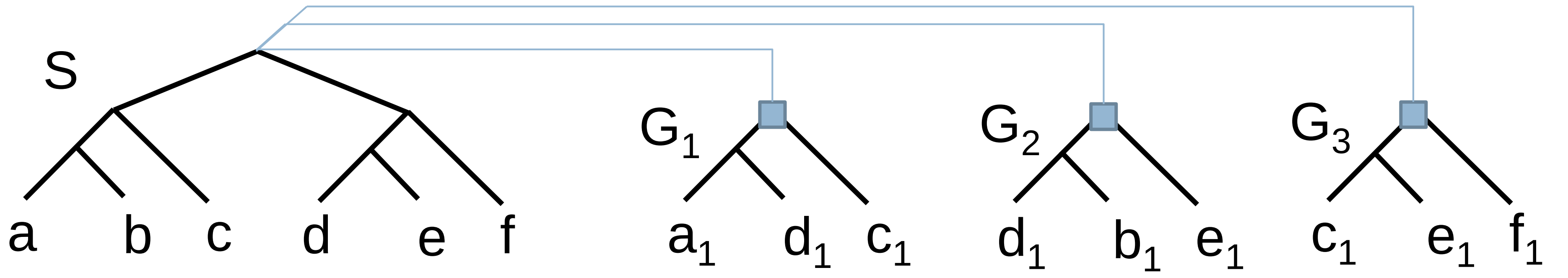
A greedy approach

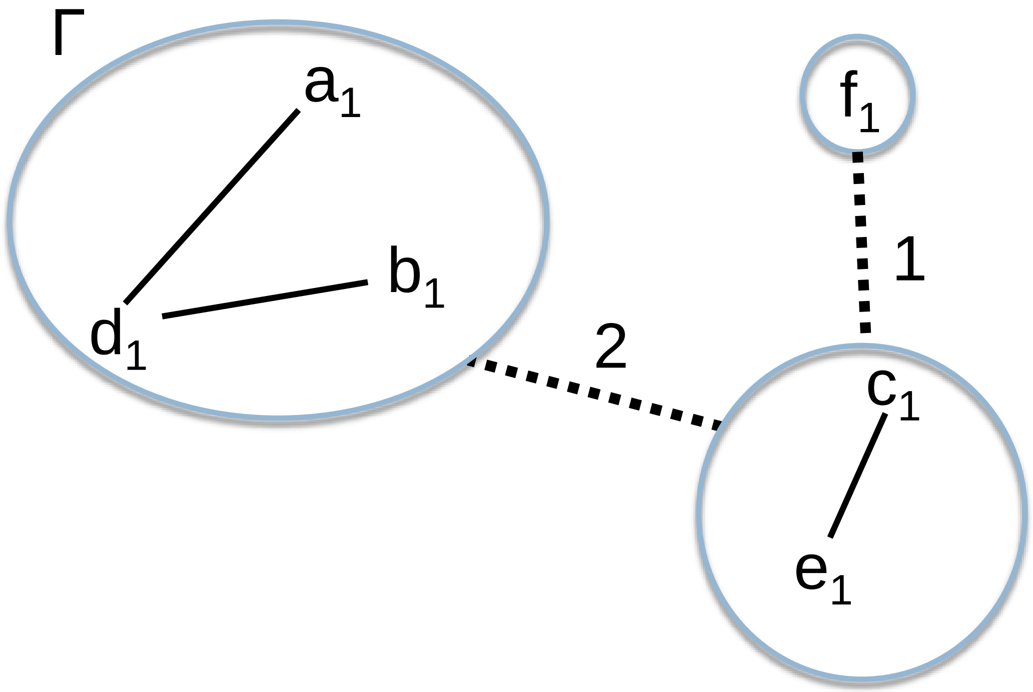




- Make the BUILD graph and identify the components.
- Add a special edge between components that requires a PreSpecDup when split.

