Parallel Inference of Phylogenetic Stands with Gentrius

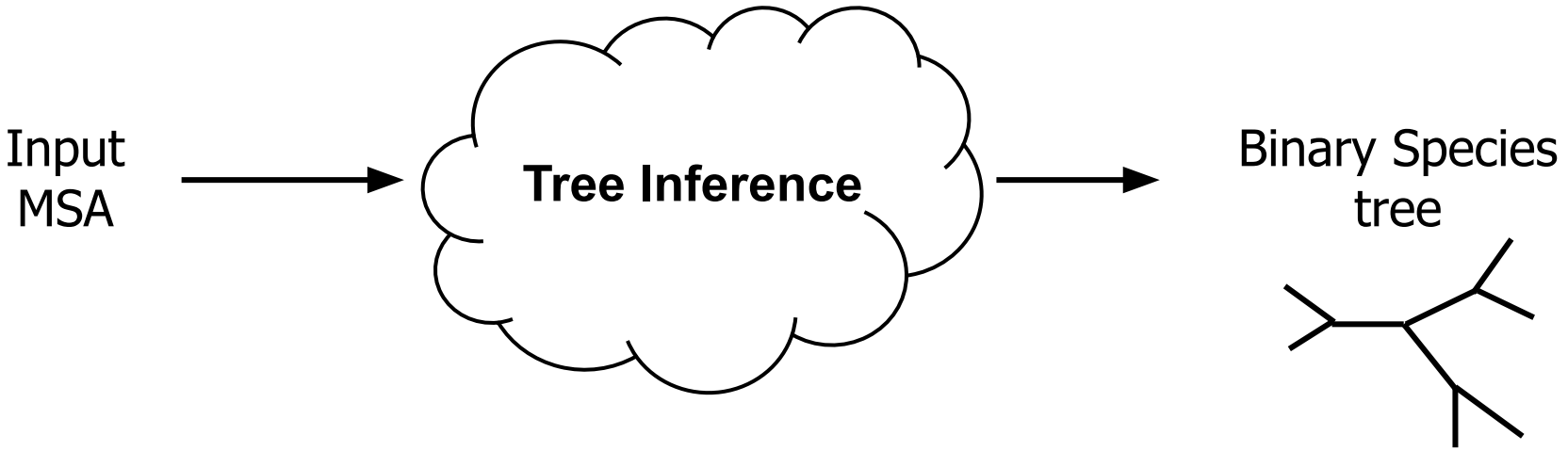
Anastasis Togkousidis, Olga Chernomor,
Alexandros Stamatakis

HiCOMB 2023 - 22nd IEEE International Workshop on
High Performance Computational Biology

15 May 2023

Introduction

General Pipeline



Standard Approach

- The **MSA** is a **single gene** or **locus** (genome region)

Taxon1: –AACGACGTAACTGGACC–CAAGA
Taxon2: TCACGACGTAAC–GGACC–CAAGA
Taxon3: TAAC–ACGTAACTCGACCGCAAGA
Taxon4: TAAC–ACGTTACTCGACCGCAAGA

- Tree Inference: Maximum Likelihood (ML) or Bayesian Inference
- **Assumption**: This gene/locus constitutes a valid proxy for the entire evolutionary history of the species.

Problems of Standard Approach

- **Different genes** might have evolved under **different models** and parameters
- Tree inferences conducted on distinct genes often yield **incongruent tree topologies**

Alternative Approach

- The **MSA** comprises multiple genes/loci (**multi-partitioned**)

	Gene 1	Gene 2	Gene 3
Taxon1:	–AACGACGTCTGGACC–CA	GC–ACGAACTGGACCTCAAGA	AATTCGAACTGGA–
Taxon2:	TCACGACGTC–GGACC–CA	GCCTCGAAC–GGACC–CAAGT	AATTCGA–TTGGA–
Taxon3:	TAAC–ACGTCTCGACCGCA	GC–ACGA–CTCGACTGCACGA	AATCCGA–CTGGAT
Taxon4:	TAAC–ACGACTCGACCGCA	GC–ACGT–CTCGACT–CAAGT	A–TCCAA–TTGGCT

- The **information** derived from multiple genes is **summarized** into a **single species tree**
- Multiple species tree inference approaches (Supermatrix, Supertree etc).

Missing data in multi-partitioned MSAs

- Multi-partitioned MSAs often exhibit **missing data**
- A **species** may have **no data present** in a **specific locus** (sampling issues or absence of target gene)

	Gene 1	**Gene 2**	**Gene 3**
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT

Missing data in multi-partitioned MSAs

- Multi-partitioned MSAs often exhibit **missing data**
- A **species** may have **no data present** in a **specific locus** (sampling issues or absence of target gene)

	Gene 1	Gene 2	Gene 3
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT

	Gene 1	Gene 2	Gene 3
Taxon 1	1	1	0
Taxon 2	0	1	1
Taxon 3	1	0	1
Taxon 4	1	0	1
Taxon 5	1	1	0
Taxon 6	1	1	1

Presence - Absence Matrix (PAM)

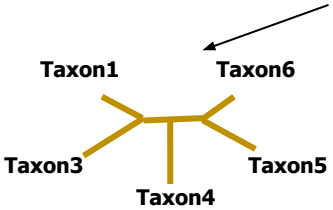
Our problem

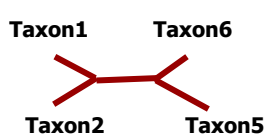
- Given a set of **incomplete** unrooted **gene trees**

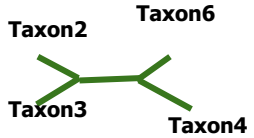
Our problem

- Given a set of **incomplete** unrooted **gene trees**

	Gene 1	Gene 2	Gene 3
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT





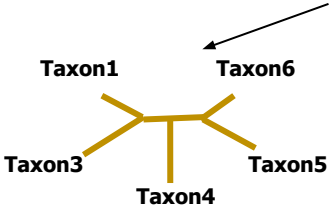


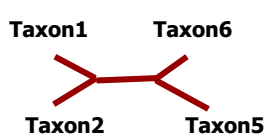
Our problem

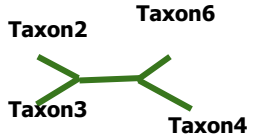
- Given a set of **incomplete** unrooted **gene trees**

- We want to **enumerate** all **complete species trees** which are **compatible with** the incomplete **gene trees**

	Gene 1	Gene 2	Gene 3
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT





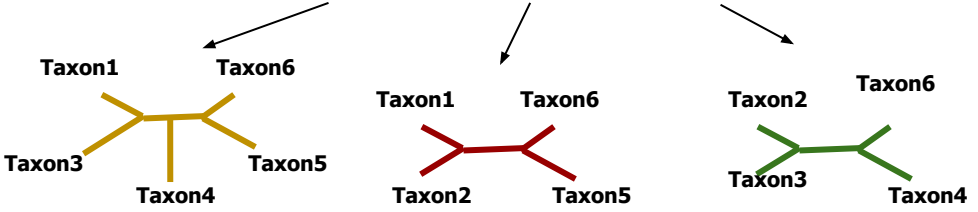


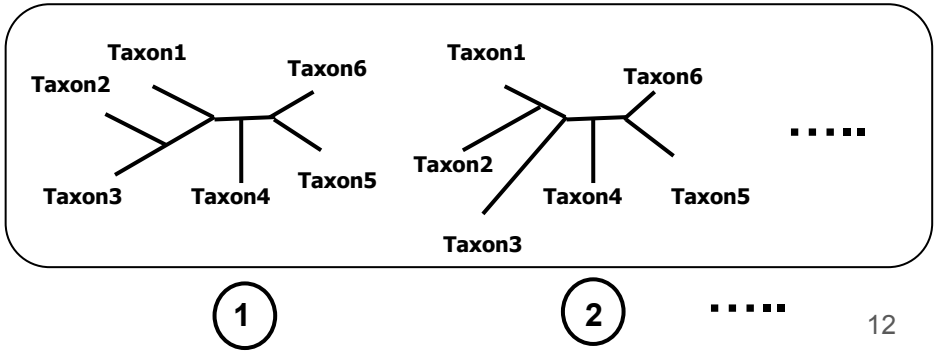
Our problem

- Given a set of **incomplete** unrooted **gene trees**

- We want to **enumerate** all **complete species trees** which are **compatible with** the incomplete **gene trees**

	Gene 1	Gene 2	Gene 3
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT





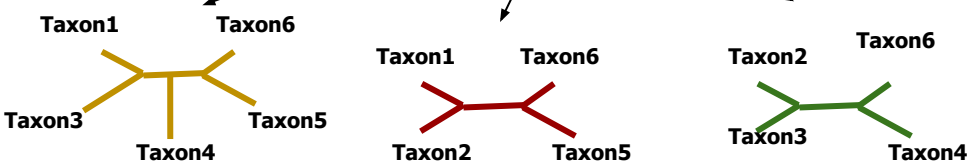
Our problem

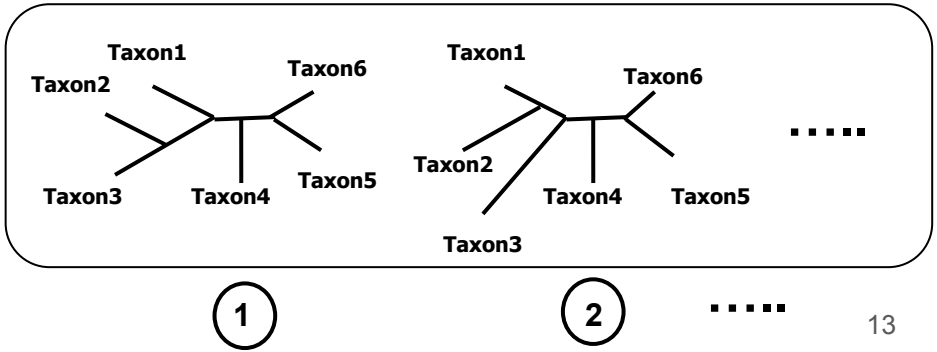
- Given a set of **incomplete** unrooted **gene trees**

- We want to **enumerate** all **complete species trees** which are **compatible with** the incomplete **gene trees**

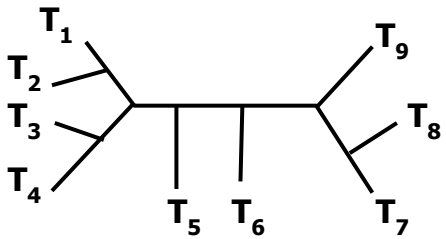
  - ***Stand***
  - To be defined …

	Gene 1	Gene 2	Gene 3
Taxon1:	AACGT-	GC–ACG	---------
Taxon2:	---------	GCCTCG	TTGGA–
Taxon3:	AACGTA	---------	CTGGAT
Taxon4:	AACGGA	---------	TTGGCT
Taxon5:	ACCG -A	- CCTCG	---------
Taxon6:	AACGGA	GCCT -A	CTGGCT





Equivalent problem



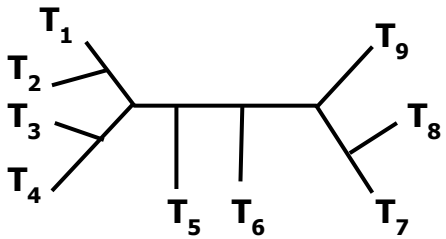
Complete Species Tree

✚

	G_1	G_2	G_3	G_4
T_1	1	0	1	1
T_2	0	0	1	0
T_3	1	0	1	1
T_4	1	0	1	0
T_5	0	1	0	0
T_6	0	1	1	0
T_7	1	1	1	1
T_8	0	1	1	0
T_9	1	0	0	1

PAM

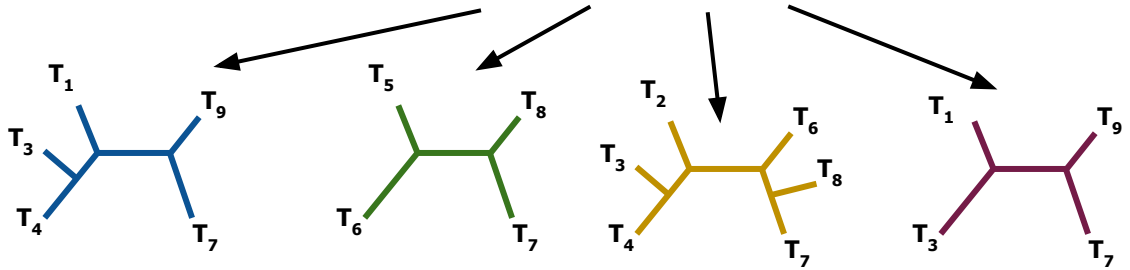
Equivalent problem



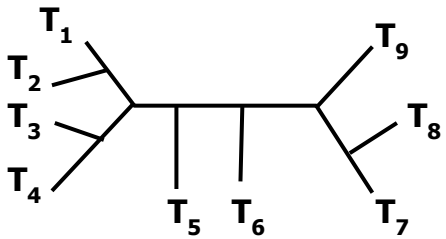
Complete Species Tree

	G_1	G_2	G_3	G_4
T_1	1	0	1	1
T_2	0	0	1	0
T_3	1	0	1	1
T_4	1	0	1	0
T_5	0	1	0	0
T_6	0	1	1	0
T_7	1	1	1	1
T_8	0	1	1	0
T_9	1	0	0	1

PAM



Equivalent problem



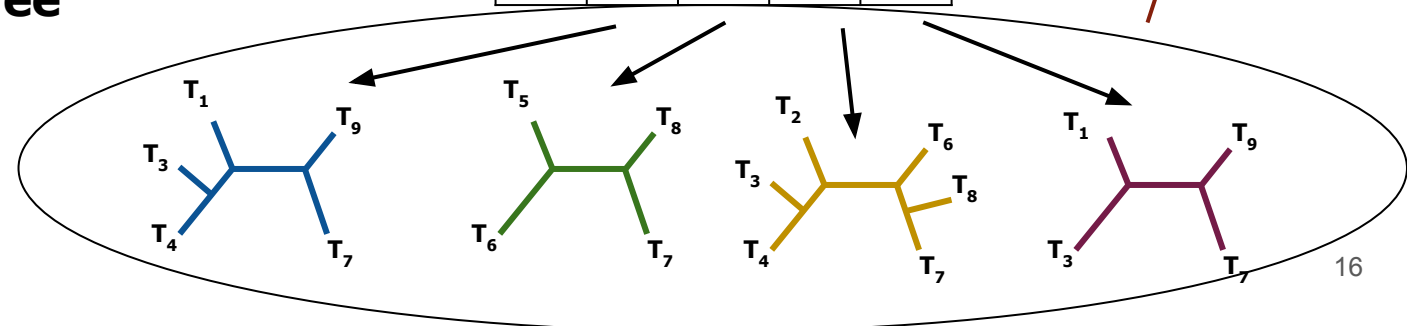
Complete Species Tree

At least one tree on stand

	G_1	G_2	G_3	G_4
T_1	1	0	1	1
T_2	0	0	1	0
T_3	1	0	1	1
T_4	1	0	1	0
T_5	0	1	0	0
T_6	0	1	1	0
T_7	1	1	1	1
T_8	0	1	1	0
T_9	1	0	0	1

PAM

Incomplete induced / displayed subtrees



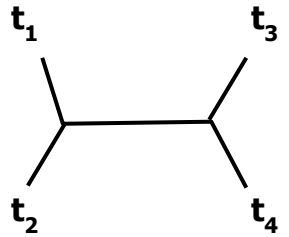
Background

Contents

- **Compatibility**
- Gentrius Algorithm
- Why care about stands?

Compatibility

We say that an **incomplete tree T_i with taxa Y_i** (e.g. $Y_i = \{t_1,t_2,t_3,t_4\}$)

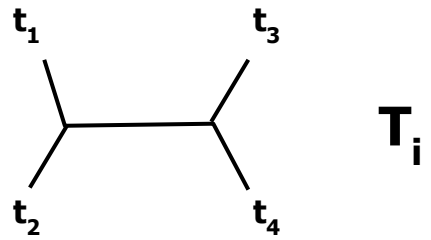


T_i

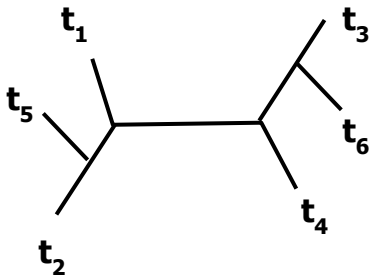
Compatibility

We say that an **incomplete tree T_i with taxa Y_i** (e.g. $Y_i = \{t_1,t_2,t_3,t_4\}$)

is **compatible** with a **complete species tree T** with **taxa $Y \supseteq Y_i$** (e.g. $Y = \{t_1,t_2,t_3,t_4,t_5,t_6\}$)



T_i



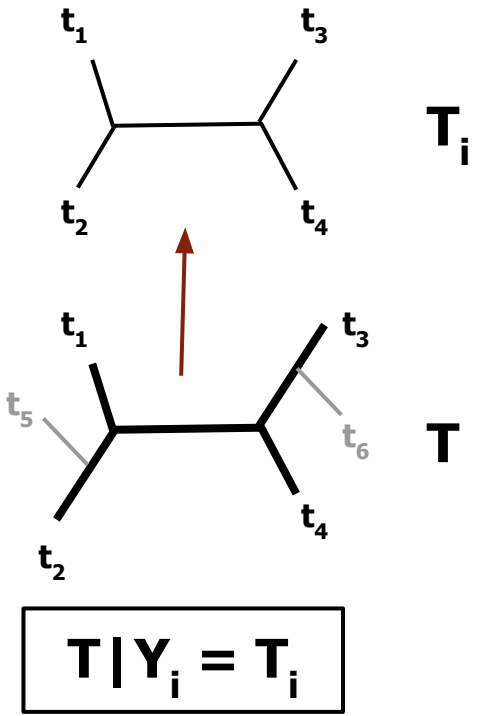
T

Compatibility

We say that an **incomplete tree T_i with taxa Y_i** (e.g. $Y_i = \{t_1,t_2,t_3,t_4\}$)

is **compatible** with a **complete species tree T** with **taxa $Y \supseteq Y_i$** (e.g. $Y = \{t_1,t_2,t_3,t_4,t_5,t_6\}$)

if **T can be reduced to T_i** by **collapsing** some of its **edges** (simply put, remove the extra taxa)



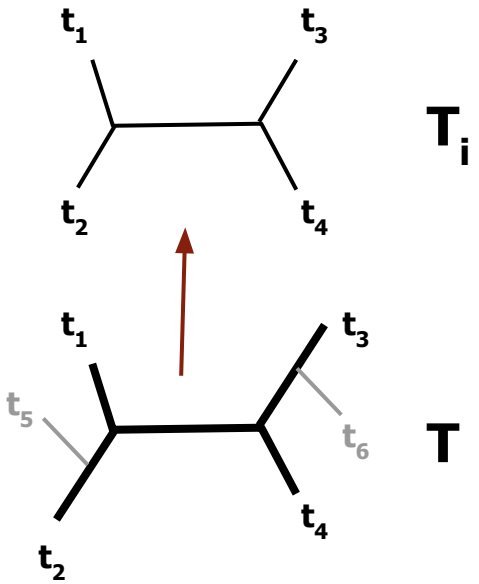
Compatibility

We say that an **incomplete tree T_i with taxa Y_i** (e.g. $Y_i = \{t_1,t_2,t_3,t_4\}$)

is **compatible** with a **complete species tree T** with **taxa $Y \supseteq Y_i$** (e.g. $Y = \{t_1,t_2,t_3,t_4,t_5,t_6\}$)

if **T can be reduced to T_i** by **collapsing** some of its **edges** (simply put, remove the extra taxa)

<u>Equivalent Phrases:</u> T displays T_i, T is reduced to T_i, T_i is induced by T (or is an induced subtree)

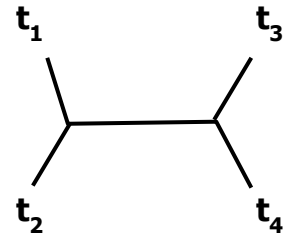


$T|Y_i = T_i$

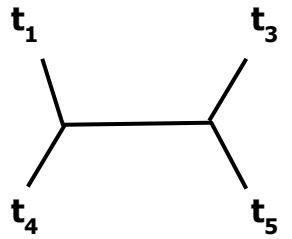
Compatibility (Generalized)

We say that **two** (incomplete) **trees**

- **T_1 with taxa Y_1** (e.g. $Y_1 = \{t_1,t_2,t_3,t_4\}$)
- **T_2 with taxa Y_2** (e.g. $Y_2 = \{t_1,t_3,t_4,t_5\}$)



T_1



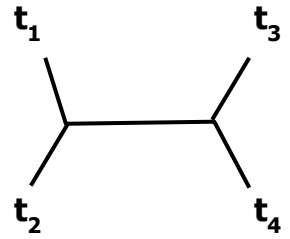
T_2

Compatibility (Generalized)

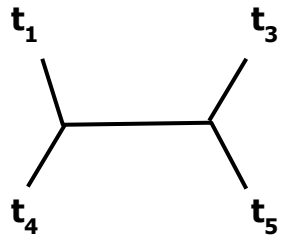
We say that **two** (incomplete) **trees**

- **T_1 with taxa Y_1** (e.g. $Y_1 = \{t_1,t_2,t_3,t_4\}$)
- **T_2 with taxa Y_2** (e.g. $Y_2 = \{t_1,t_3,t_4,t_5\}$)

Are **compatible** if **there is a tree T that displays both** (with **taxa $Y = Y_1 \cup Y_2$**)

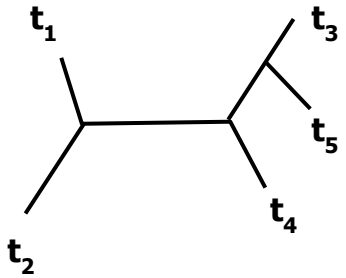


T_1



T_2

T

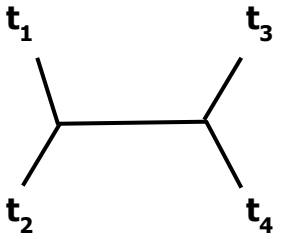


Compatibility (Generalized)

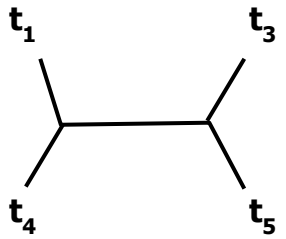
We say that **two** (incomplete) **trees**

- **T_1 with taxa Y_1** (e.g. $Y_1 = \{t_1,t_2,t_3,t_4\}$)
- **T_2 with taxa Y_2** (e.g. $Y_2 = \{t_1,t_3,t_4,t_5\}$)

Are **compatible** if **there is a tree T that displays both** (with **taxa $Y = Y_1 \cup Y_2$**)

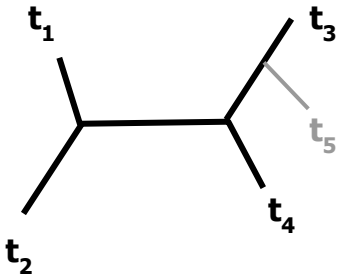


T_1



T_2

T

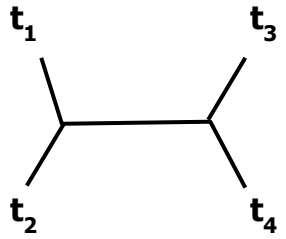


Compatibility (Generalized)

We say that **two** (incomplete) **trees**

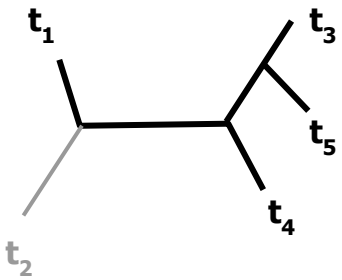
- **T_1 with taxa Y_1** (e.g. $Y_1 = \{t_1,t_2,t_3,t_4\}$)
- **T_2 with taxa Y_2** (e.g. $Y_2 = \{t_3,t_4,t_5,t_5\}$)

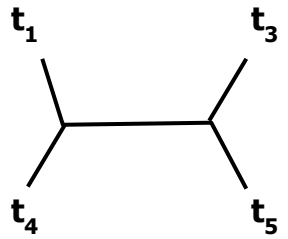
Are **compatible** if **there is a tree T that displays both** (with **taxa $Y = Y_1 \cup Y_2$**)



T_1

T





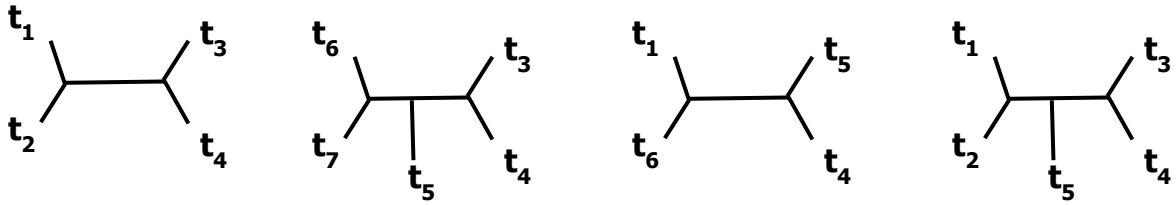
T_2

Contents

- **Compatibility**
- **Gentrius Algorithm**
- Why care about stands?

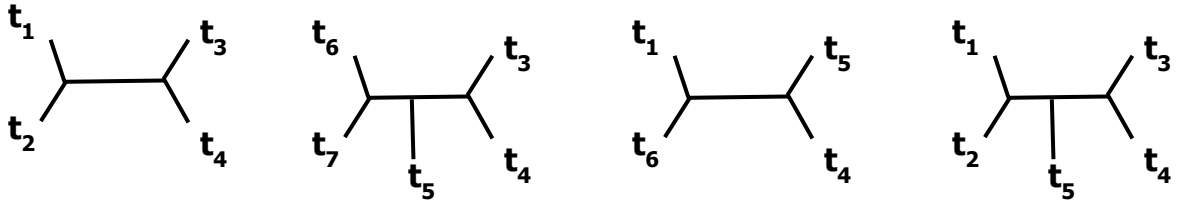
Gentrius Algorithm (Chernomor *et al.*)

- Given a set of incomplete unrooted trees

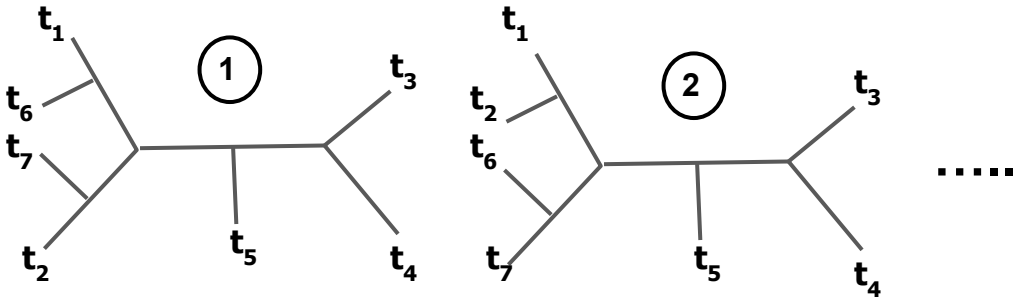


Gentrius Algorithm (Chernomor *et al.*)

- Given a set of incomplete unrooted trees

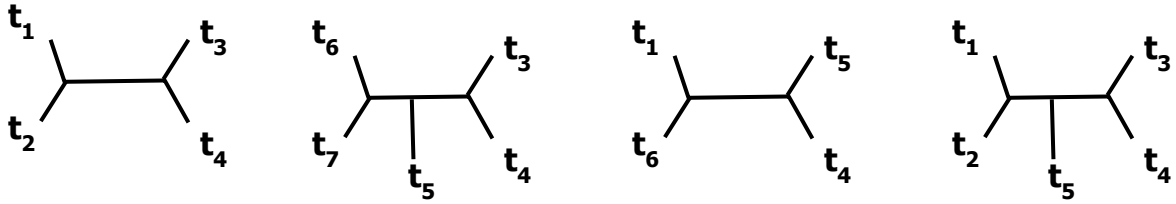


- **Gentrius generates all trees on stand** (trees that are compatible with all the unrooted subtrees)

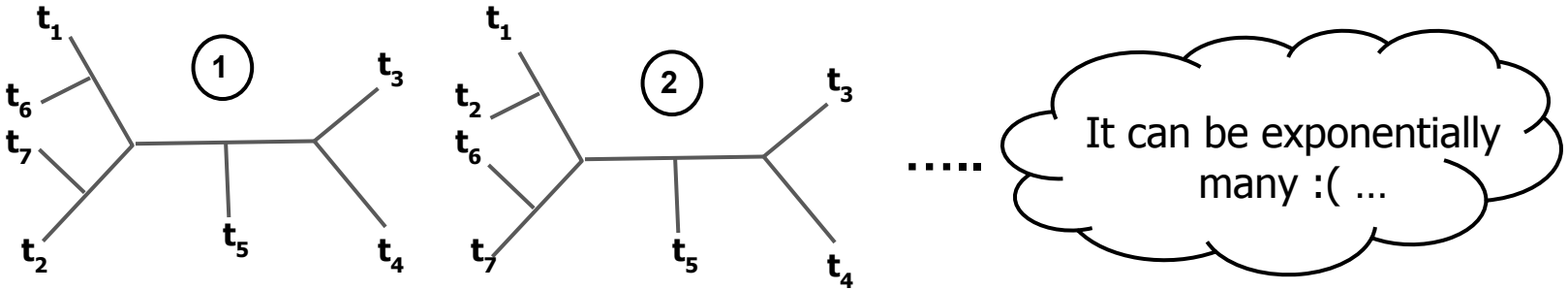


Gentrius Algorithm (Chernomor *et al.*)

- Given a set of incomplete unrooted trees



- **Gentrius generates all trees on stand** (trees that are compatible with all the unrooted subtrees)

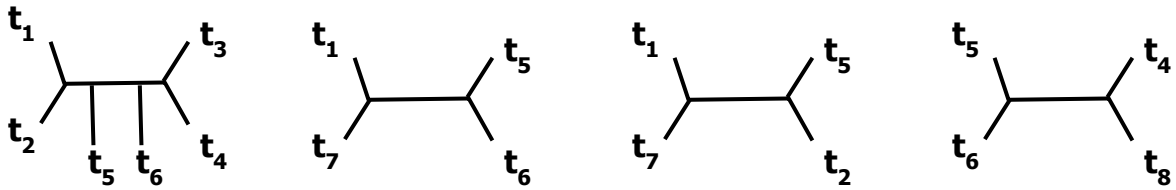


How Gentrius Works

- **Deterministic**, Branch and bound algorithm
- **One tree** serves as the initial tree (**agile tree**).
- The whole stand is generated by **stepwise taxon insertion**. Missing taxa are sequentially inserted into the agile tree
- Missing taxa are inserted into **admissible branches** on the agile tree
- Admissible branches are determined mathematically from the **double-edged mappings** (see Chernomor *et al.*)

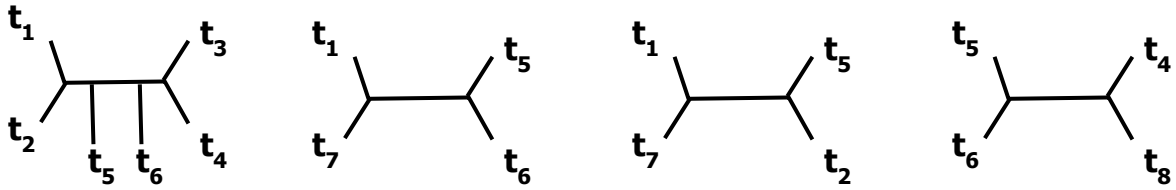
Simple Example

- Initial set of incomplete unrooted trees



Simple Example

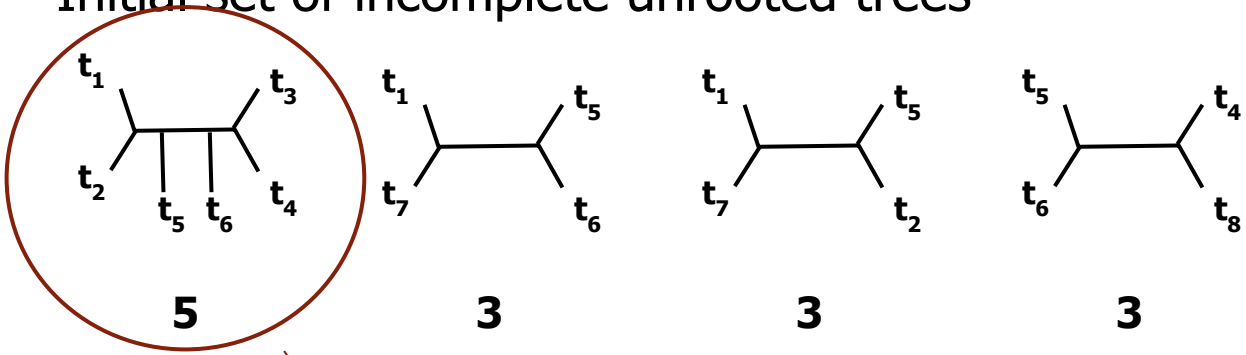
- Initial set of incomplete unrooted trees



- Gentrius **selects** the tree with the **largest number of shared taxa** as the initial **agile tree**

Simple Example

- Initial set of incomplete unrooted trees

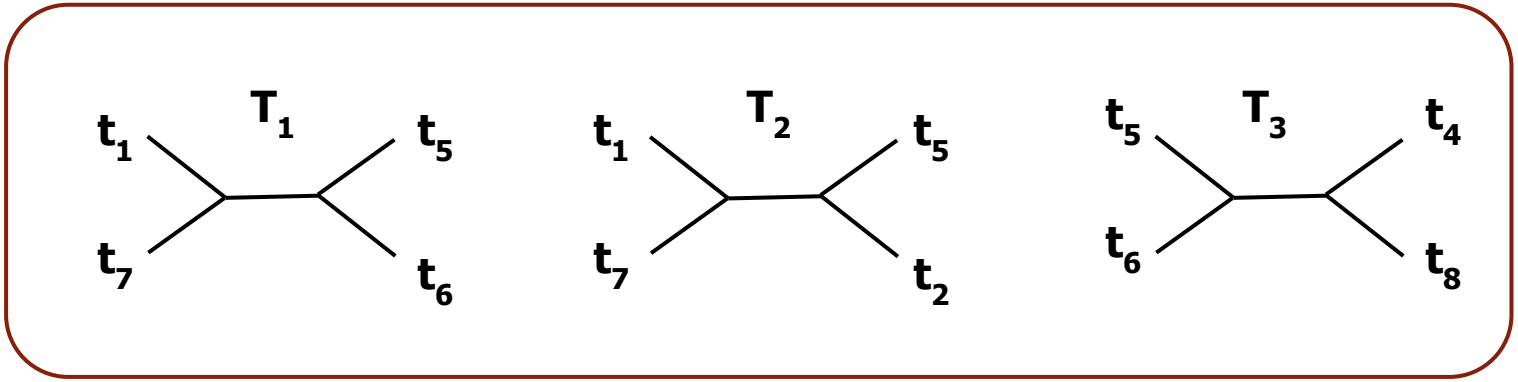


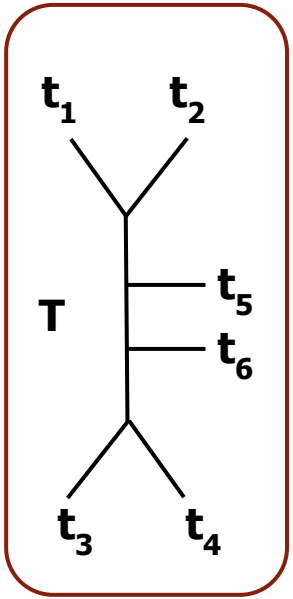
- Gentrius **selects** the tree with the **largest number of shared taxa** as the initial **agile tree**

Agile tree

Simple Example

Missing Taxa: t_7, t_8



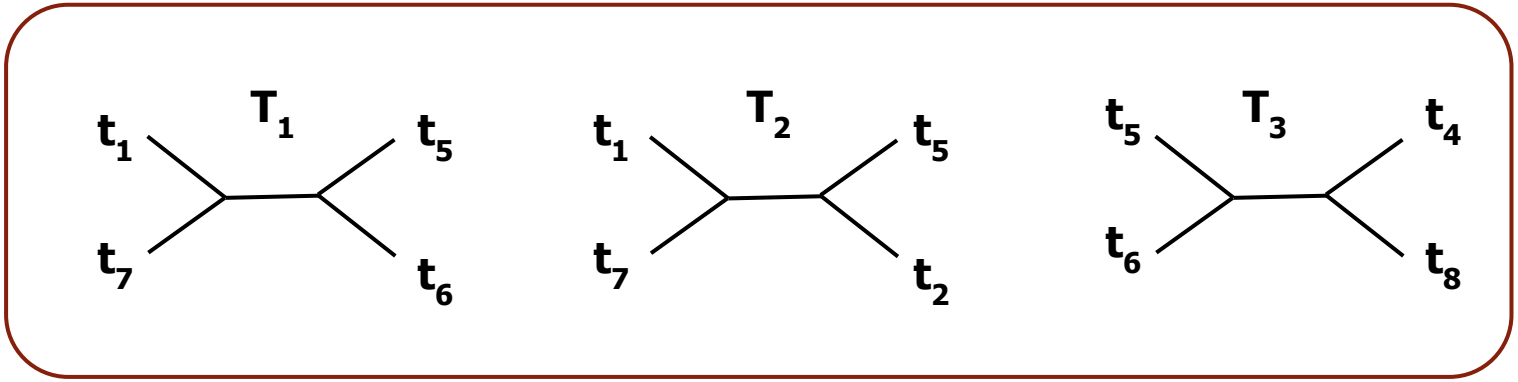


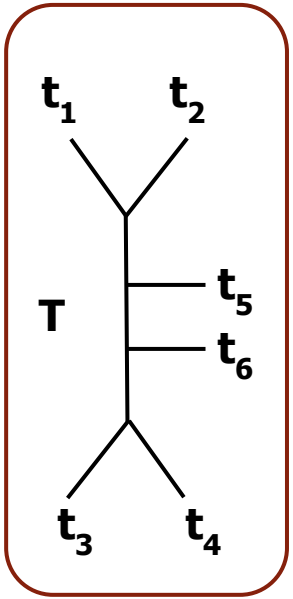
→ **Agile tree**

Constraint Trees

Simple Example

Missing Taxa: t_7, t_8



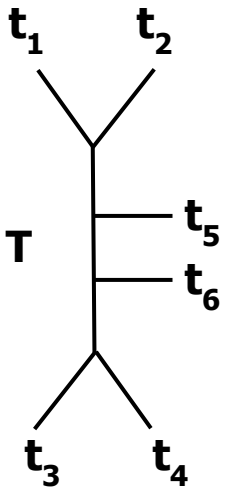


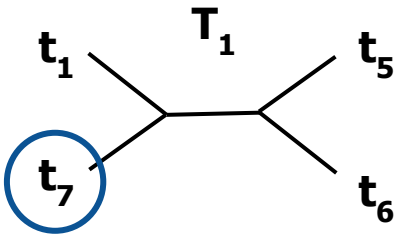
→ **Agile tree** + **Constraint Trees** △ **State** of the algorithm

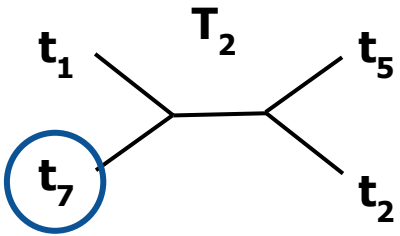
Simple Example

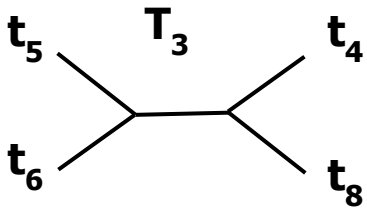
Missing Taxa: t_7, t_8

- t_7 is present on constraint trees T_1 and T_2





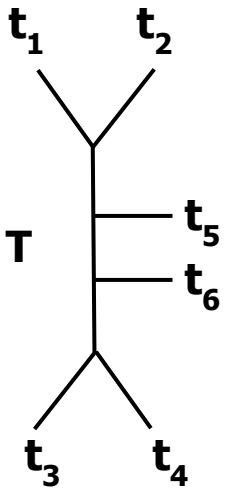


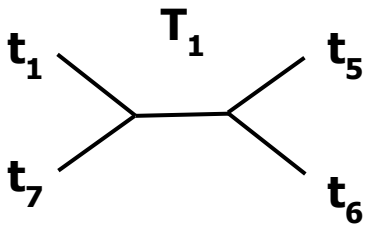


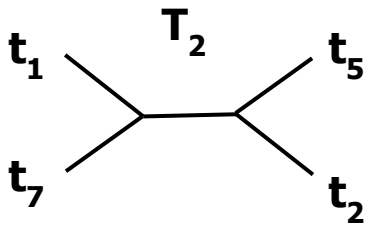
Simple Example

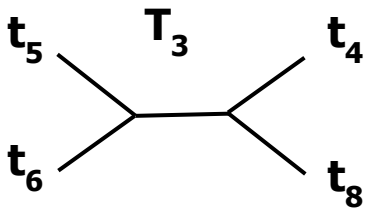
Missing Taxa: t_7, t_8

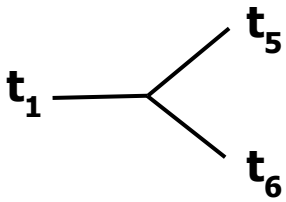
- Build **common subtree** between **T** and T_1







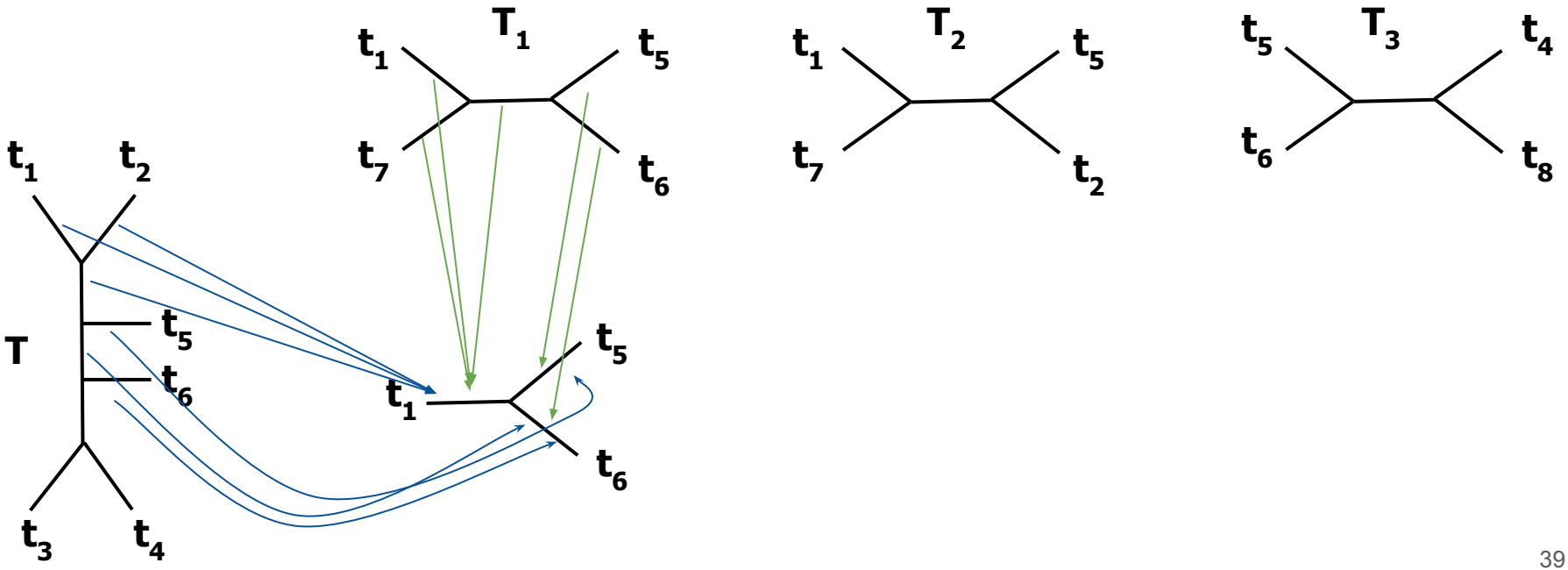




Simple Example

Missing Taxa: t_7, t_8

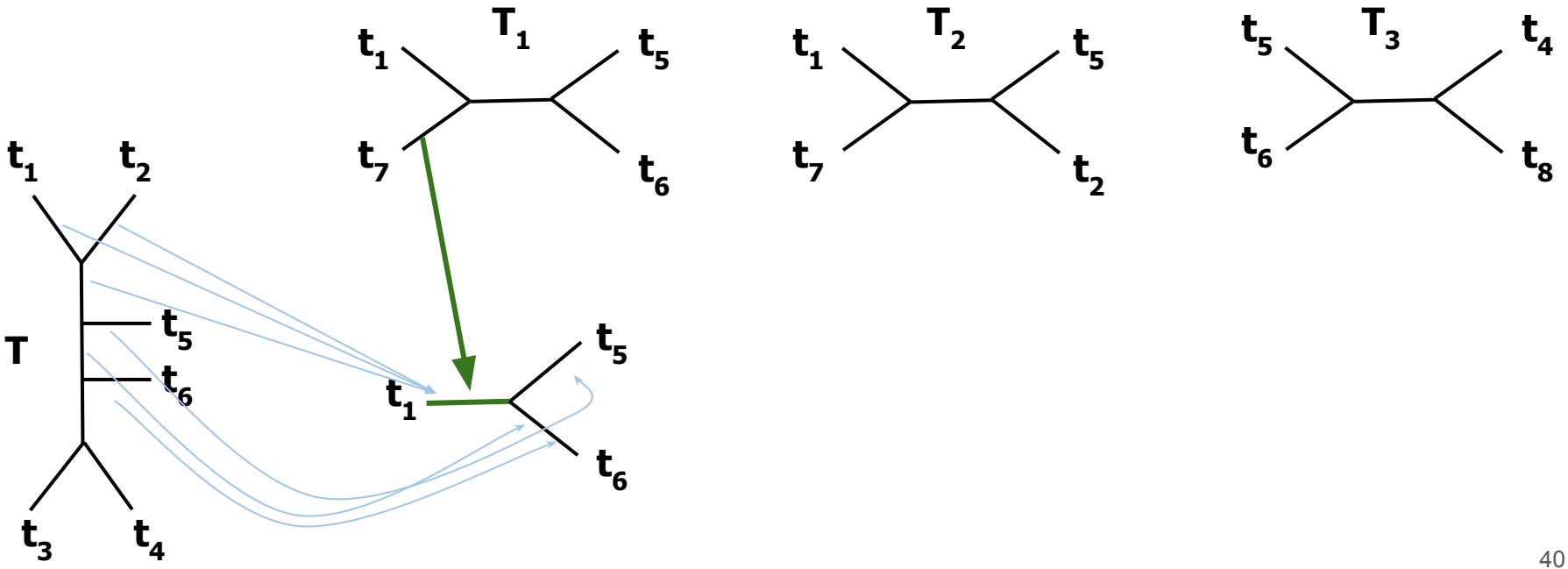
- **Map edges** (I'm omitting some mappings for simplicity)



Simple Example

Missing Taxa: t_7, t_8

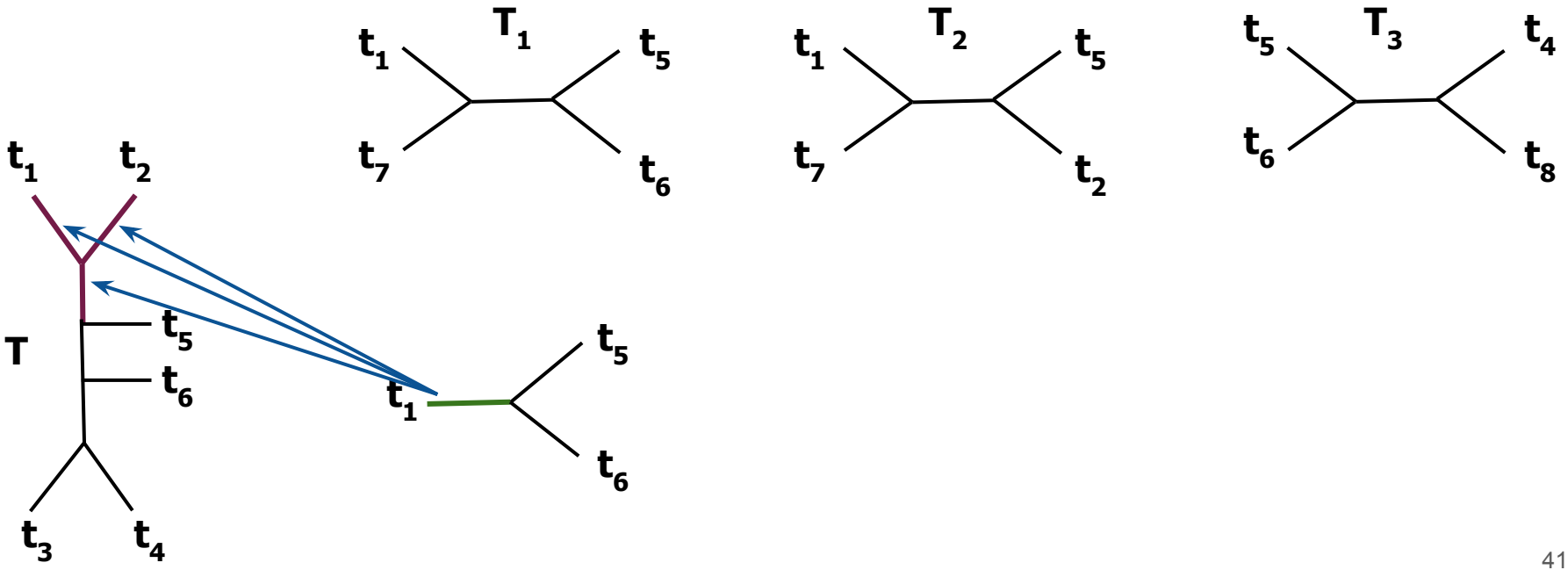
- Find the edge in which t_7 is mapped (it is only one edge)