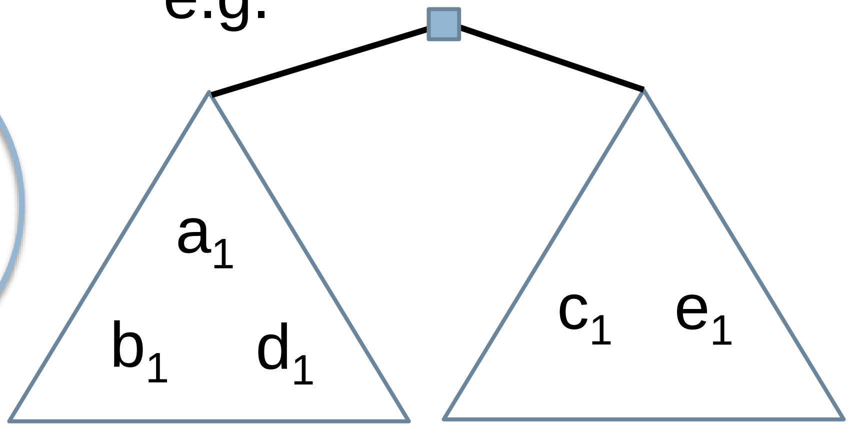
A greedy approach



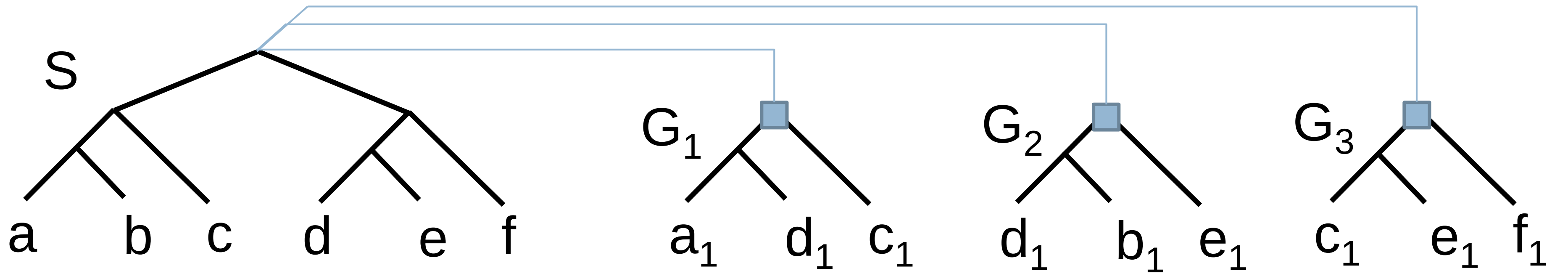


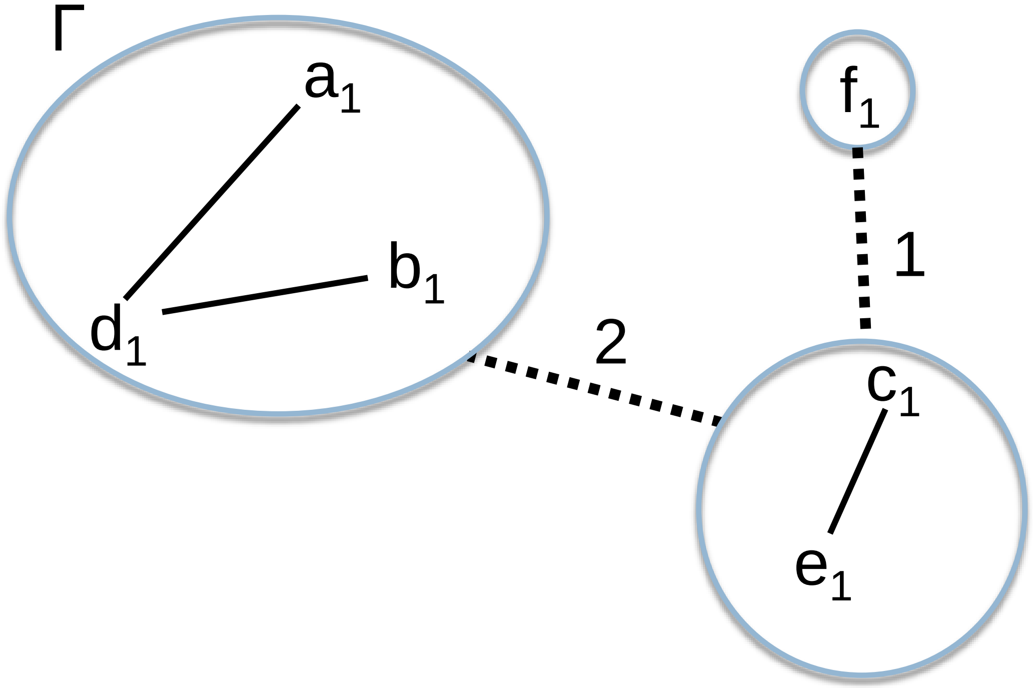
- Make the BUILD graph and identify the components.
- Add a special edge between components that requires a PreSpecDup when split.

e.g.