Simple Example

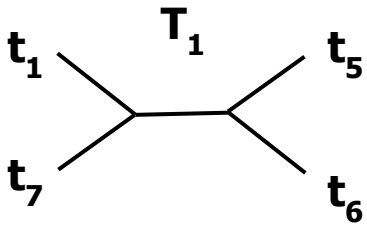
Missing Taxa: t_7, t_8

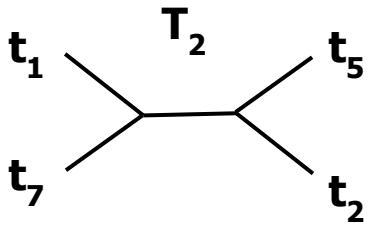
- Map backwards

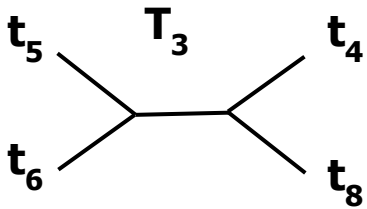


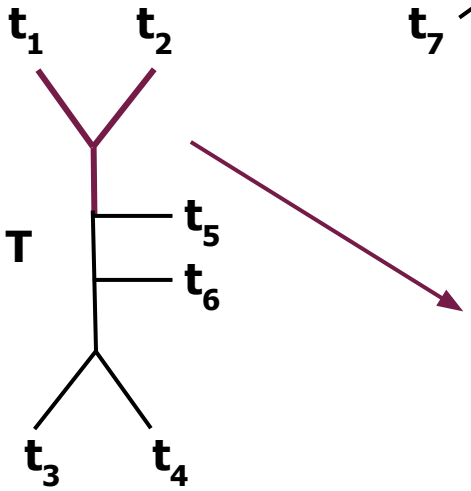
Simple Example

Missing Taxa: t_7, t_8









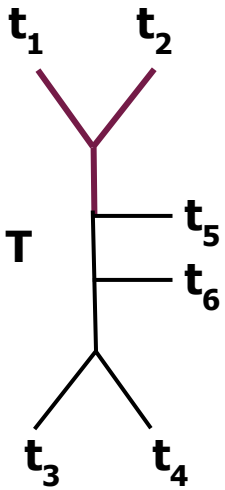
Admissible branches

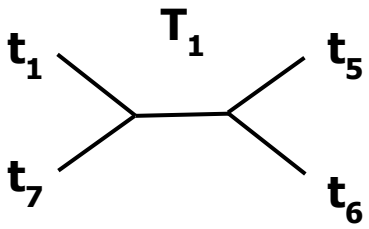
(We're not finished yet …)

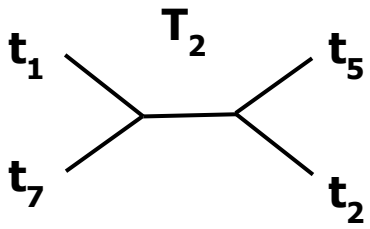
Simple Example

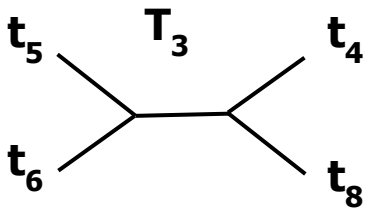
Missing Taxa: t_7, t_8

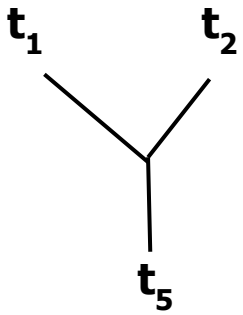
- Build **common subtree** between **T** and T_2







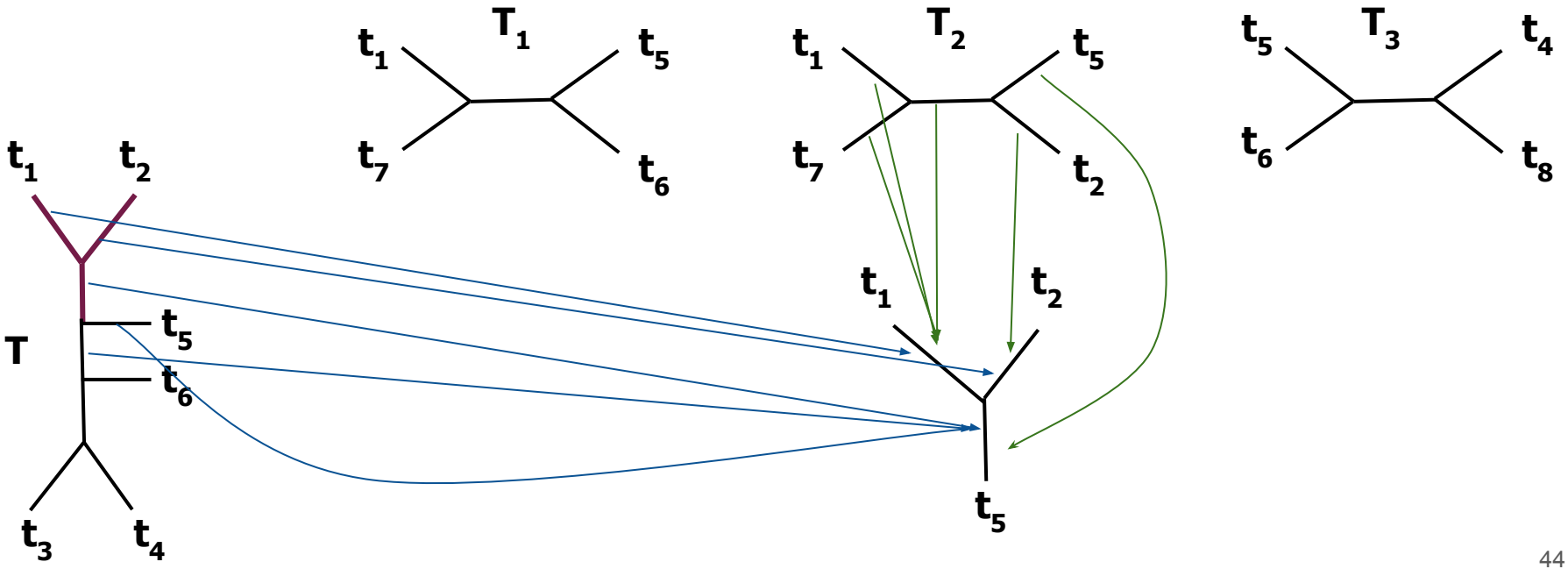




Simple Example

Missing Taxa: t_7, t_8

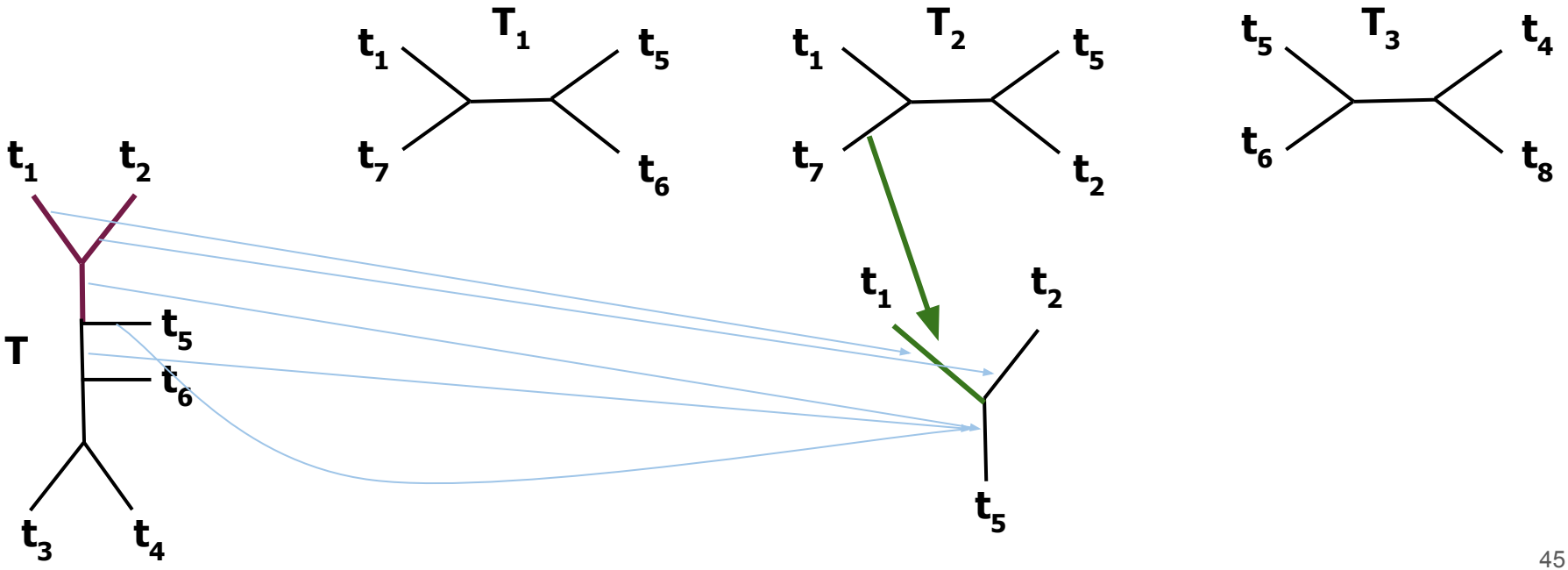
- **Map edges** (I'm omitting some mappings for simplicity)



Simple Example

Missing Taxa: t_7, t_8

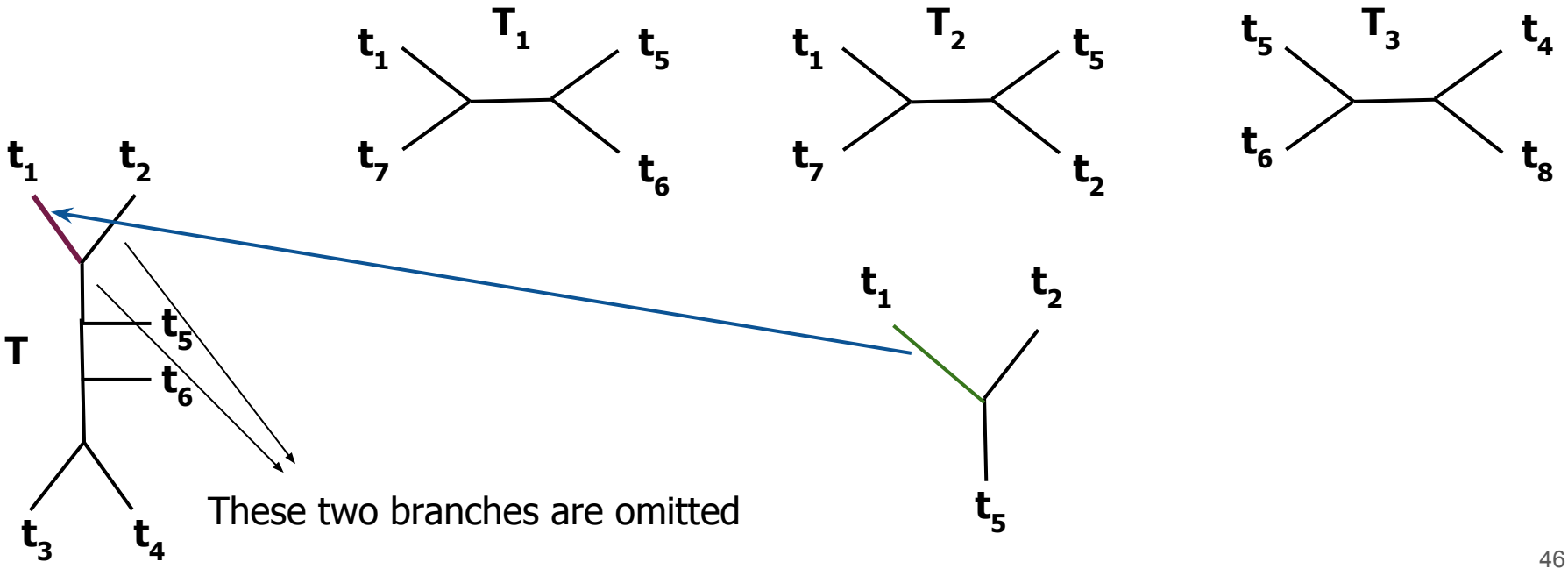
- Find the edge in which t_7 is mapped (it is only one edge)



Simple Example

Missing Taxa: t_7, t_8

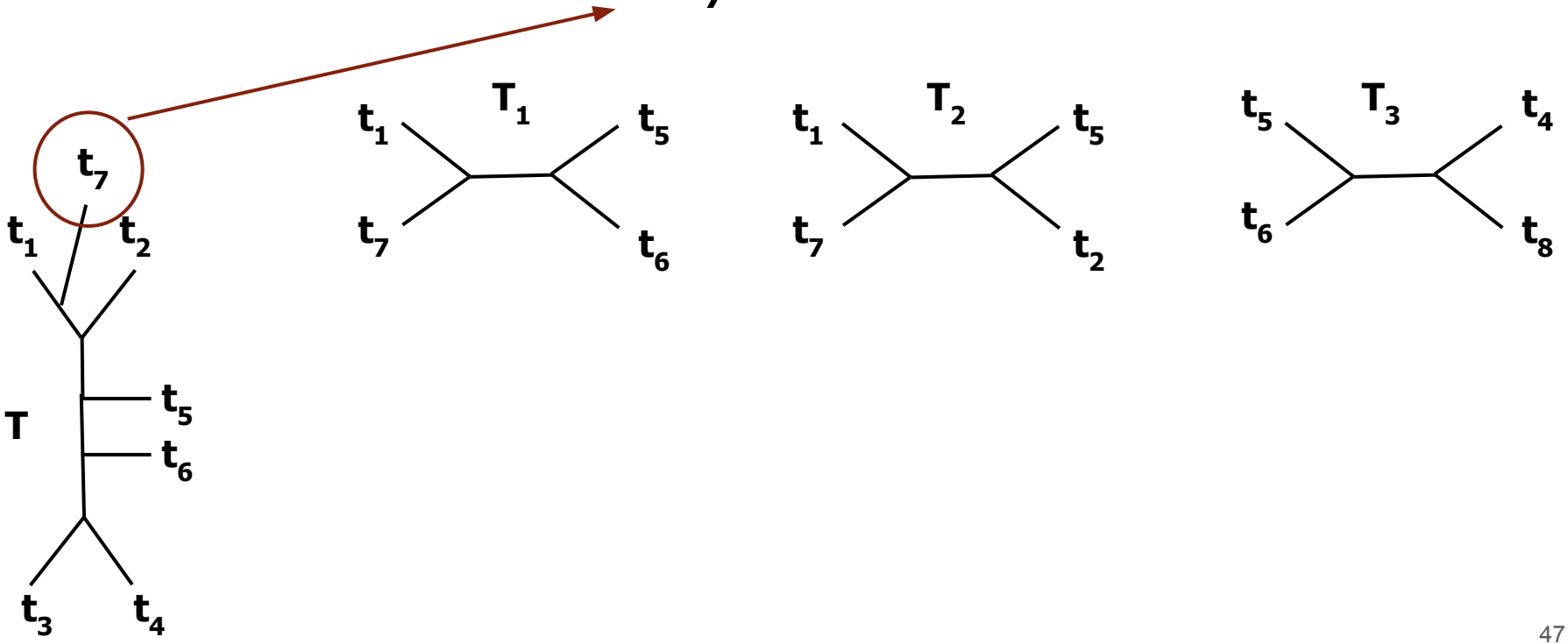
- Map backwards (**take the intersection**)



Simple Example

Missing Taxa: t_7, t_8

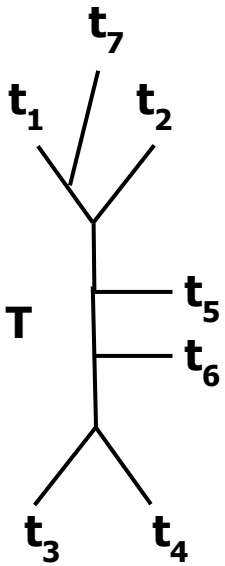
- **One** admissible branch, **add t_7**

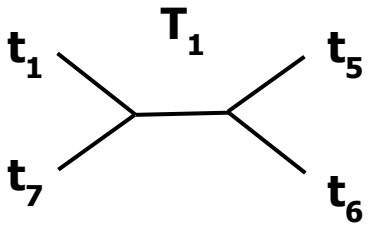


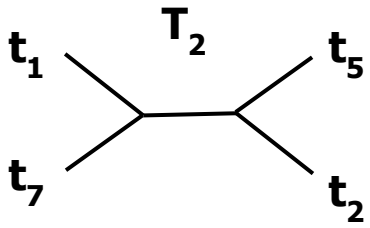
Simple Example

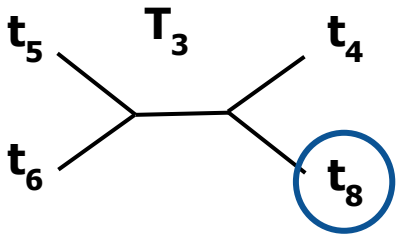
Missing Taxa: ~~t_7~~, t_8

- t_8 is present on constraint tree T_3





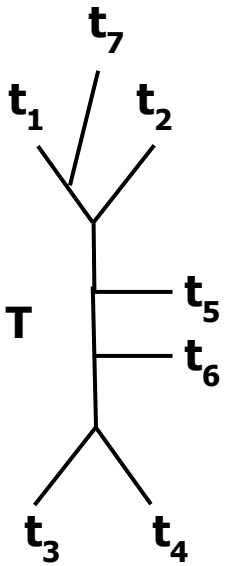


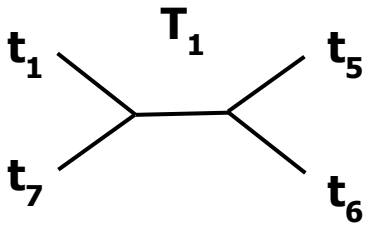


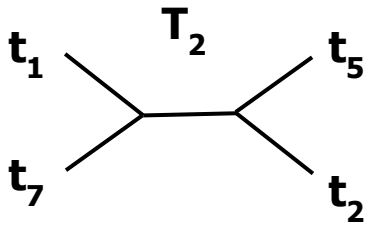
Simple Example

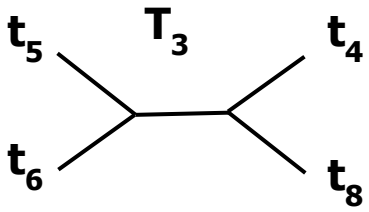
Missing Taxa: ~~t_7~~, t_8

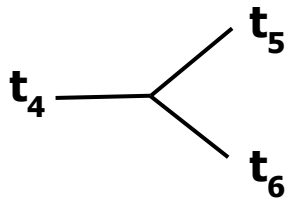
- Build **common subtree** between **T** and T_3







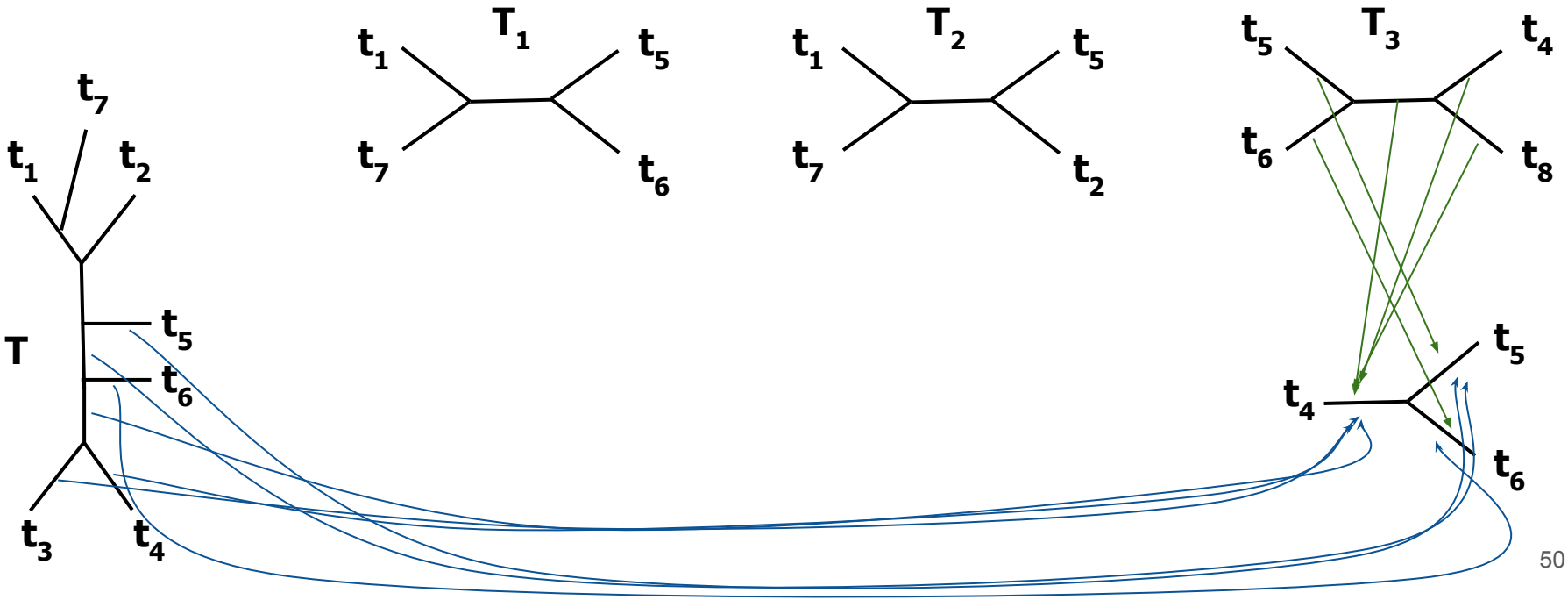




Simple Example

Missing Taxa: ~~t_7~~, t_8

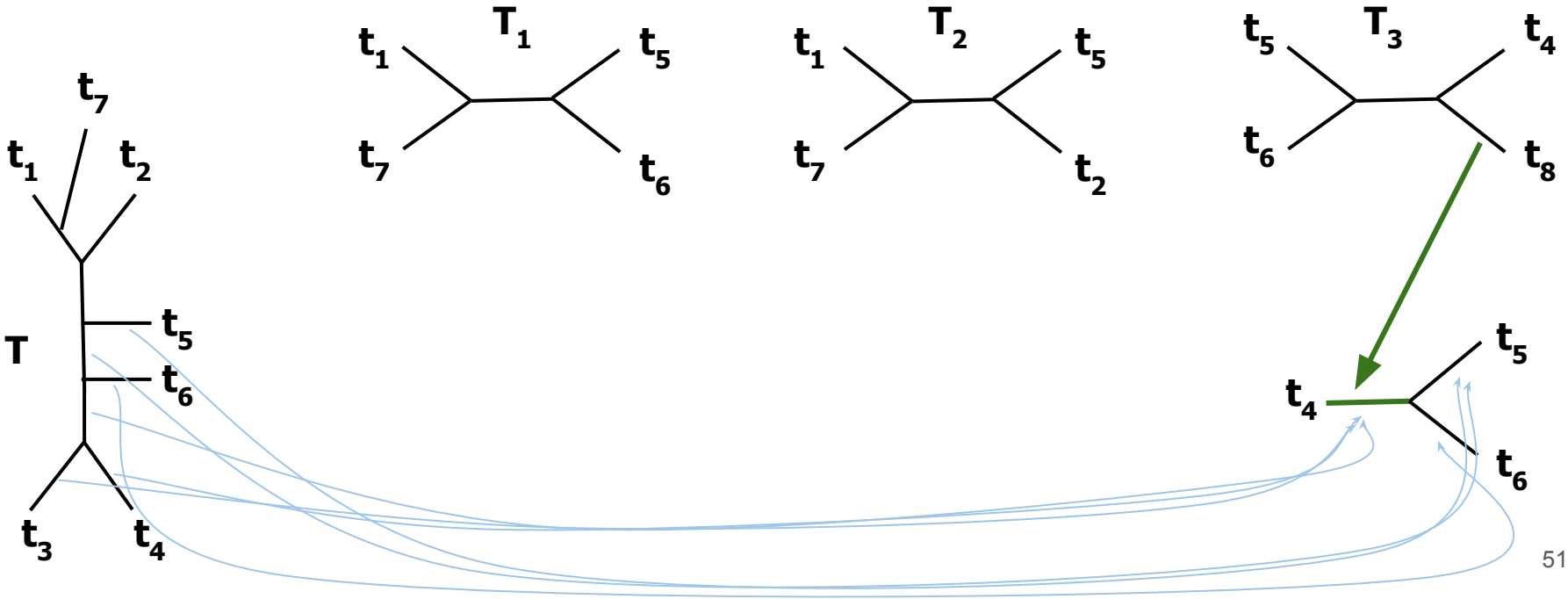
- **Map edges** (I'm omitting some mappings for simplicity)



Simple Example

Missing Taxa: ~~t_7~~, t_8

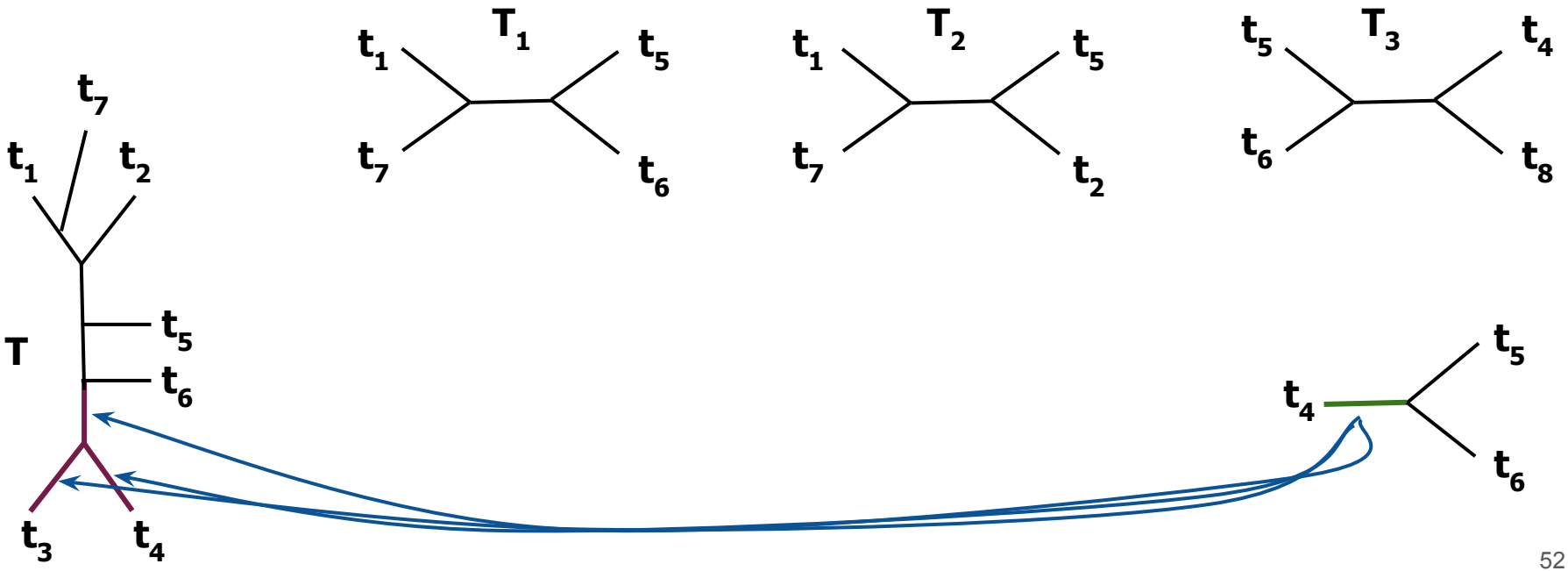
- Find the edge in which t_8 is mapped (it is only one edge)



Simple Example

Missing Taxa: ~~t_7~~, t_8

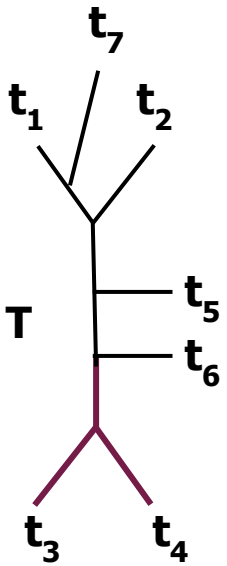
- Map backwards

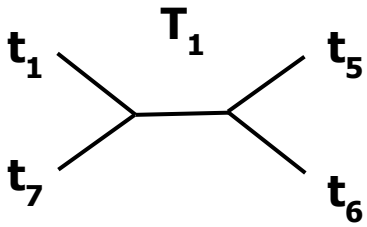


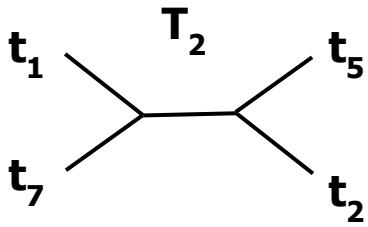
Simple Example

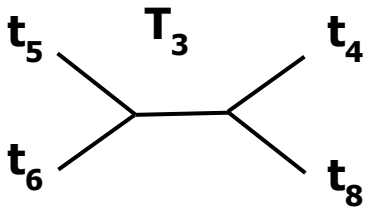
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)







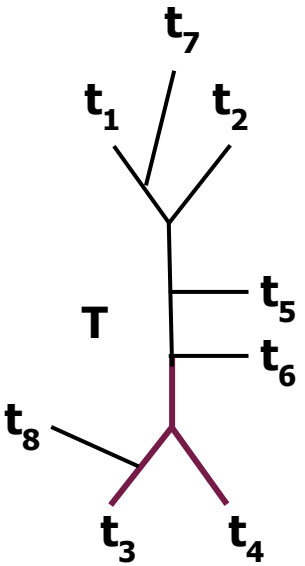


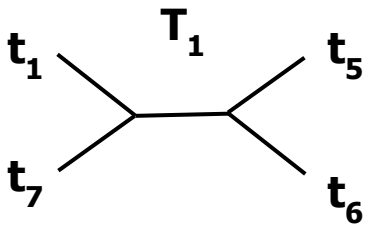
Stand Trees = 0

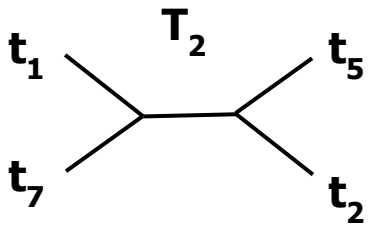
Simple Example

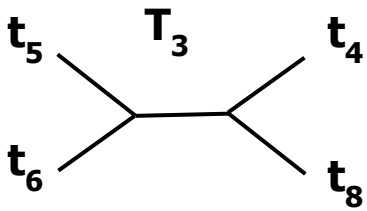
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)









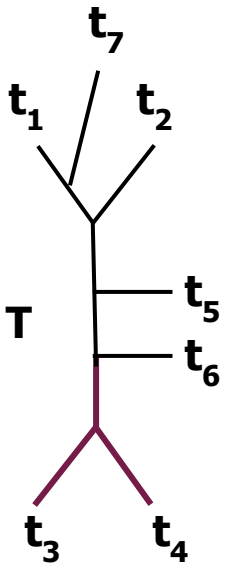
Add

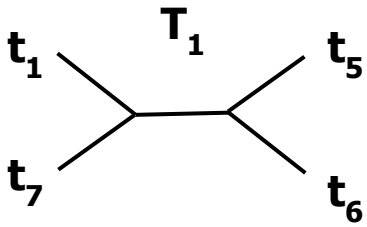
Stand Trees = 1

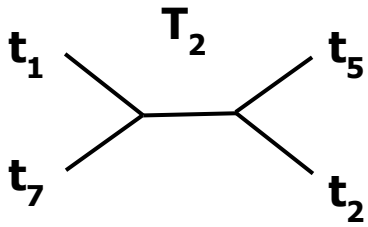
Simple Example

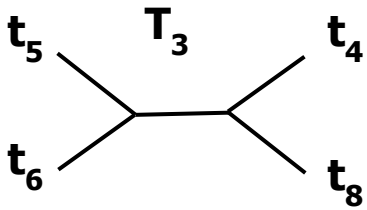
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)









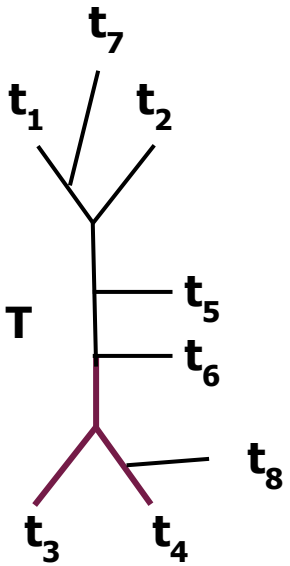
Remove

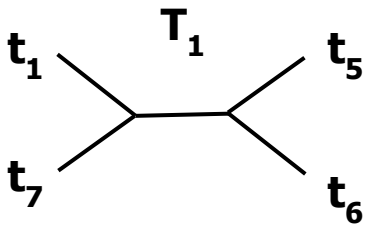
Stand Trees = 1

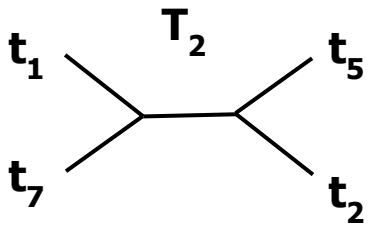
Simple Example

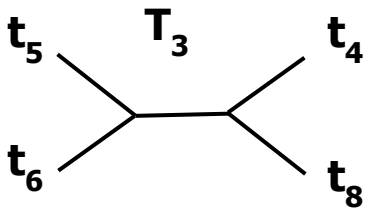
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)









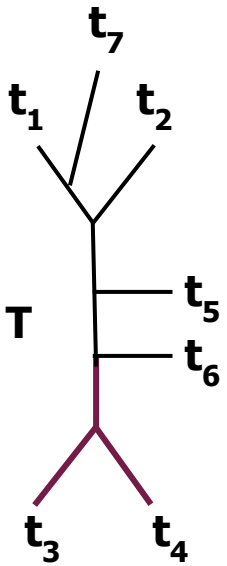
Add

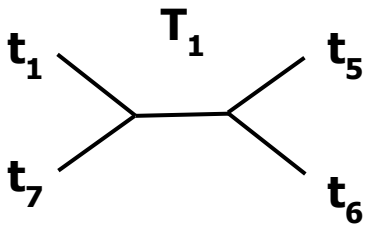
Stand Trees = 2

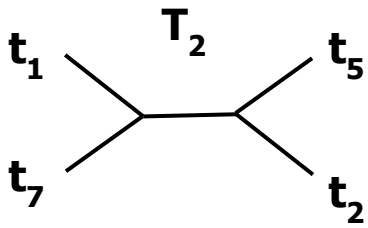
Simple Example

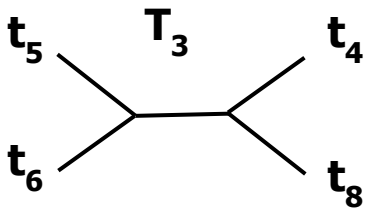
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)









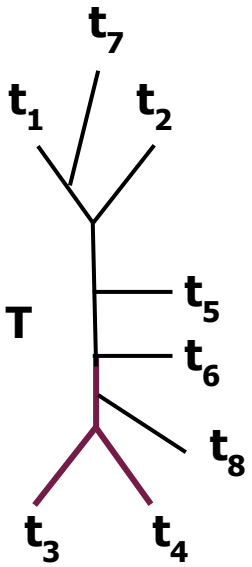
Remove

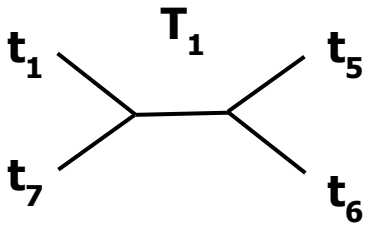
Stand Trees = 2

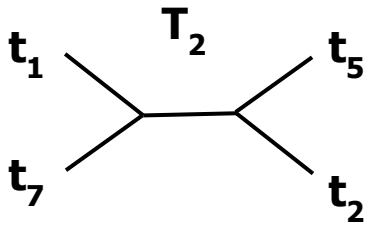
Simple Example

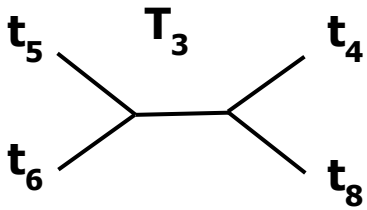
Missing Taxa: ~~t_7~~, t_8

- **Three** admissible branches for t_8 (now **branch and bound**)









Add

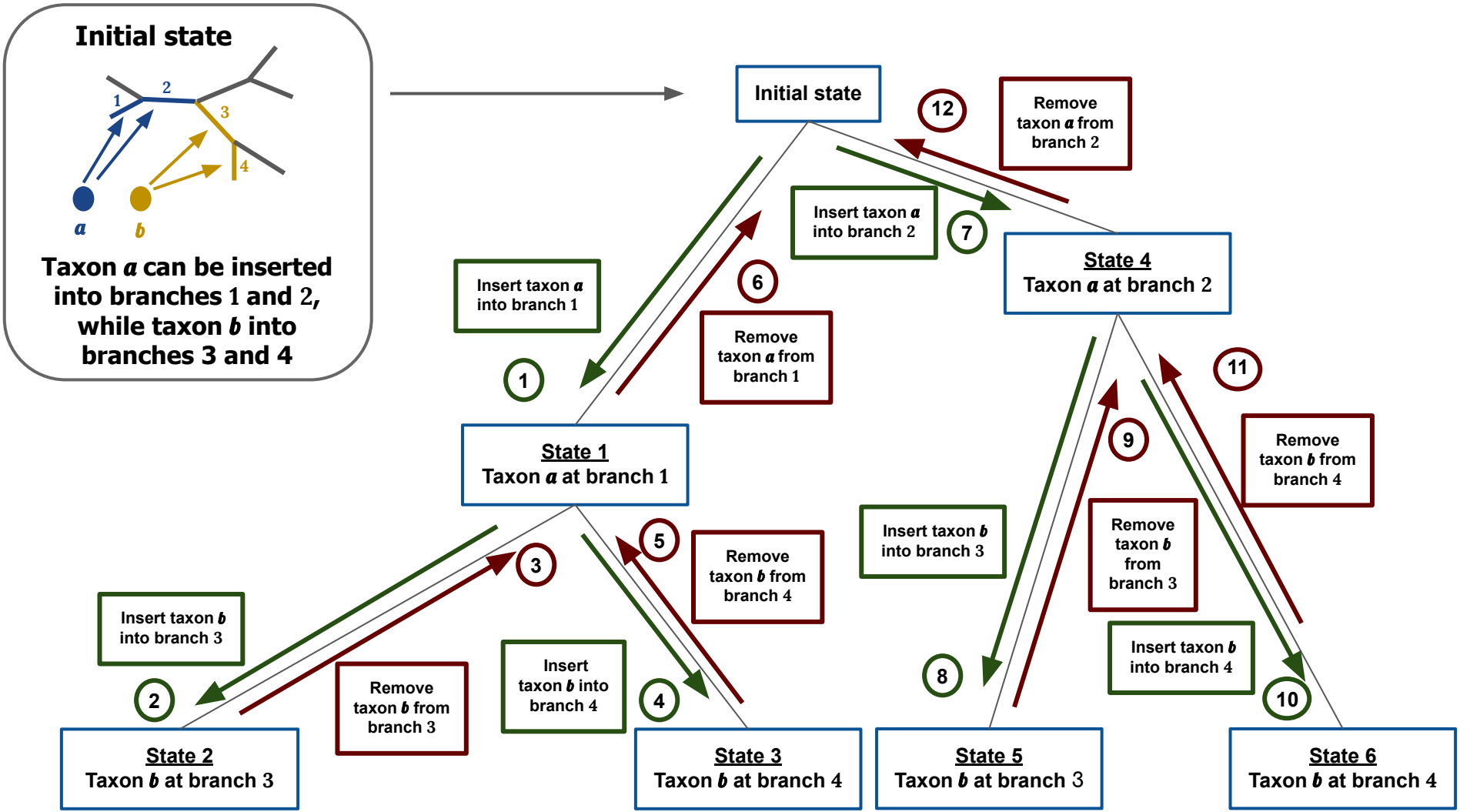
Stand Trees = 3

Three stand trees in total 🥳🥳🥳

Types of States

State = agile tree + constraint trees

- If the **agile tree** is **incomplete** → **intermediate state**
- If the **agile tree** is **complete** → **stand tree**
- If the **agile tree is incomplete** but **no more taxa can be inserted**, because compatibility is violated: → **dead end**

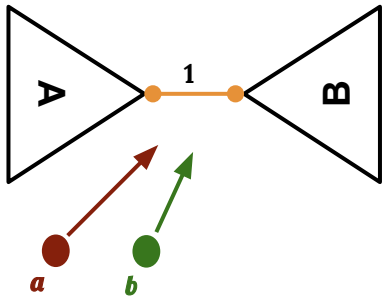

Initial state
1
2
3
4
a
b
Taxon a can be inserted into branches 1 and 2, while taxon b into branches 3 and 4
Initial state
12
Remove taxon a from branch 2
Insert taxon a into branch 2
7
State 4
Taxon a at branch 2
Insert taxon a into branch 1
6
Remove taxon a from branch 1
1
State 1
Taxon a at branch 1
11
Remove taxon b from branch 4
9
Insert taxon b into branch 3
Remove taxon b from branch 3
Insert taxon b into branch 4
5
Remove taxon b from branch 4
3
Insert taxon b into branch 3
8
10
2
Remove taxon b from branch 3
Insert taxon b into branch 4
4
State 2
Taxon b at branch 3
State 3
Taxon b at branch 4
State 5
Taxon b at branch 3
State 6
Taxon b at branch 4

Comments

- Gentrius is a branch and bound algorithm
- Thus, the **workflow graph** (graph of **states**) itself has a **tree structure**
- We **cannot know *a priori* the admissible branches** for all taxa

Comments

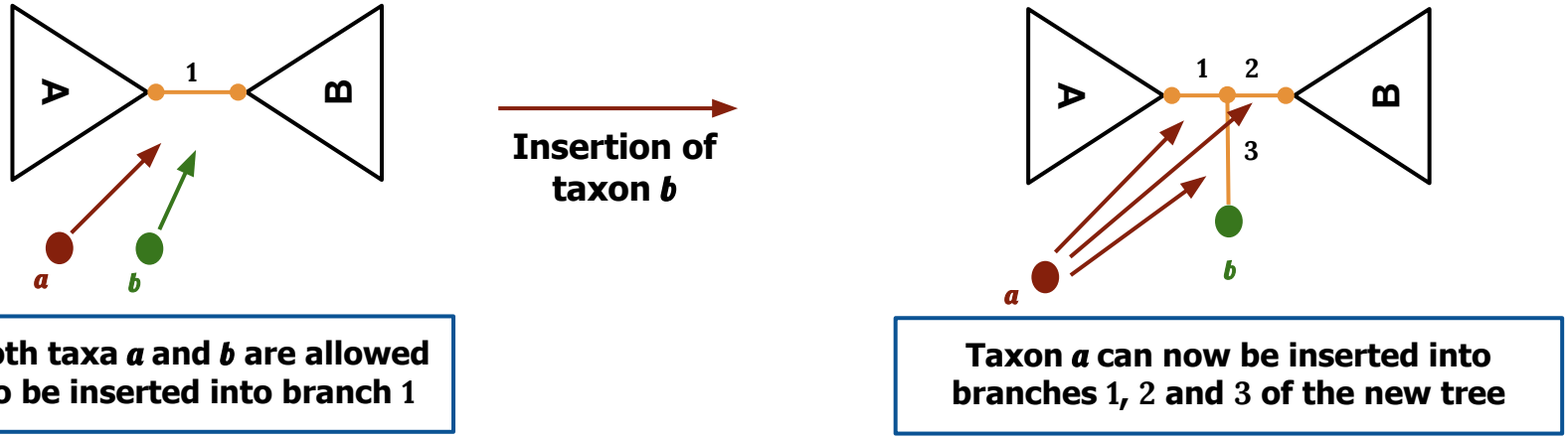
- Gentrius is a branch and bound algorithm
- Thus, the **workflow graph** (graph of **states**) itself has a **tree structure**
- We **cannot know *a priori* the admissible branches** for all taxa



Both taxa *a* and *b* are allowed to be inserted into branch 1

Comments

- Gentrius is a branch and bound algorithm
- Thus, the **workflow graph** (graph of **states**) itself has a **tree structure**
- We **cannot know *a priori* the admissible branches** for all taxa



Comments

- **Dynamic taxon insertion** heuristic
- Every time, the **taxon with the smallest number of admissible branches** is selected for insertion

Stopping Rules

- In the **worst-case scenario**, the **number of trees on a stand** can be **exponentially many**
- To prevent excessive runtimes, Gentrius employs **three stopping rules**:
  - when the algorithm counts **more than N stand trees** (default $N=10^6$)
  - when **more than M intermediate states** have been visited (default $M=10^8$)
  - when the execution requires **more than T hours** (default T=168h)

Contents

- **Compatibility**
- **Gentrius Algorithm**
- **Why care about stands?**

Why care about stands?

- **Stands** are **associated** with the concept of **terraces**
- We say that **trees inferred from multi-partitioned MSAs** with **missing data** lie on a **terrace** when they have equal analytical score (ML score, quartet-consistency score)
- Under **certain criteria**, stand trees are also terrace trees
- Identifying stands is a quantification of the uncertainty of the tree inference under certain methods

Gentrius preprint

Title: Identifying equally scoring trees in phylogenomics with incomplete data using Gentrius

Authors: O. Chernomor[1]*, C. Elgert[1], A. von Haeseler[1,2]

Affiliations:

[1]Center for Integrative Bioinformatics Vienna (CIBIV), Max Perutz Laboratories, University of Vienna and Medical University of Vienna, Vienna Bio Center (VBC); Vienna, Austria.

[2]Department of computer science, University of Vienna; Vienna Austria.

*Corresponding author. Email: o.chernomor@gmail.com

Abstract: Phylogenetic trees are routinely built from huge and yet incomplete multi-locus datasets often leading to multiple equally scoring trees under many common criteria. As typical tree inference software output only a single tree, identifying all trees with identical score challenges phylogenomics. Here, we introduce Gentrius – an efficient algorithm that tackles this problem. We showed on simulated and biological datasets that Gentrius generates millions of trees within seconds. Depending on the distribution of missing data across species and loci and the inferred phylogeny, the number of equally good trees varies tremendously. The strict consensus tree computed from them displays all the branches unaffected by the pattern of missing data. Thus, Gentrius provides an important systematic assessment of phylogenetic trees inferred from incomplete data.

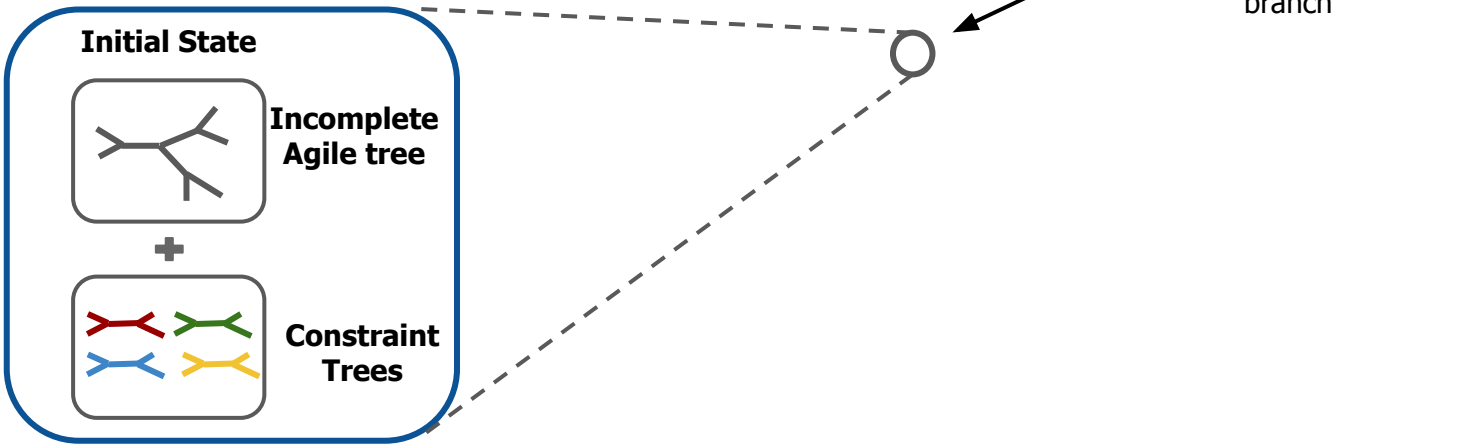
One-Sentence Summary: Gentrius - the algorithm to generate a complete stand, i.e. all binary unrooted trees compatible with the same set of subtrees.