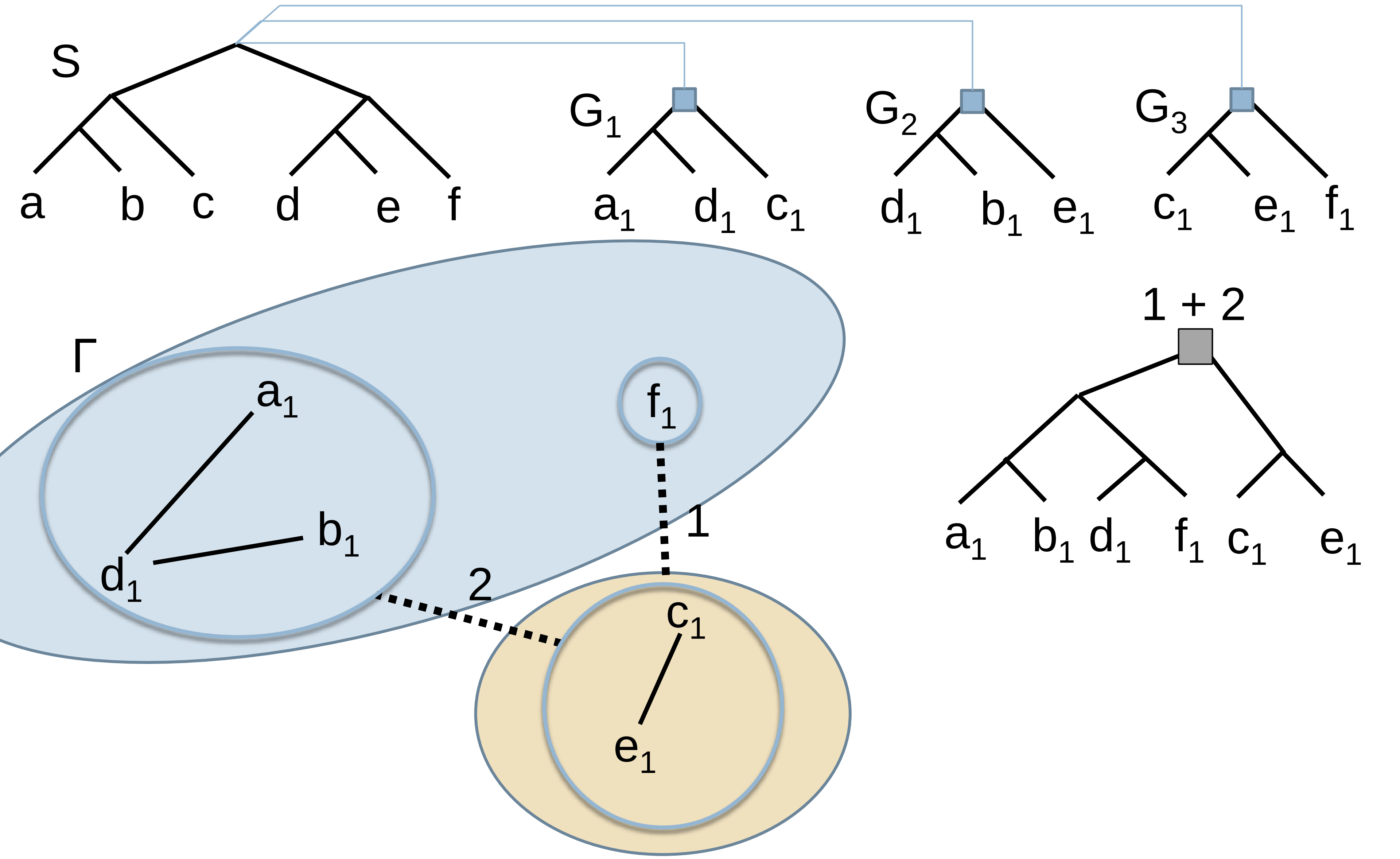
A greedy approach



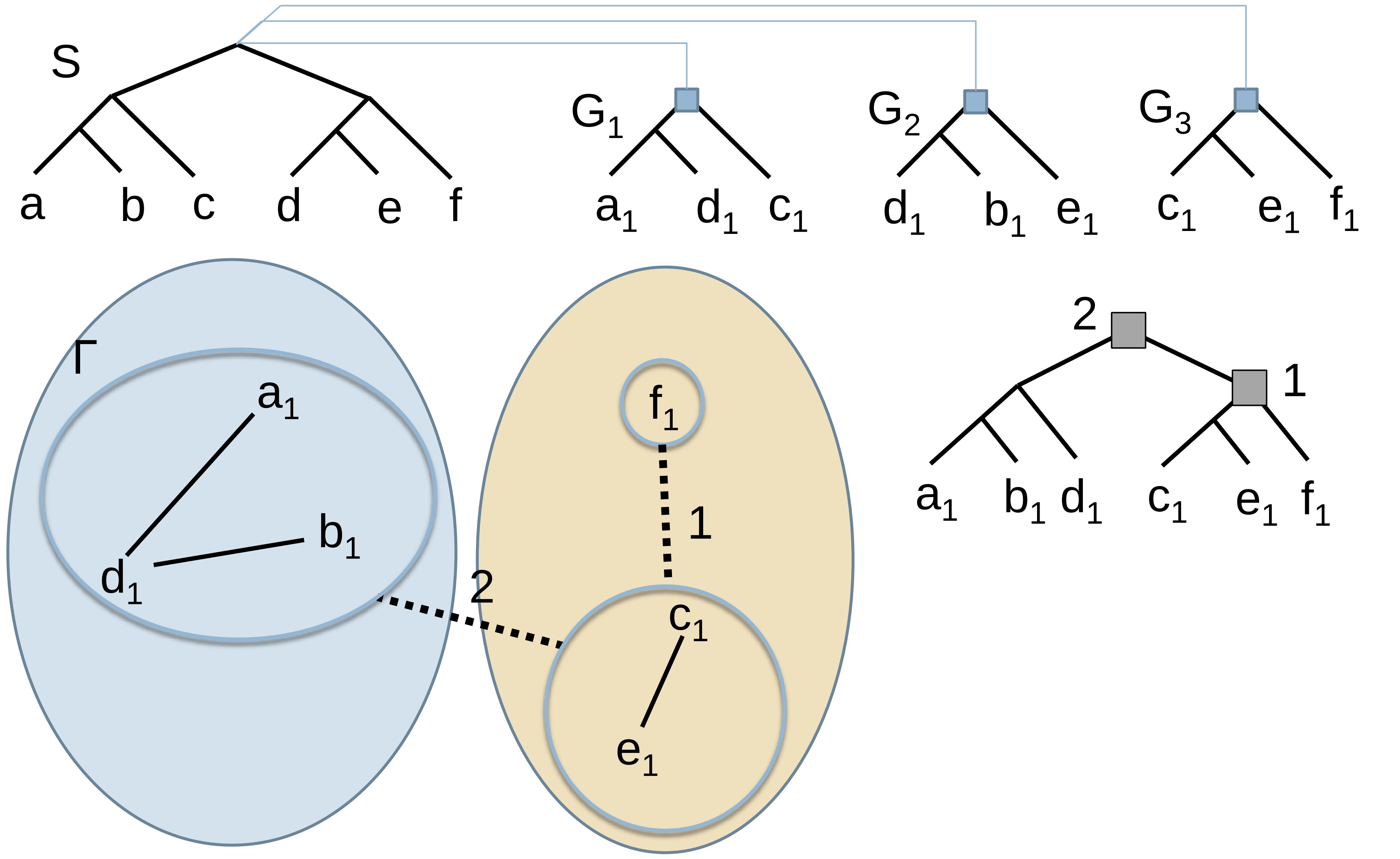


- Make the BUILD graph and identify the components.
- Add a special edge between components that requires a PreSpecDup when split.
- Find the partition that merges a maximum of duplications.