Parallelization

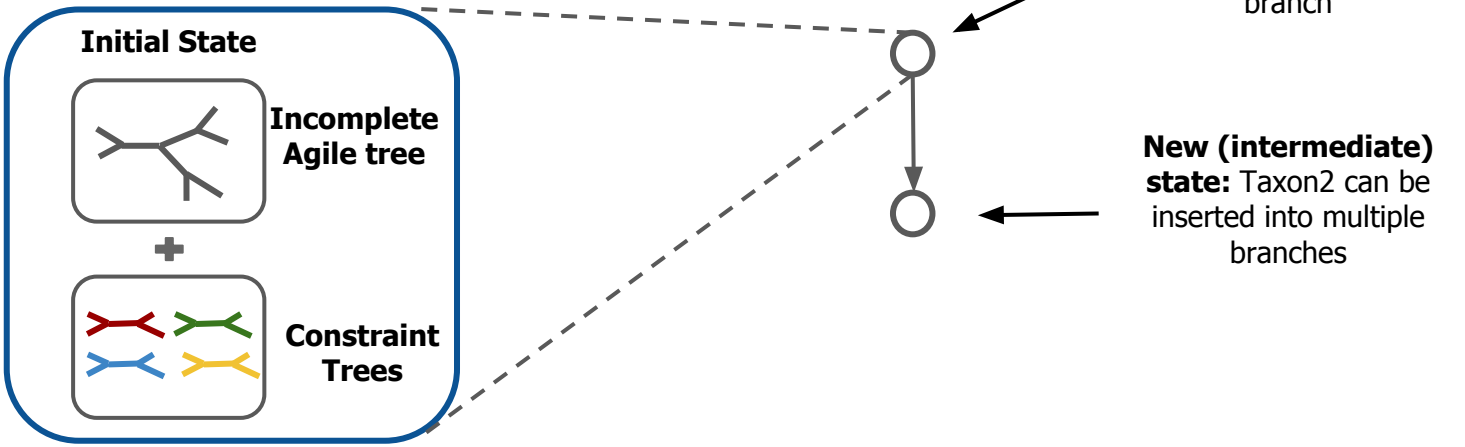
Preliminares

- The **workflow graph**, i.e. the graph of states of the algorithm, has a **treelike structure**
- **From now on**, **the trees in this presentation** will represent the **workflow graph** (tree of states), not phylogenetic trees

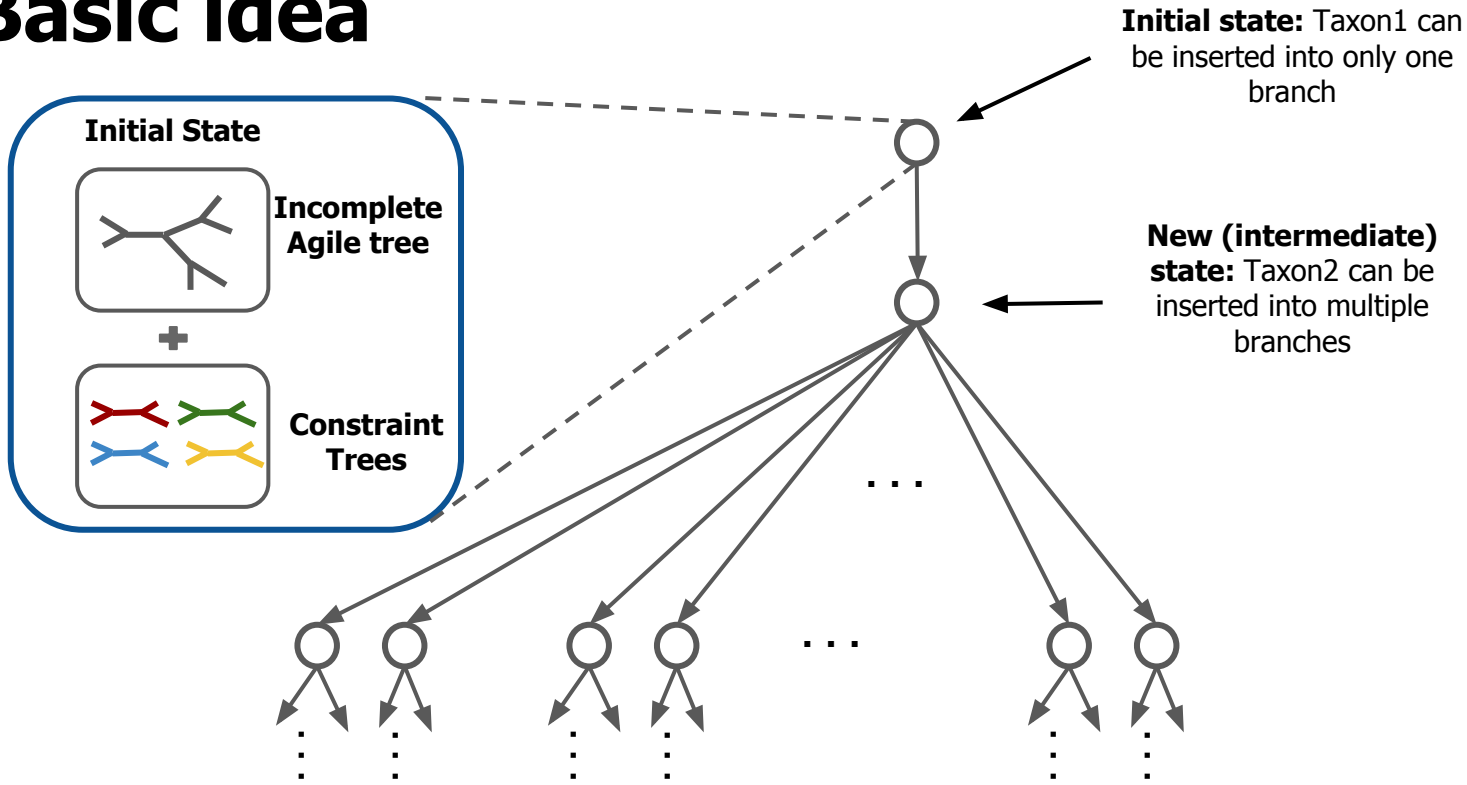
Basic idea


Initial State
Incomplete Agile tree
Constraint Trees
Initial state: Taxon1 can be inserted into only one branch

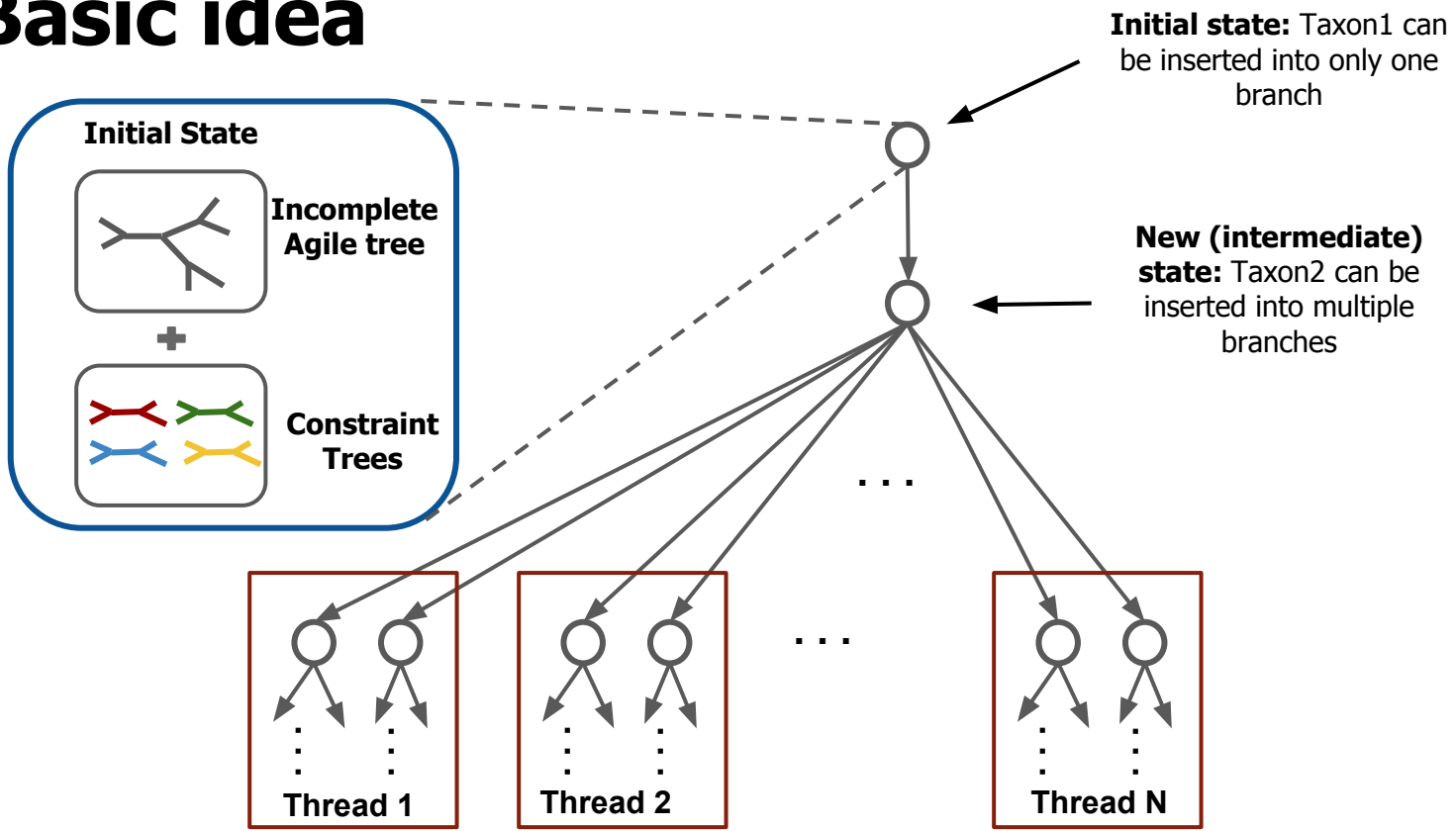
Basic idea


Initial State
Incomplete Agile tree
Constraint Trees
Initial state: Taxon1 can be inserted into only one branch
New (intermediate) state: Taxon2 can be inserted into multiple branches

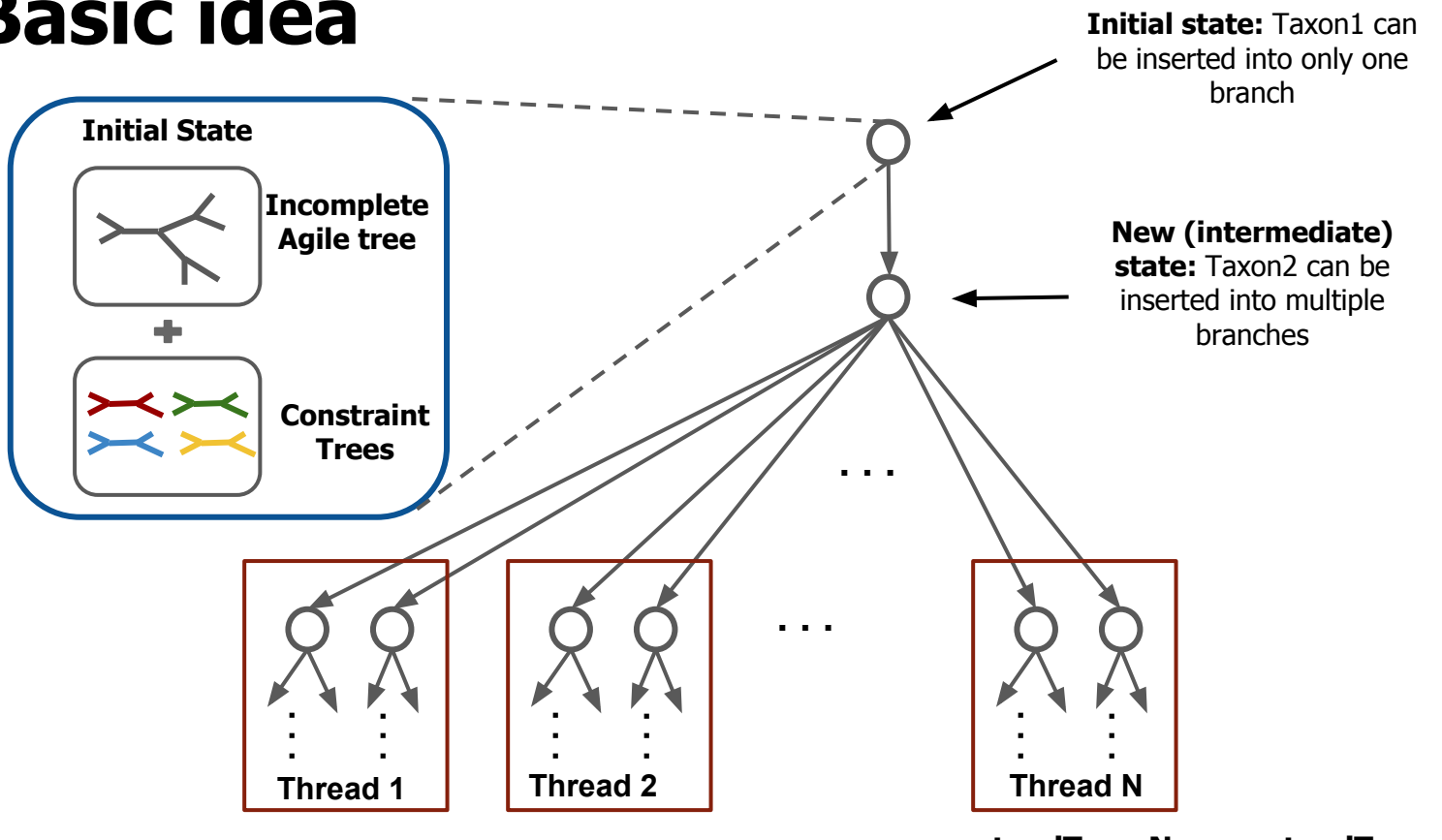
Basic idea


Initial State
Incomplete Agile tree
Constraint Trees
Initial state: Taxon1 can be inserted into only one branch
New (intermediate) state: Taxon2 can be inserted into multiple branches

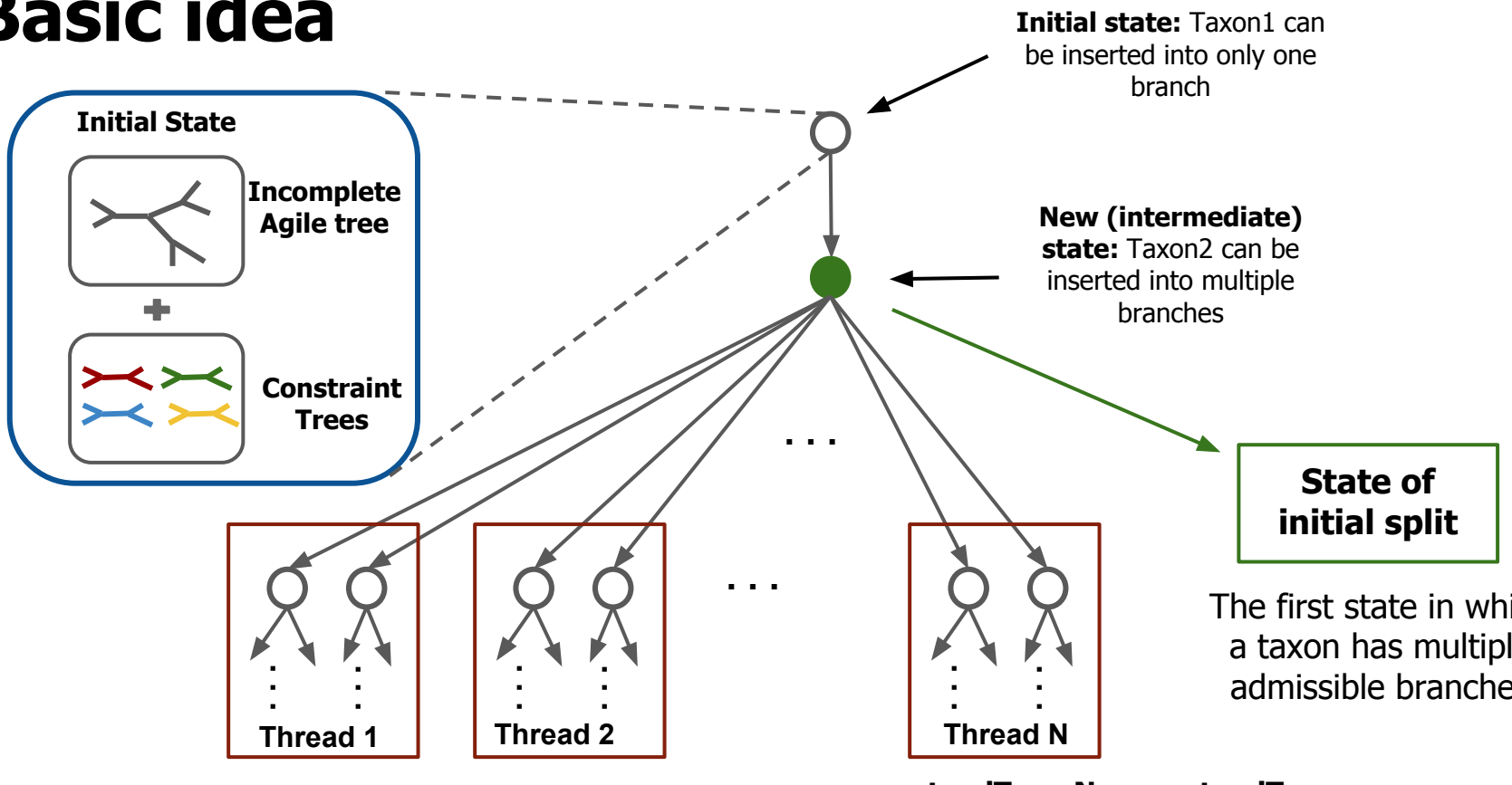
Basic idea


Initial state: Taxon1 can be inserted into only one branch
New (intermediate) state: Taxon2 can be inserted into multiple branches
Initial State
Incomplete Agile tree
Constraint Trees
Thread 1
Thread 2
Thread N

Basic idea

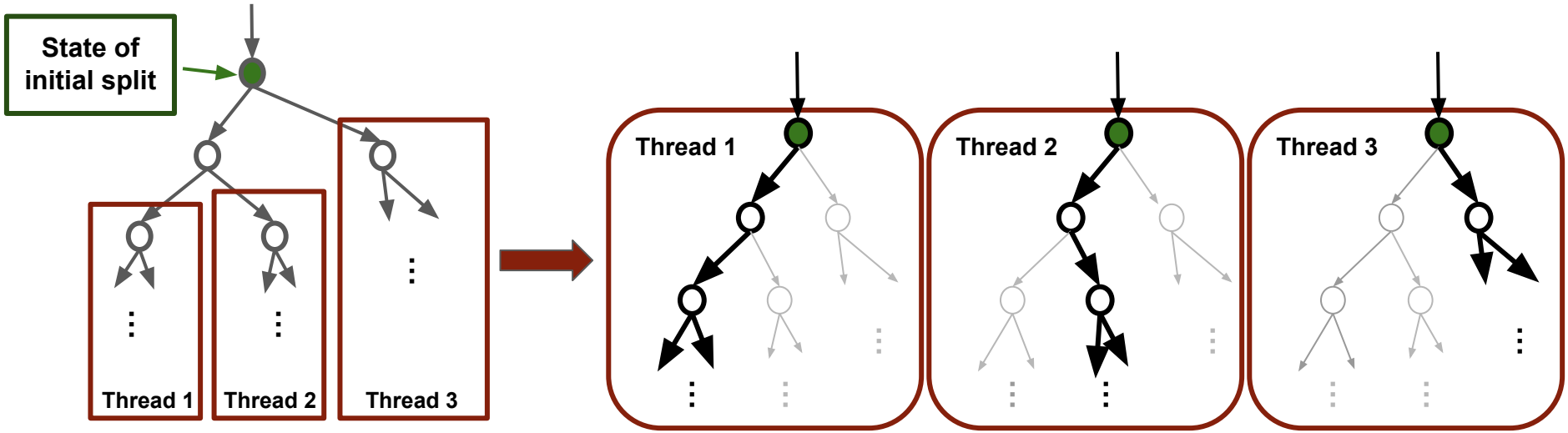

Initial state: Taxon1 can be inserted into only one branch
Initial State
Incomplete Agile tree
Constraint Trees
New (intermediate) state: Taxon2 can be inserted into multiple branches
Thread 1
Thread 2
Thread N
standTrees1 + standTrees2 + . . . + standTreesN = standTrees

Basic idea


Initial state: Taxon1 can be inserted into only one branch
Initial State
Incomplete Agile tree
Constraint Trees
New (intermediate) state: Taxon2 can be inserted into multiple branches
State of initial split
The first state in which a taxon has multiple admissible branches
Thread 1
Thread 2
Thread N
standTrees1 + standTrees2 + . . . + standTreesN = standTrees

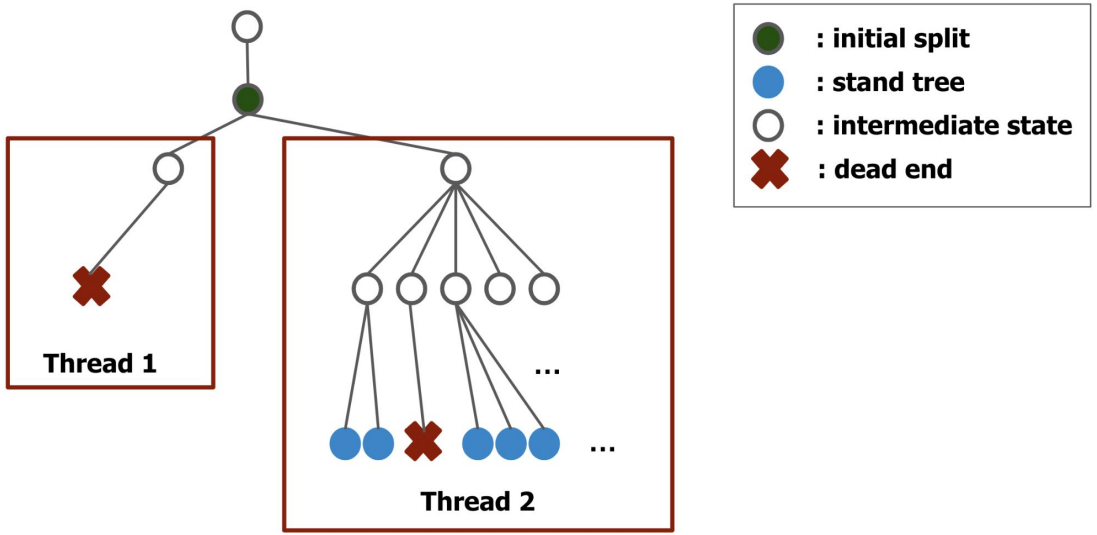
Alternative Parallelization Scheme

- When the **number of threads** used is **greater** than the **number of admissible branches** on the state of initial split



Problem

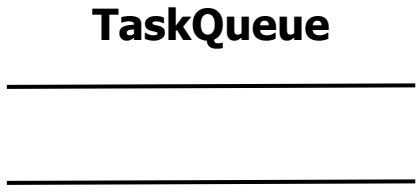
- The **workflow tree** is often **highly unbalanced**
- We **cannot know *a priori* its structure**, since we cannot know a priori the admissible branches for each taxon



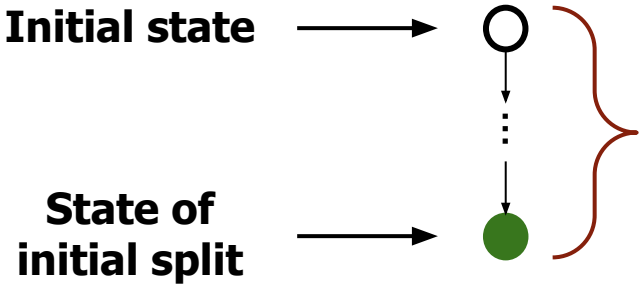
Solution

- We use a **thread pooling** approach
- We introduce a **taskqueue**
- **Active** (working) **threads** create and **push** new **tasks** into the queue
- **Inactive threads** (those that finished their jobs) **remain in busy-wait mode** until a **new task becomes available** in the queue.

Example



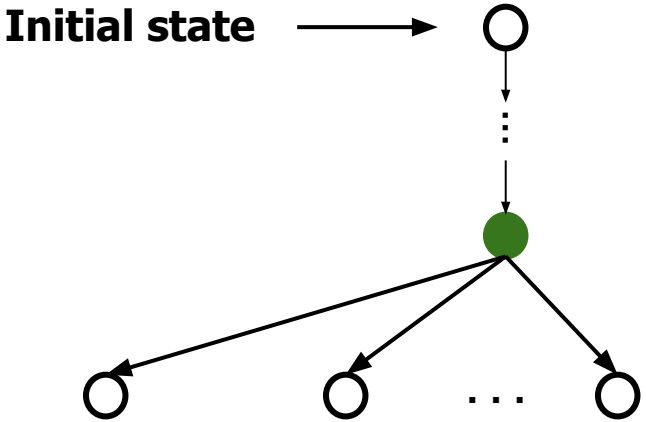
Example



Threads have not been separated yet

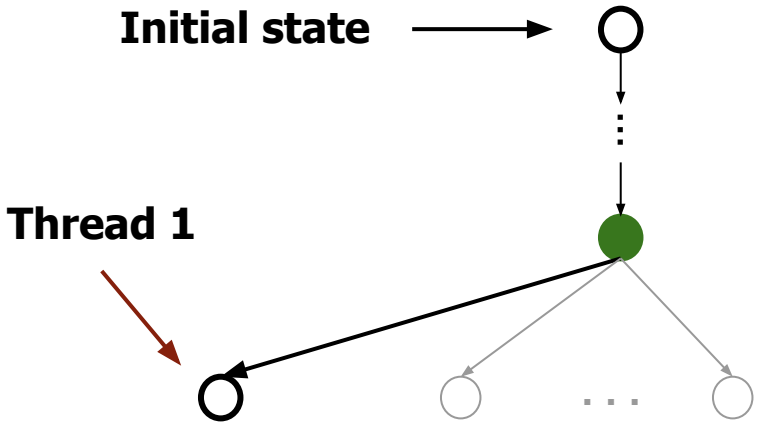
TaskQueue

Example



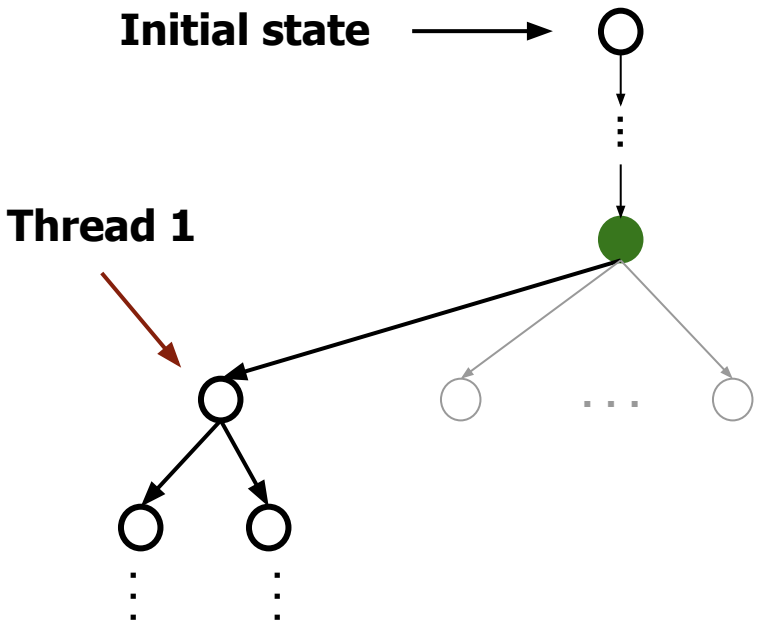
TaskQueue

Example



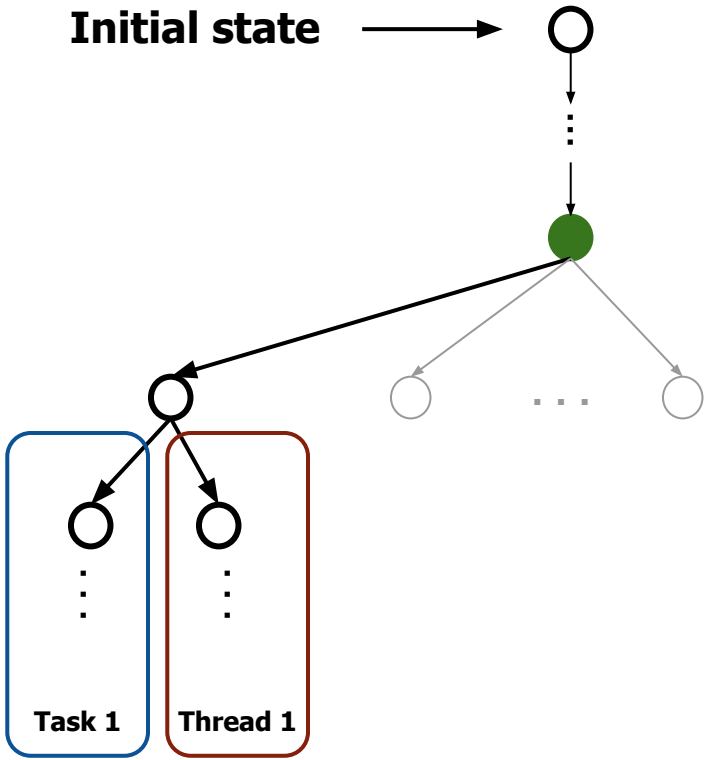
TaskQueue

Example



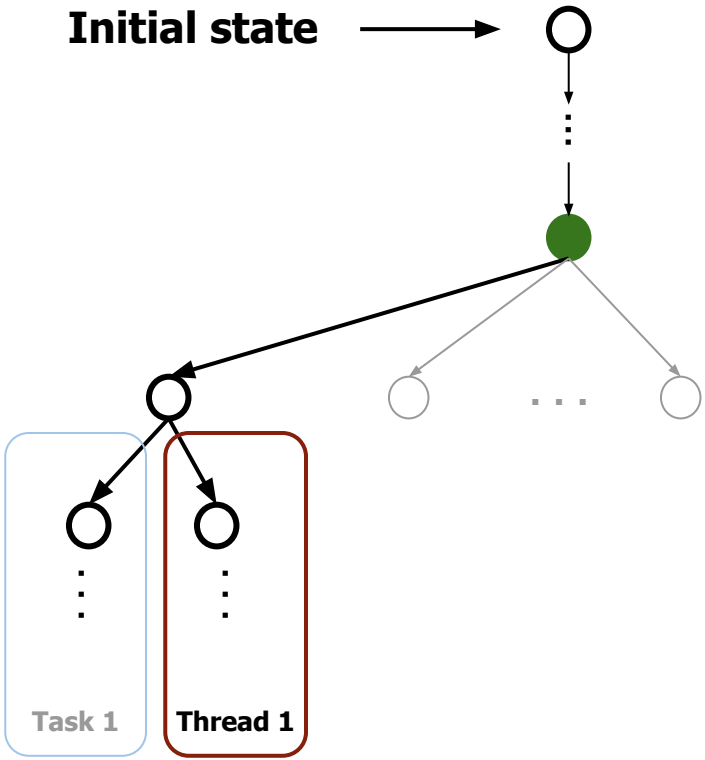
TaskQueue

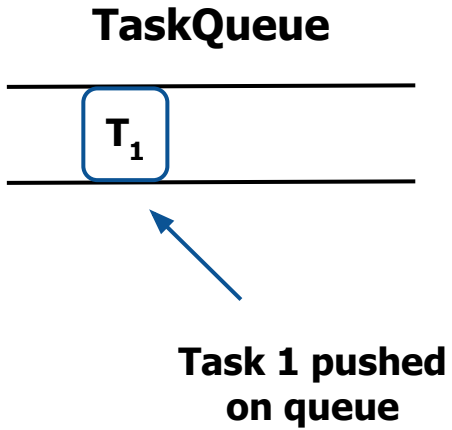
Example



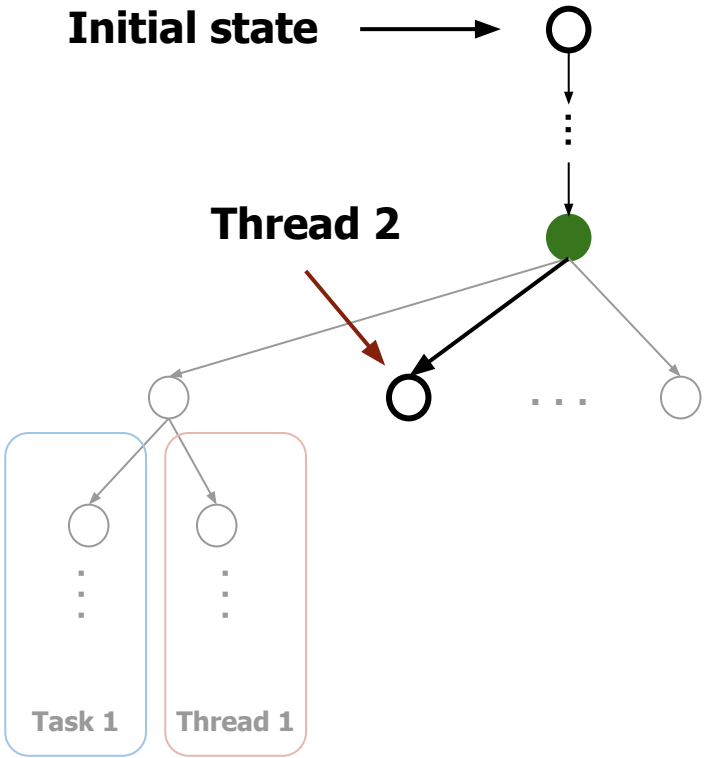
TaskQueue

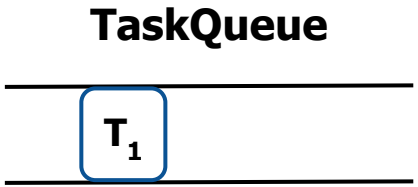
Example


Initial state
Task 1
Thread 1

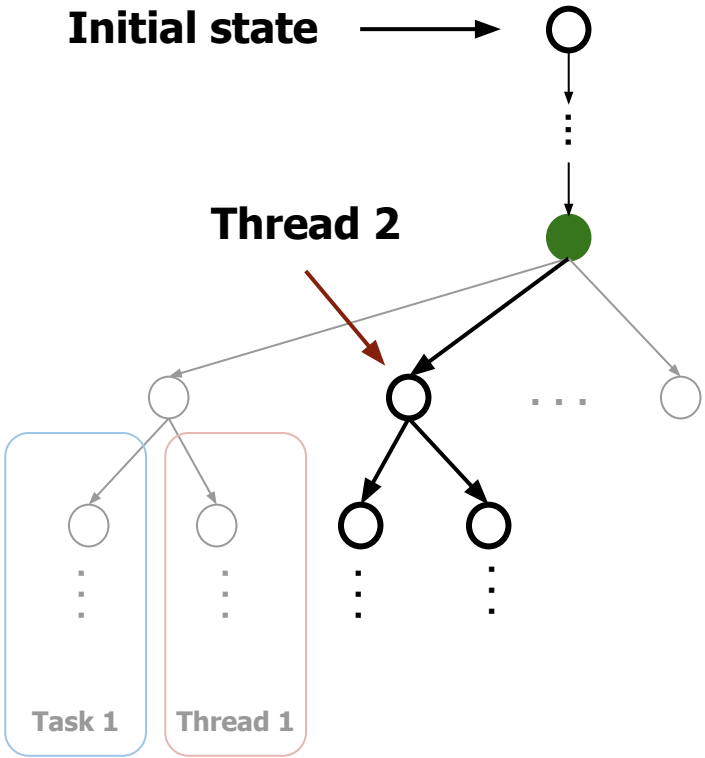

TaskQueue
T1
Task 1 pushed on queue

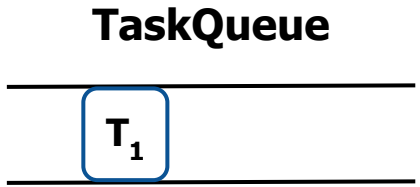
Example



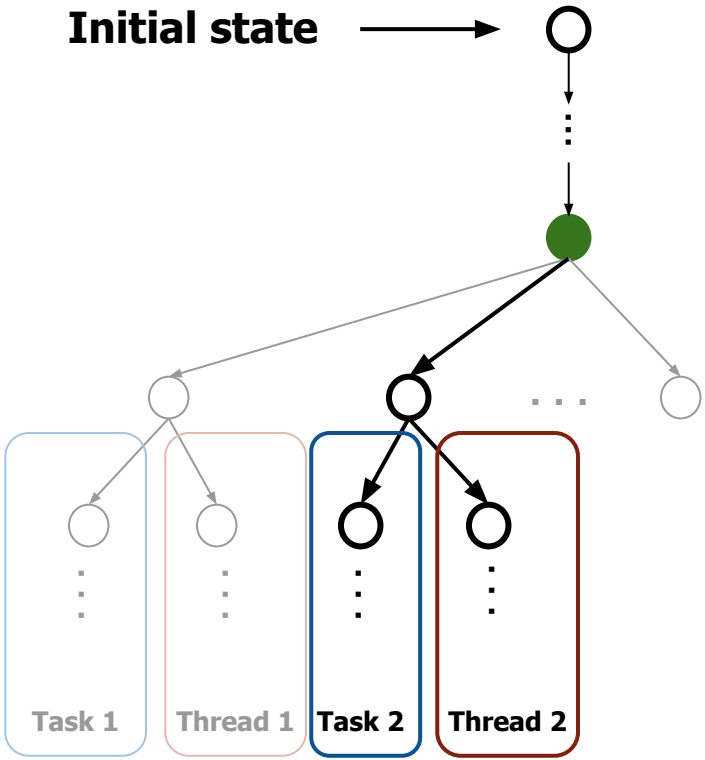


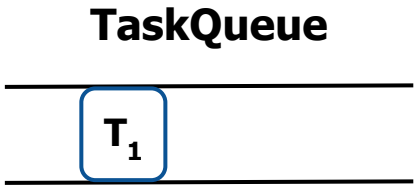
Example



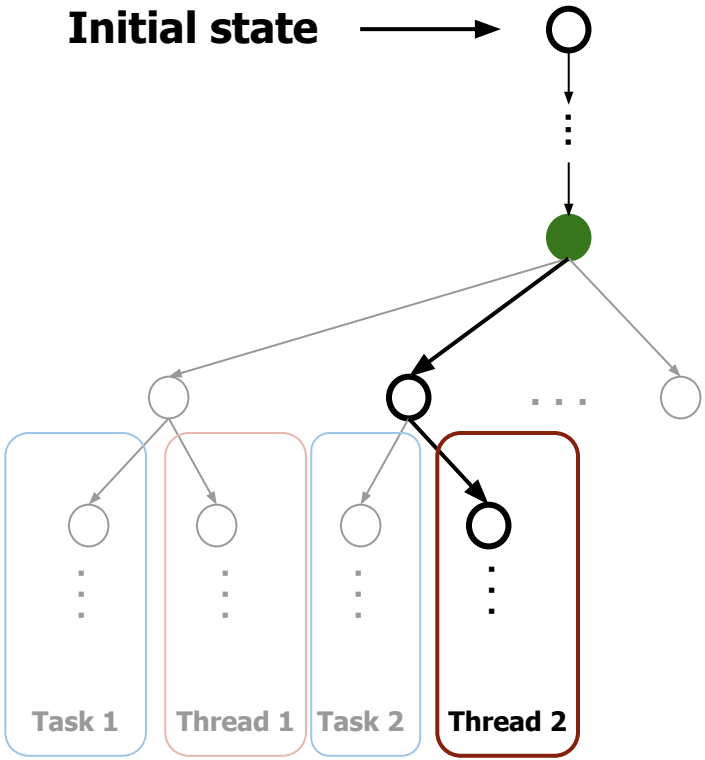


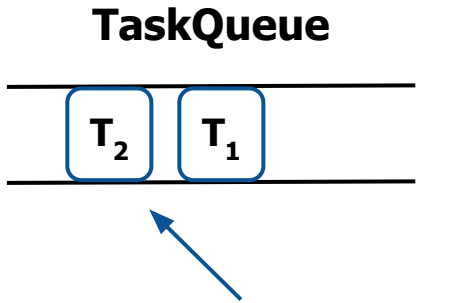
Example



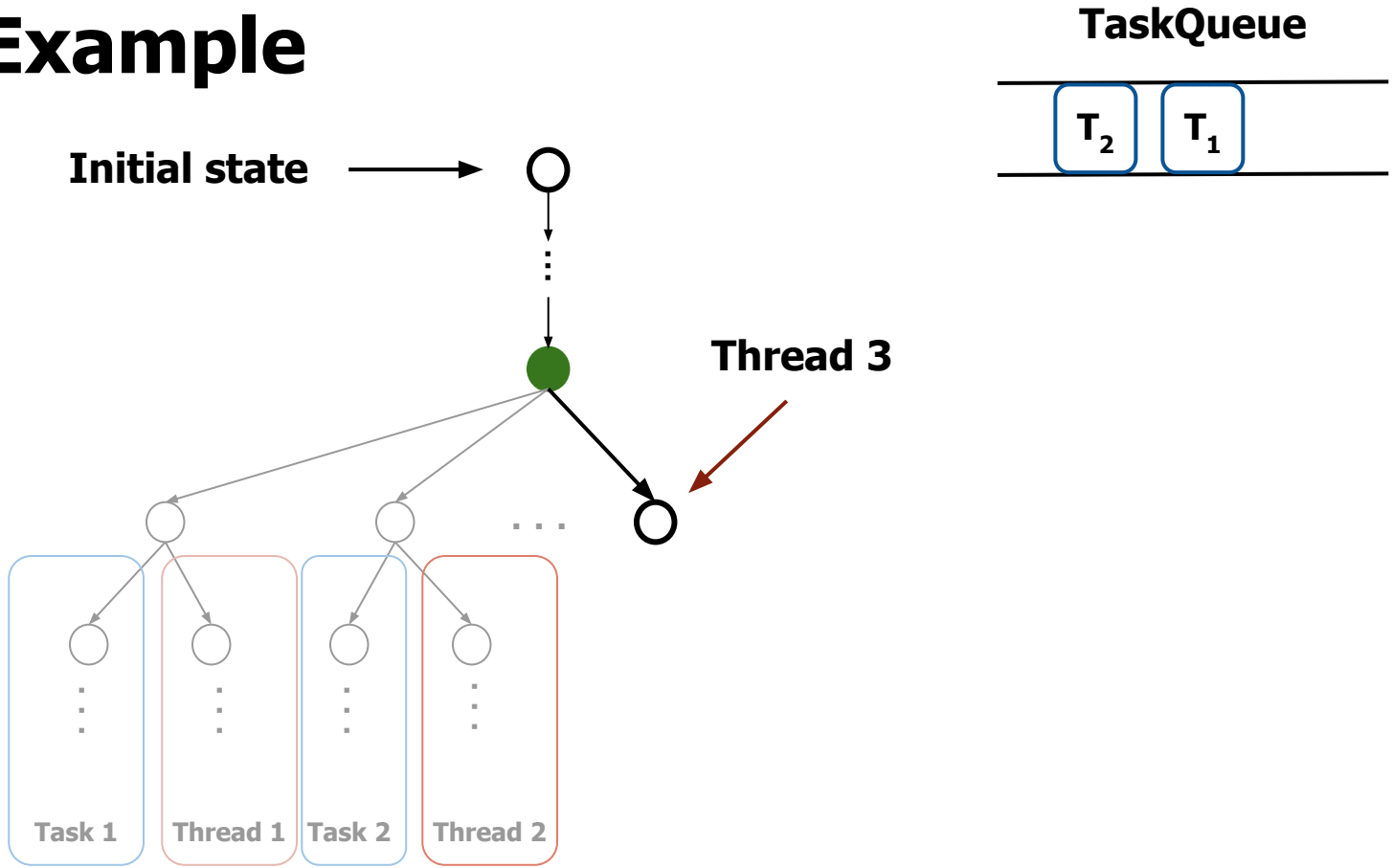


Example

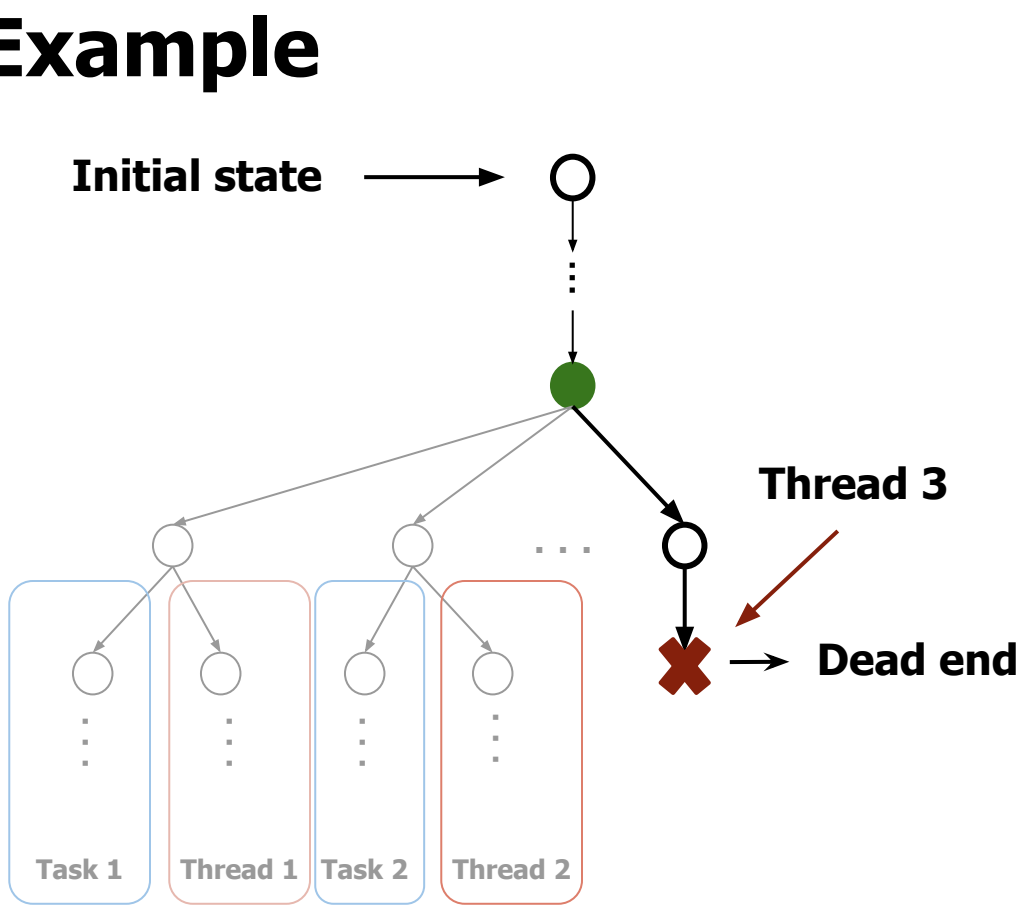

Initial state
Task 1
Thread 1
Task 2
Thread 2

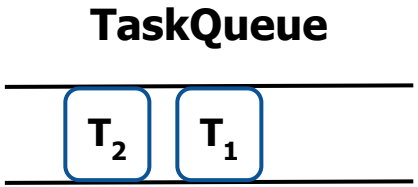

TaskQueue
T2
T1
Task 2 pushed on queue

Example

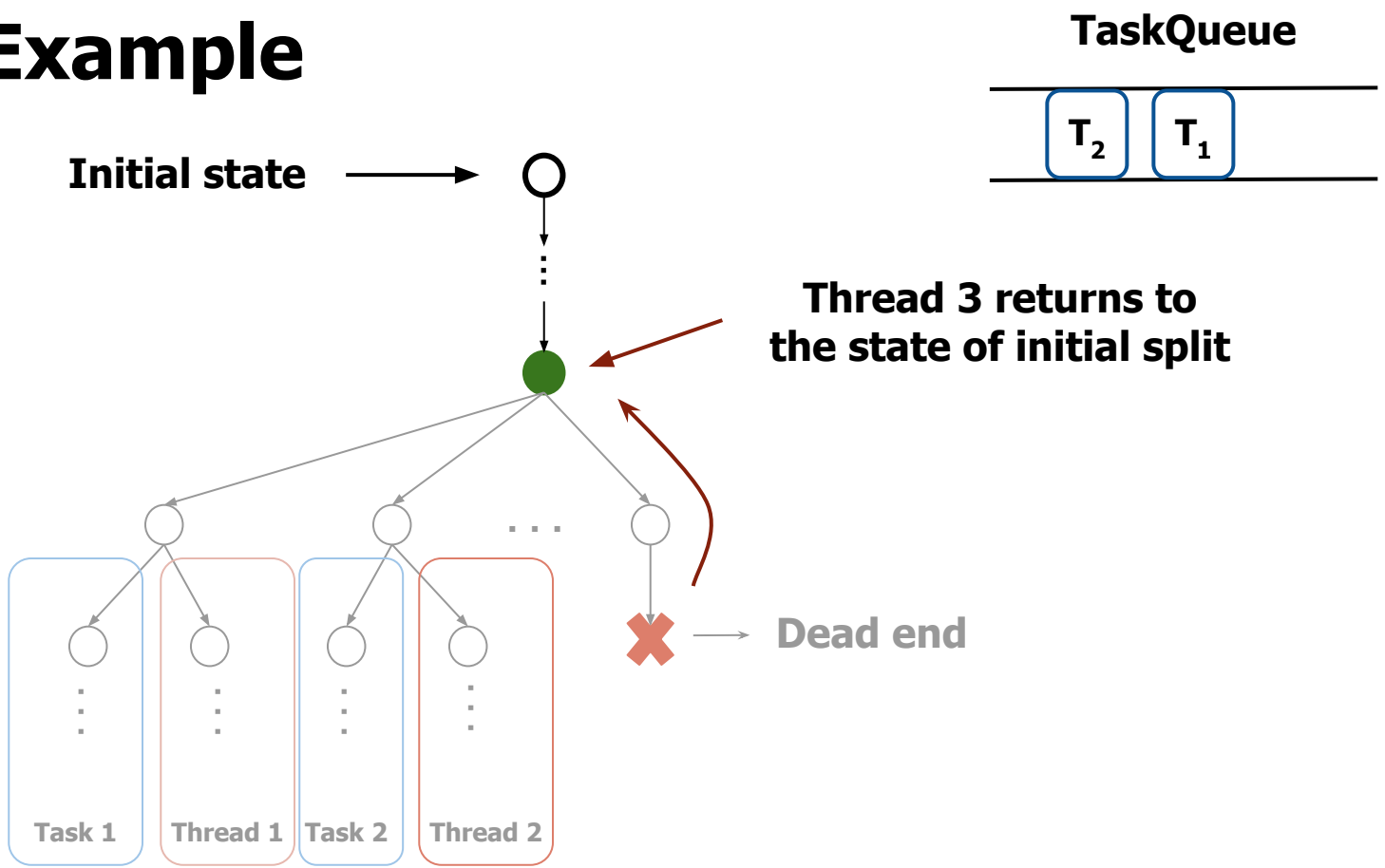

TaskQueue
T2
T1
Initial state
Thread 3
Task 1
Thread 1
Task 2
Thread 2

Example

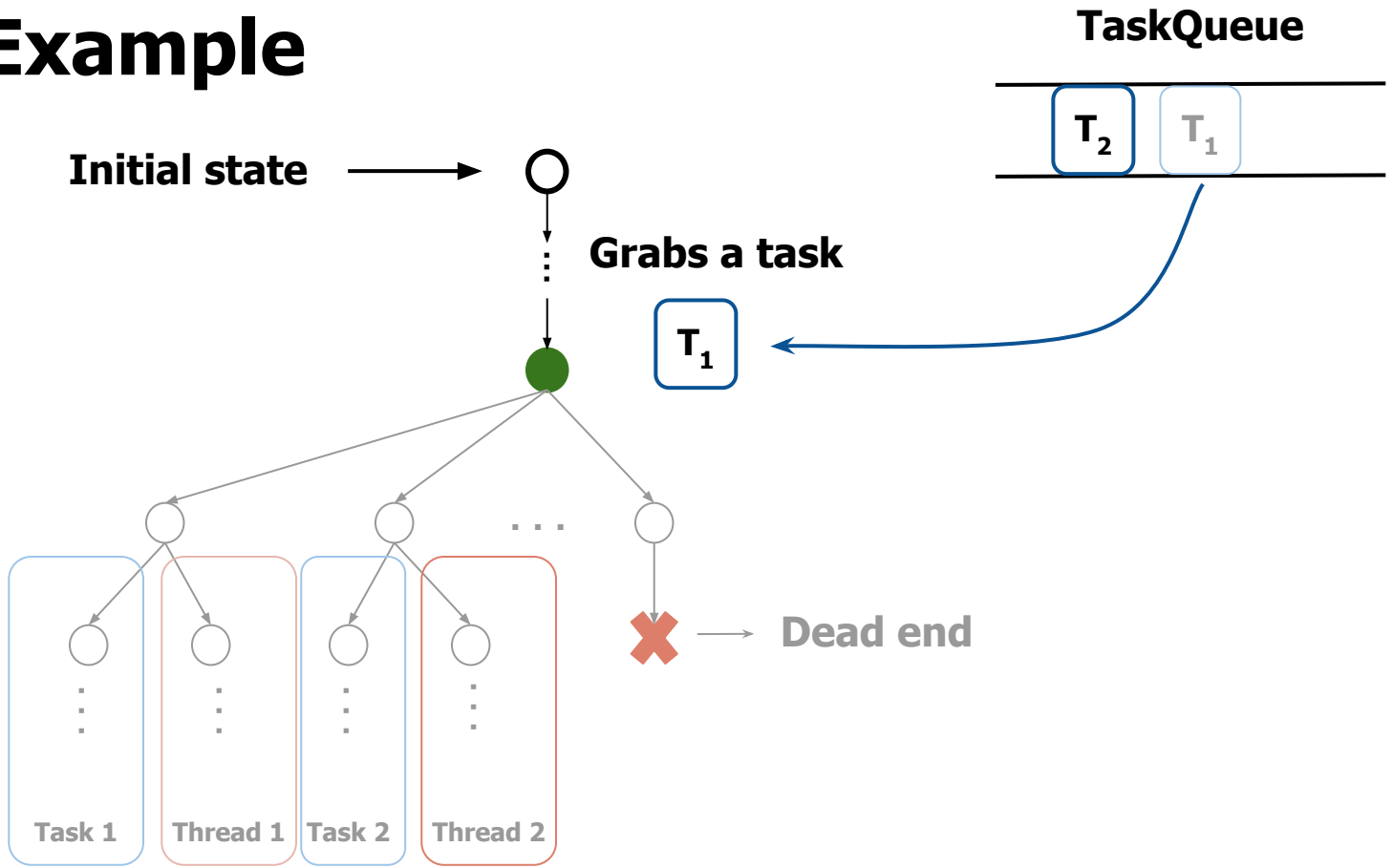

Initial state
Thread 3
Dead end
Task 1
Thread 1
Task 2
Thread 2


TaskQueue
T2
T1

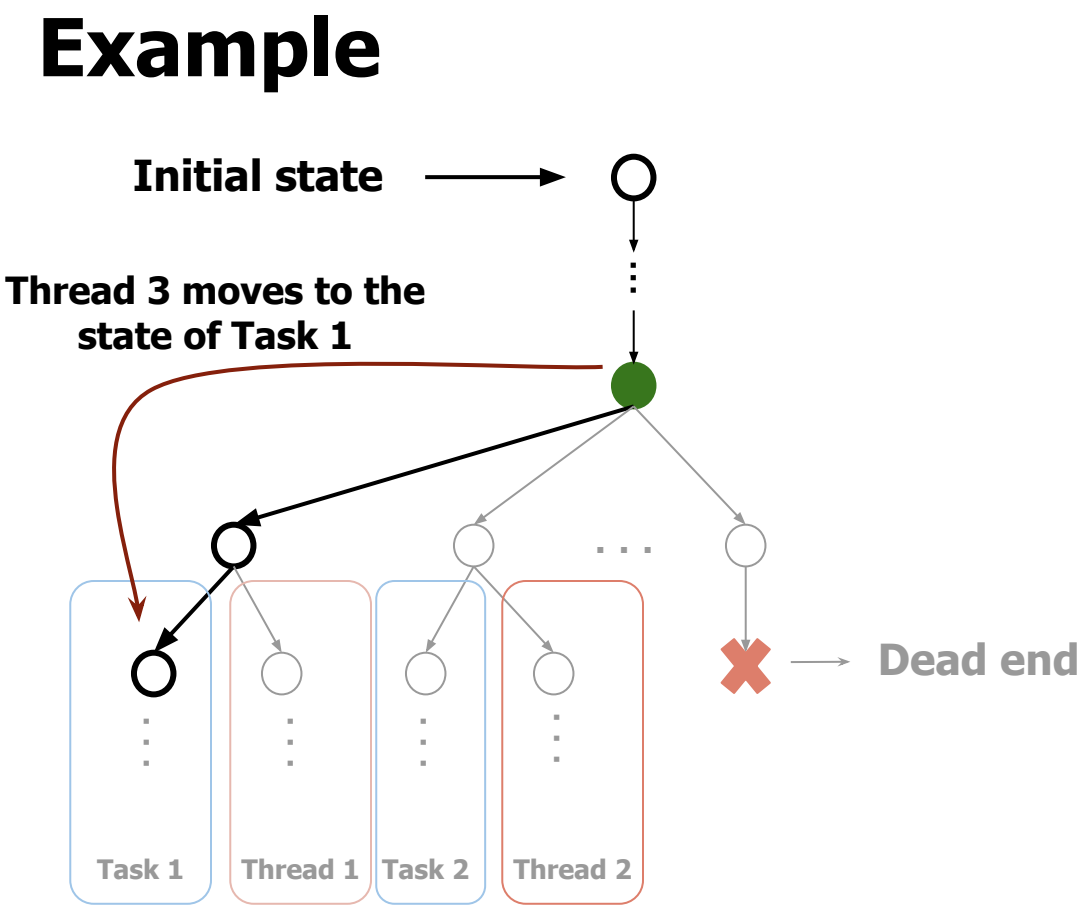
Example

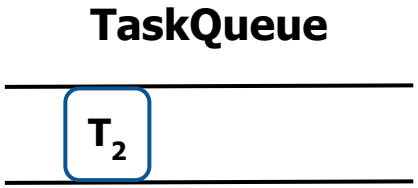

TaskQueue
T2
T1
Initial state
Thread 3 returns to
the state of initial split
Dead end
Task 1
Thread 1
Task 2
Thread 2

Example

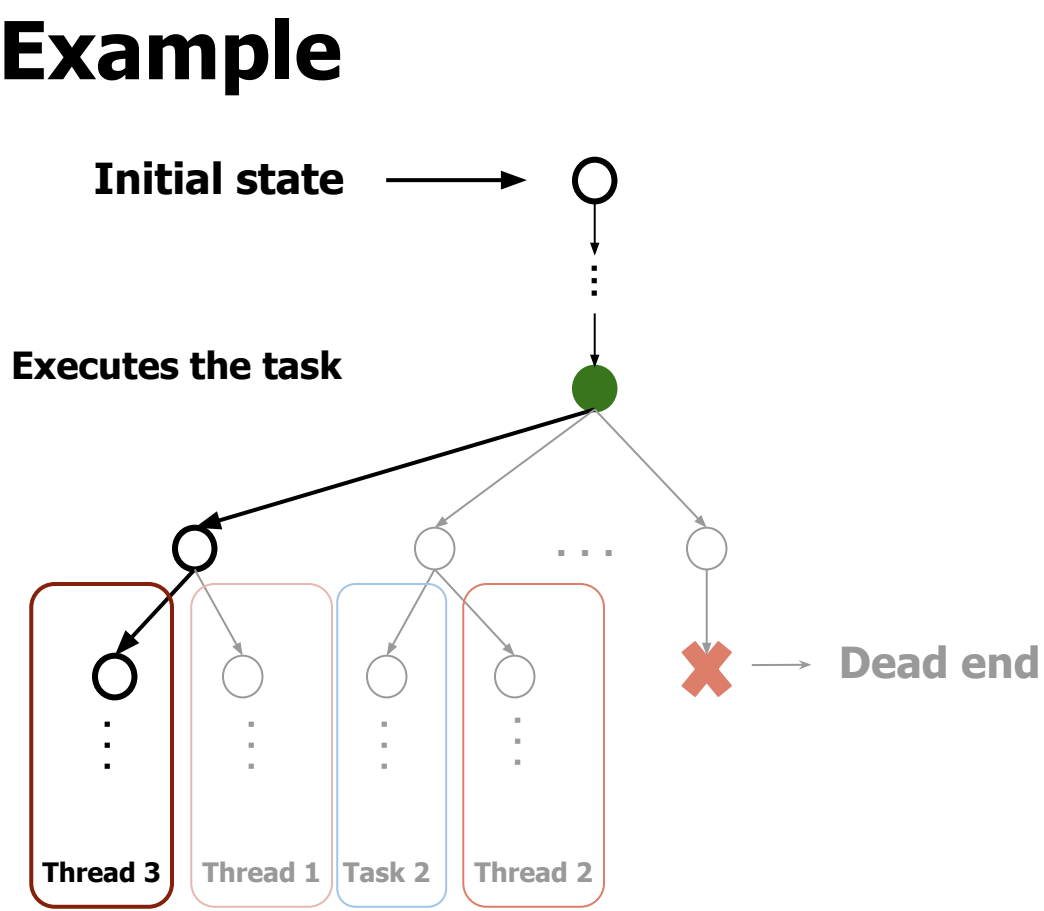

TaskQueue
T2
T1
Initial state
Grabs a task
T1
Dead end
Task 1
Thread 1
Task 2
Thread 2

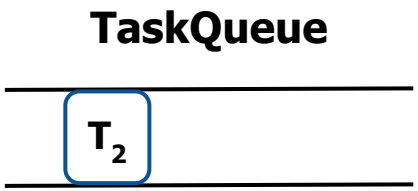
Example


Initial state
Thread 3 moves to the state of Task 1
Task 1
Thread 1
Task 2
Thread 2
Dead end


TaskQueue
T2

Example


Initial state
Executes the task