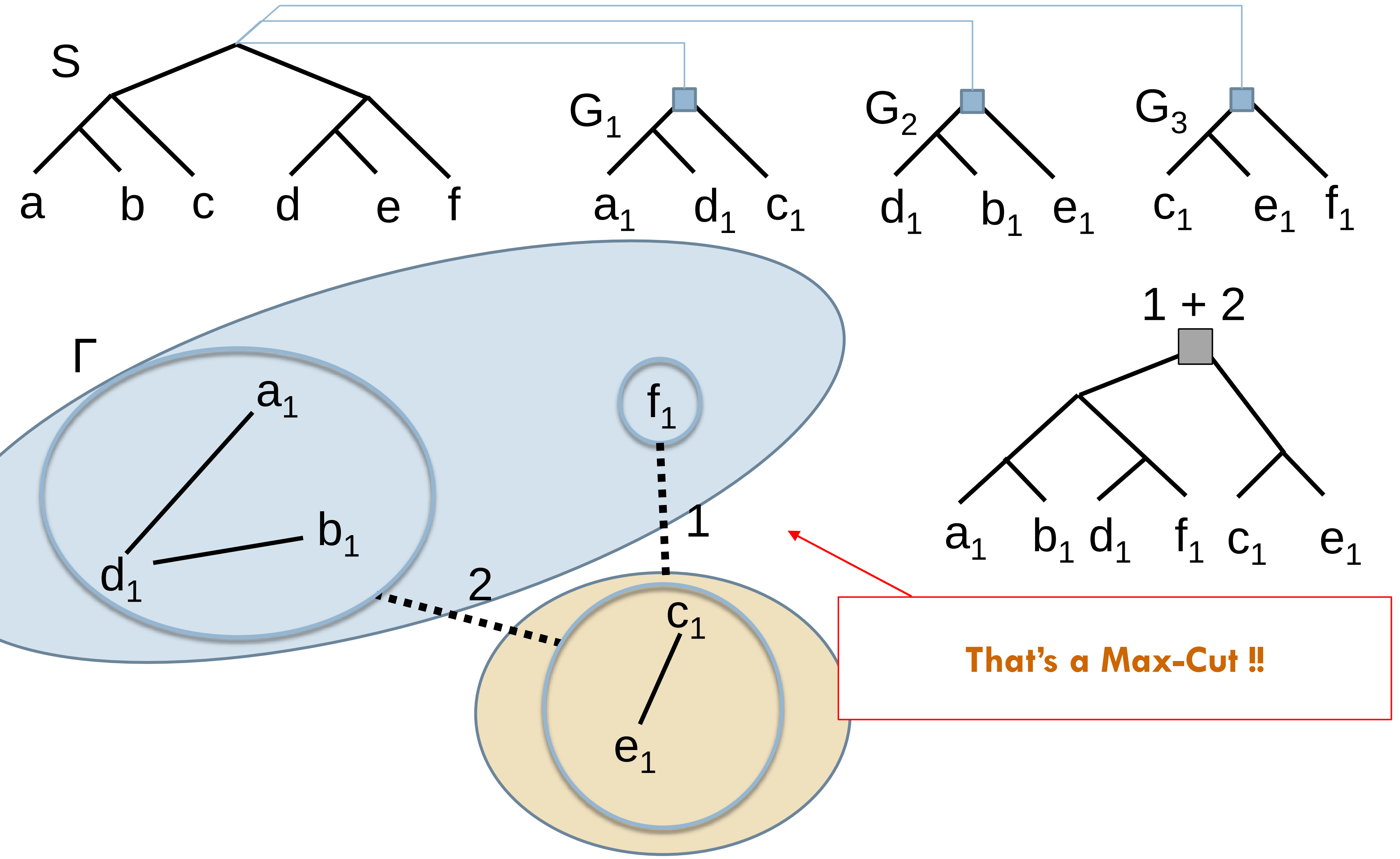
A greedy approach



A greedy approach



A greedy approach



That's a Max-Cut !!

Extending the BUILD algorithm

To minimize the number of PreSpecDups :

- Make the BUILD graph
- Add the PreSpecDup edges
- Find a Max-Cut partition of the components
- Repeat recursively on the parts

Extending the BUILD algorithm

To minimize the number of PreSpecDups :

- Make the BUILD graph
- Add the PreSpecDup edges
- Find a Max-Cut partition of the components
- Repeat recursively on the parts

That's NP-Hard ! And we have to repeat it recursively ‼

Extending the BUILD algorithm

To minimize the number of PreSpecDups :

- Make the BUILD graph
- Add the PreSpecDup edges
- Find a Max-Cut partition of the components
- Repeat recursively on the parts

That's NP-Hard ! And we have to repeat it recursively !!

The result : even this problem is hard to approximate !

Conclusion

- Fixed Parameter Tractability ?
- Criteria other than duplications ?
  - e.g. gene losses
- What to do if the input gene trees are incompatible ?

Acknowledgements

Aïda Ouangraoua

Nadia El-Mabrouk

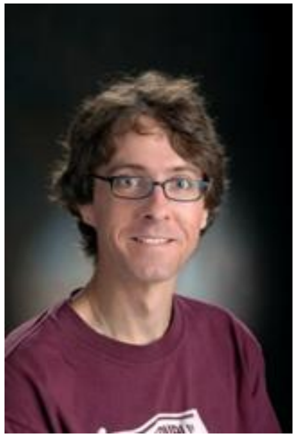
NSERC CRSNG

Fonds de recherche Nature et technologies Québec

Canada Research Chairs
www.chairs-chaires.gc.ca

DIRO

The 14th RECOMB-CG
October 2016 in MONTRÉAL ☺
Probably from Monday 10 to Wednesday 12



SFU
Simon Fraser University

DIRO

CENTRE DE RECHERCHES MATHÉMATIQUES

Université de Montréal