Thread 3
Thread 1
Task 2
Thread 2
Dead end


TaskQueue
T2

Comments

- The execution **finishes** when **all threads are inactive** and **no task** is **available** on queue**.**
- We use **mutexes**/**locks** to **push**/**pull tasks** into the **queue**.

Comments

- The execution **finishes** when **all threads are inactive** and **no task** is **available** on queue**.**
- We use **mutexes**/**locks** to **push**/**pull tasks** into the **queue**.

Thresholds

1. To avoid **task overflow**, we set a **threshold** into the **number of tasks** that the queue can concurrently hold**.**
2. We also set a **threshold** in **task creation**. If an **active thread** is in a **state with less than 3 remaining taxa**, this thread **cannot create** and push **tasks**

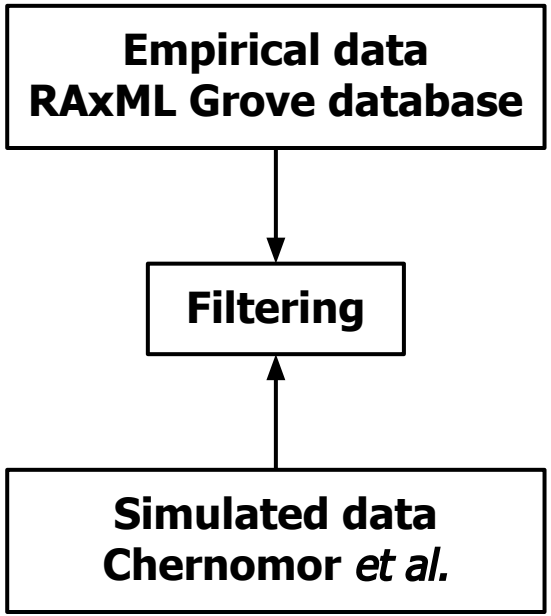
Results

Experimental Setup

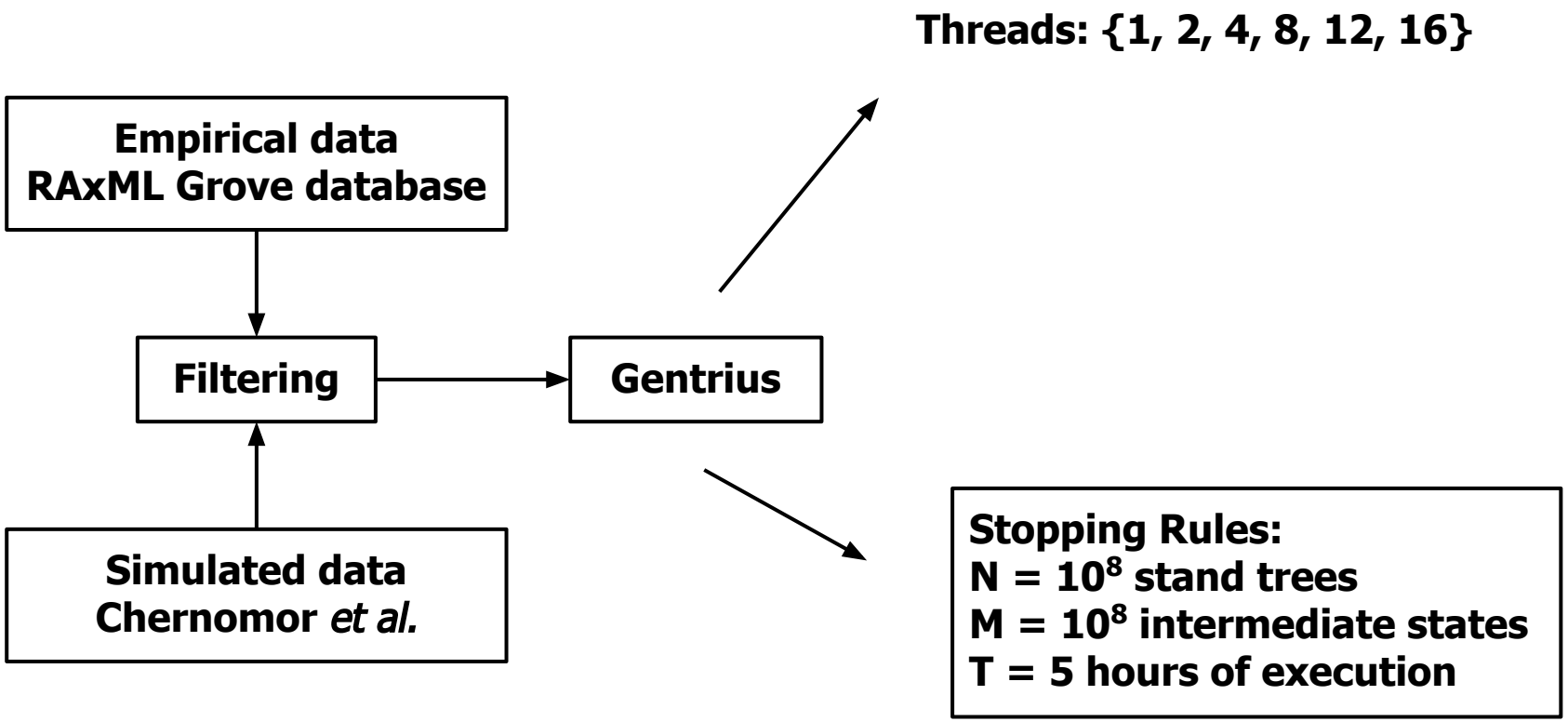
Empirical data
RAxML Grove database

Simulated data
Chernomor *et al.*

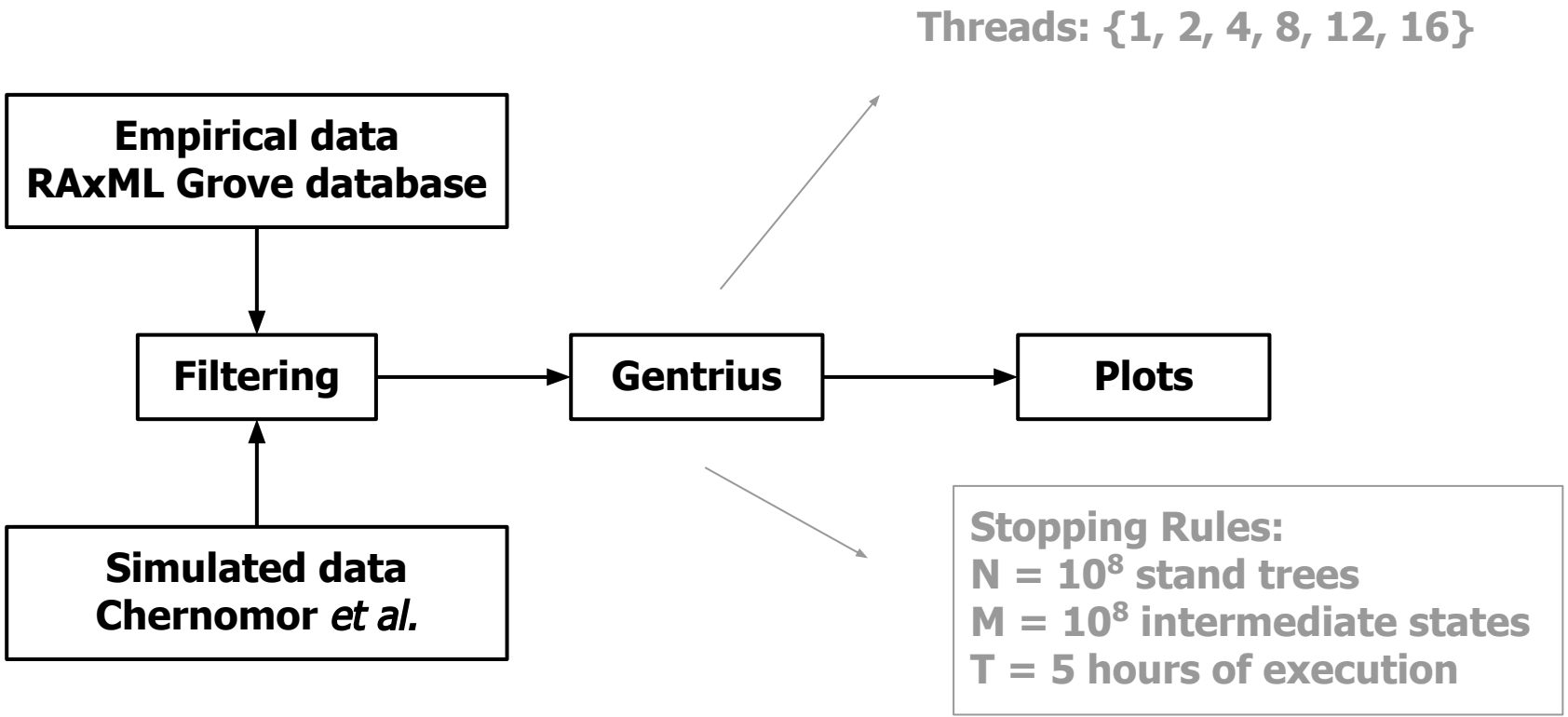
Experimental Setup



Experimental Setup



Experimental Setup



Empirical data

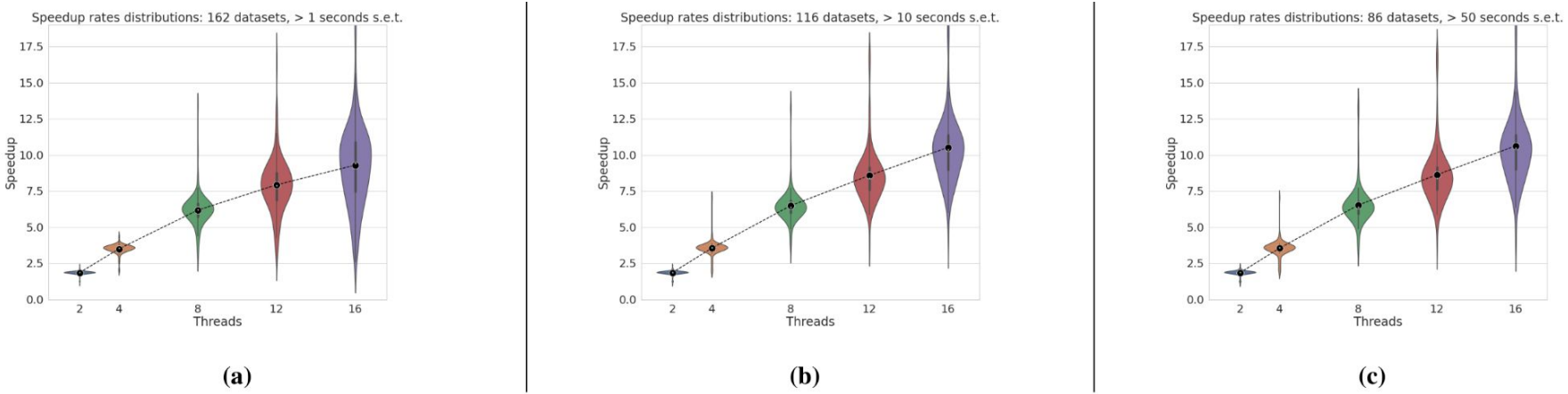


Fig. 7: Per thread speedups on empirical data, the mean speedup values are depicted by a dashed line (s.e.t. stands for 'serial execution time'). Speedups were calculated for: **(a)** 162 datasets with more than 1 second s.e.t. **(b)**, 116 with more than 10 seconds s.e.t. **(c)**, 86 with more than 50 seconds s.e.t.

Simulated data

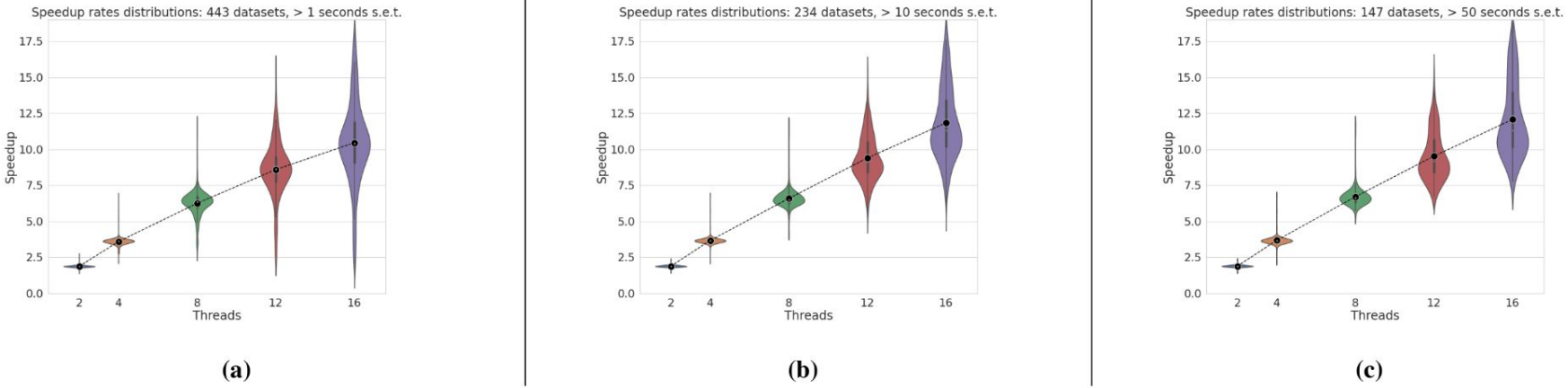
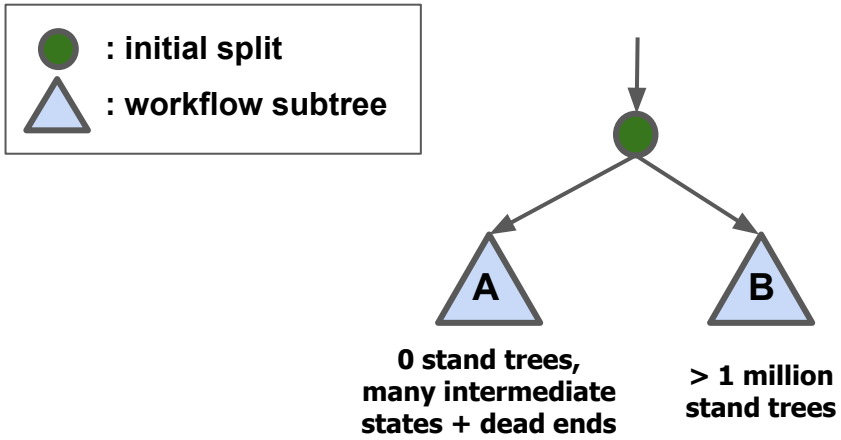


Fig. 6: Per thread speedup distributions on simulated data. The mean speedups are depicted by a dashed line (s.e.t. stands for 'serial execution time'). Speedup rates were calculated for: **(a)** 443 datasets have more than 1 second of s.e.t. **(b)** 234 more than 10 seconds s.e.t. **(c)**, 147 more than 50 seconds s.e.t.

Speedup variances

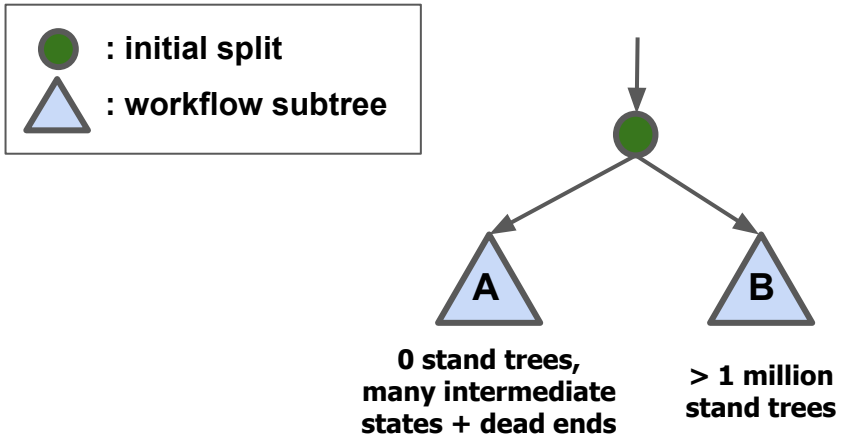
- **The following unbalanced structure** in the workflow tree can either cause **super-linear speedup** or **speedup plateau**.



Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	-	-
Intermediate states	-	-
Time	-	-

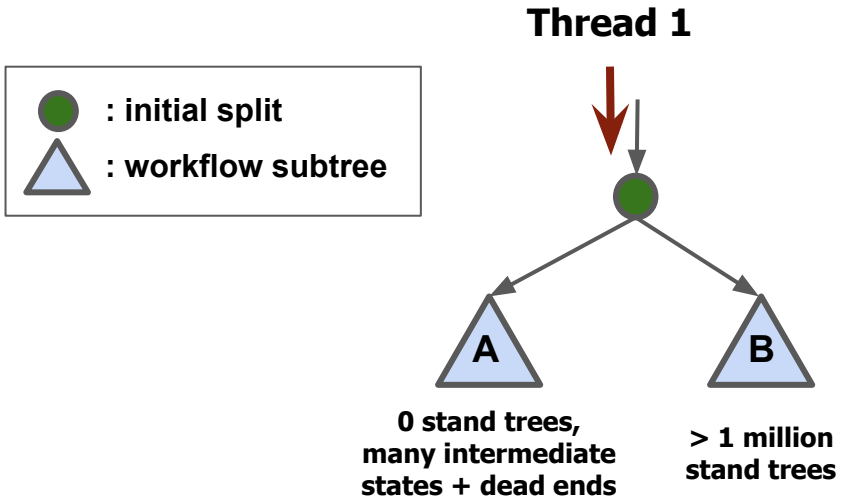


Sequential Execution

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	-	-
Intermediate states	-	-
Time	-	-



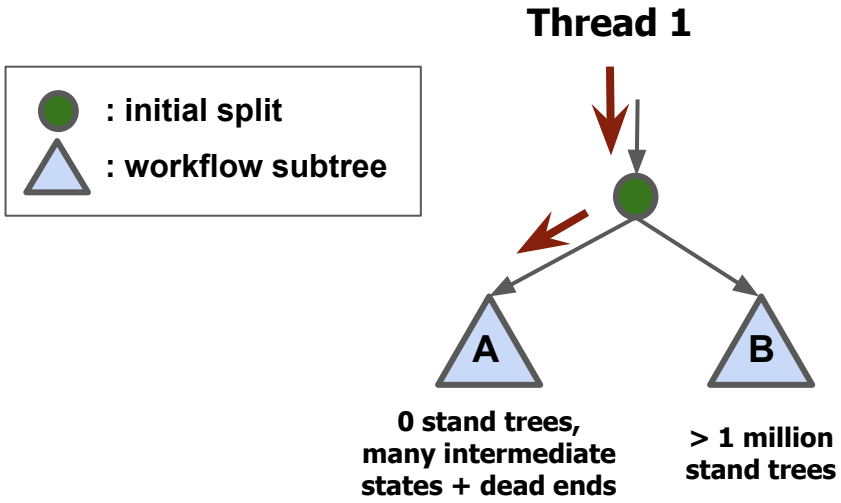
Sequential Execution

Thread 1 reaches the state of initial split

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	-	-
Intermediate states	-	-
Time	-	-



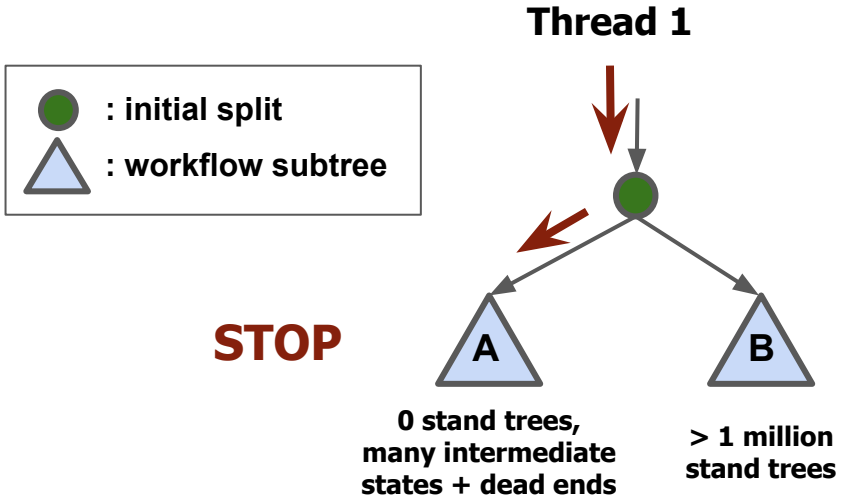
Sequential Execution

Thread 1 enters the left subtree, with many intermediate states and dead ends

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	0	-
Intermediate states	10^7	-
Time	100 s	-



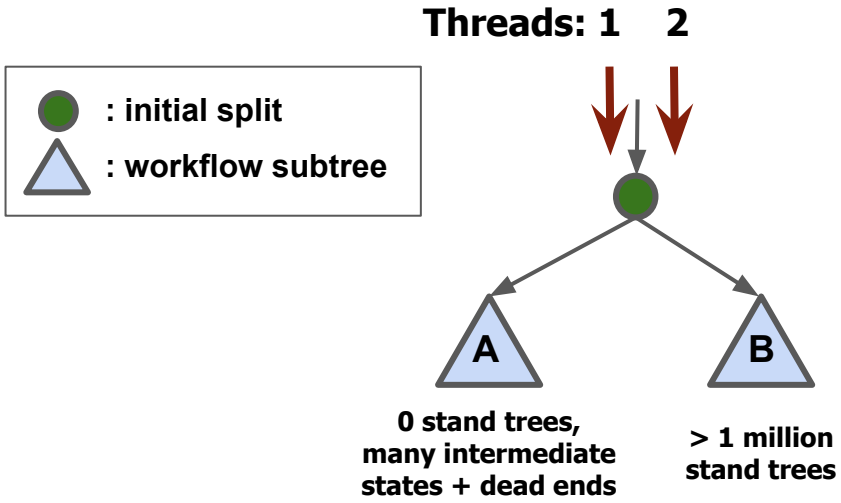
Sequential Execution

Stopping rule for intermediate states is activated. The execution terminates.

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	0	-
Intermediate states	10^7	-
Time	100 s	-



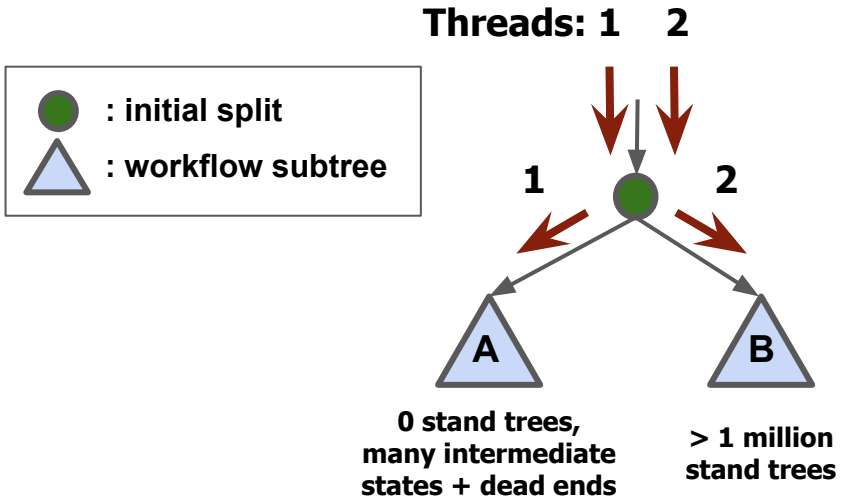
2 Threads

Threads 1 and 2 reach the state of initial split

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	0	-
Intermediate states	10^7	-
Time	100 s	-



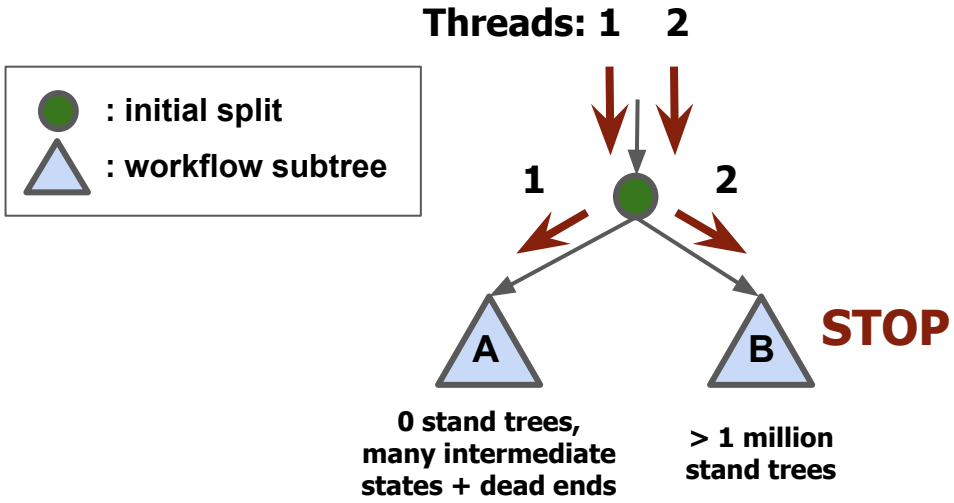
2 Threads

Split:
Thread 1 enters the left subtree with many intermediate states and dead ends
Thread 2 enters the right subtree with 10^6 stand trees

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	0	10^6
Intermediate states	10^7	$< 10^7$
Time	100 s	10 s



2 Threads

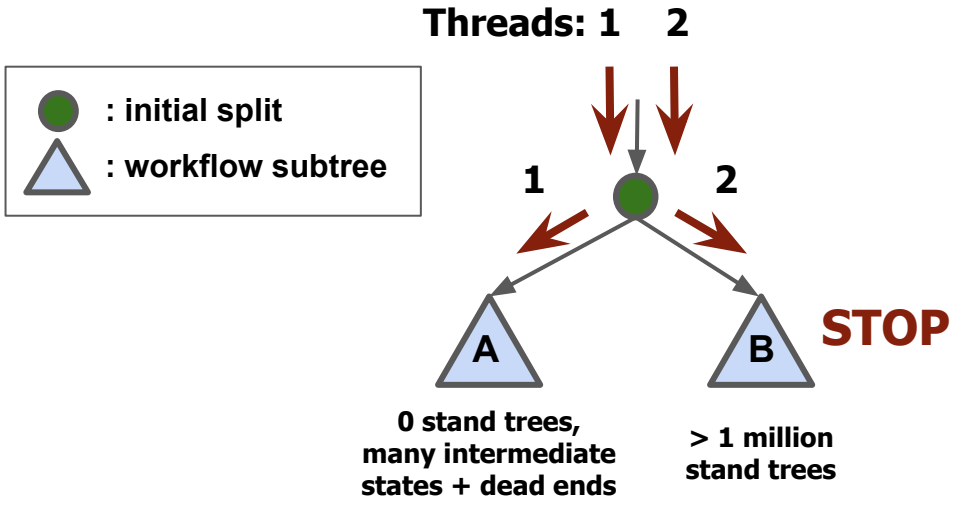
Stopping rule for stand trees is activated (Thread 2). The execution terminates.

Speedup variances

a) **Superliner speedups** (Example)

	1 Thread	2 Threads
Stand trees	0	10^6
Intermediate states	10^7	$< 10^7$
Time	100 s	10 s

Speedup 10x !



2 Threads

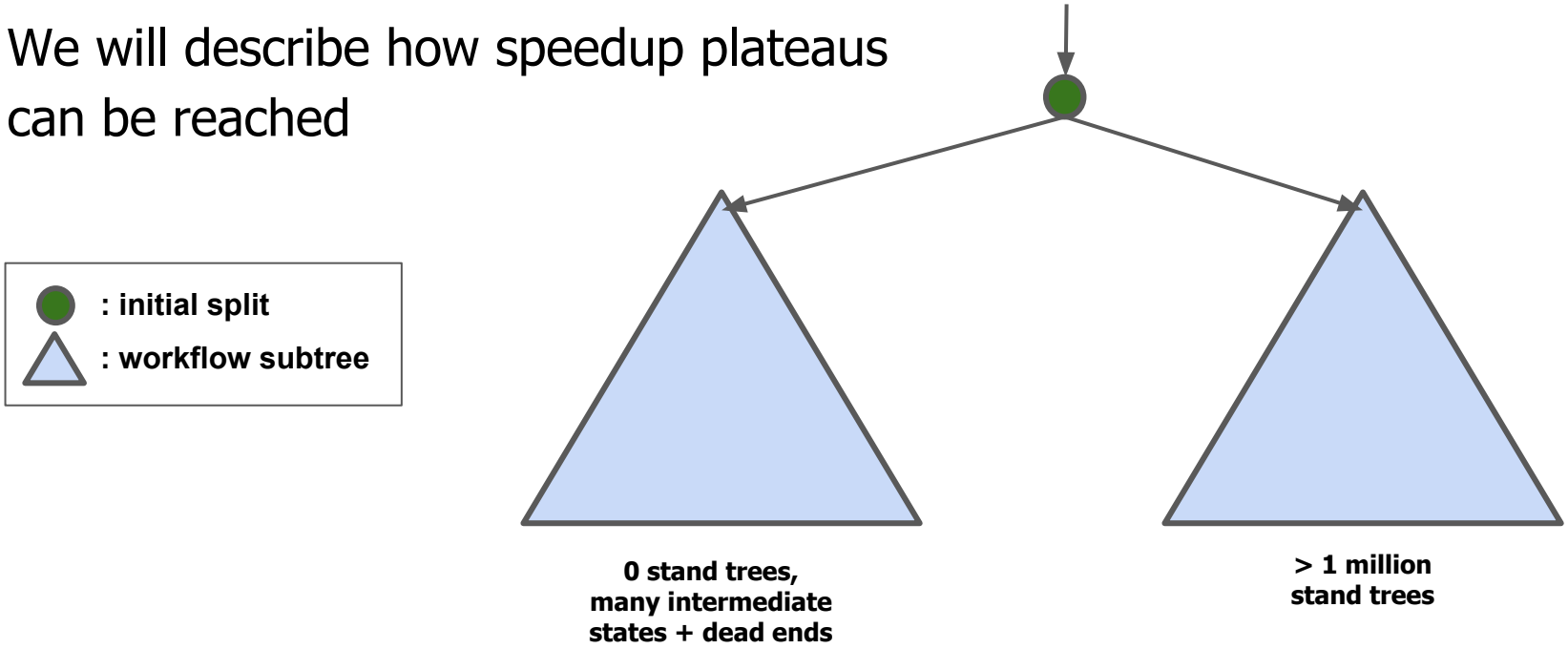
Stopping rule for stand trees is activated (Thread 2). The execution terminates.

Example datasets:
sr_sim-data-44

Speedup variances

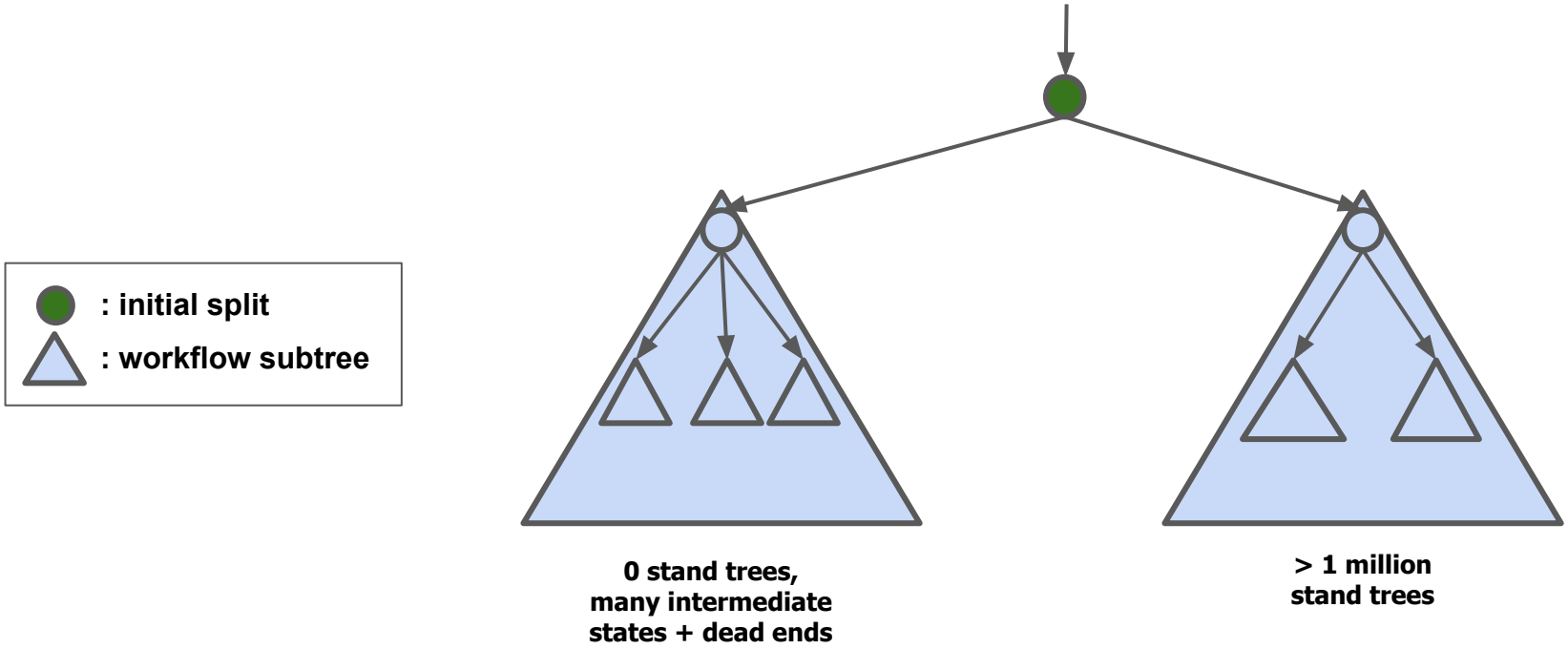
b) Speedup plateaus

We will describe how speedup plateaus can be reached



Speedup variances

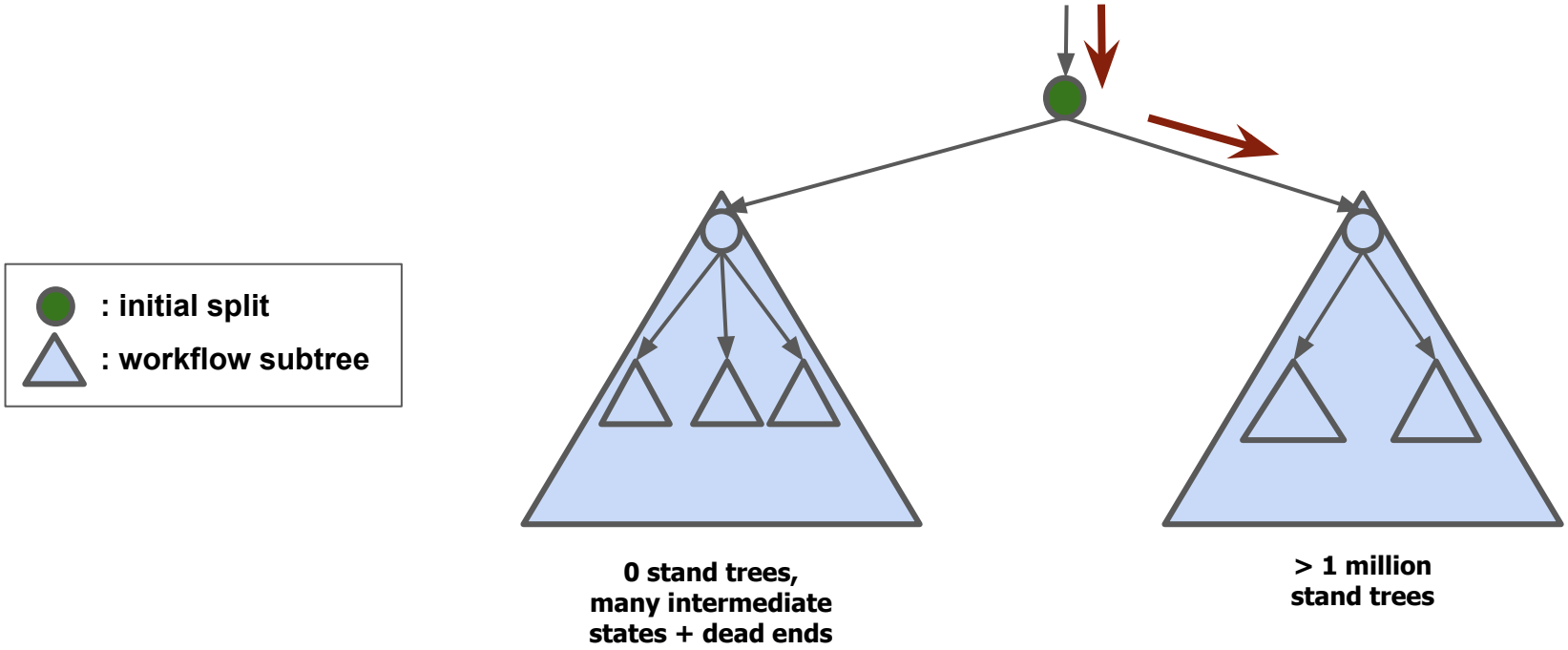
b) Speedup plateaus



Speedup variances

Sequential Execution

b) Speedup plateaus

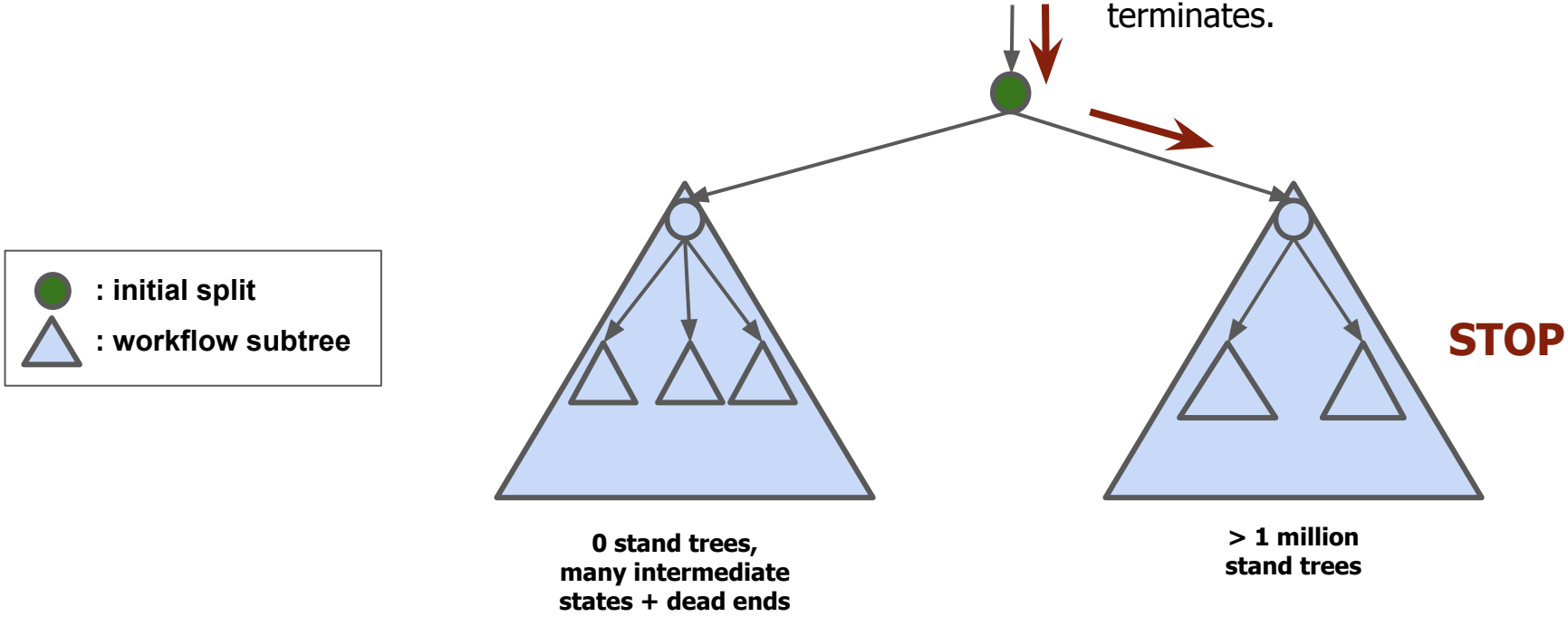


Speedup variances

b) Speedup plateaus

Sequential Execution

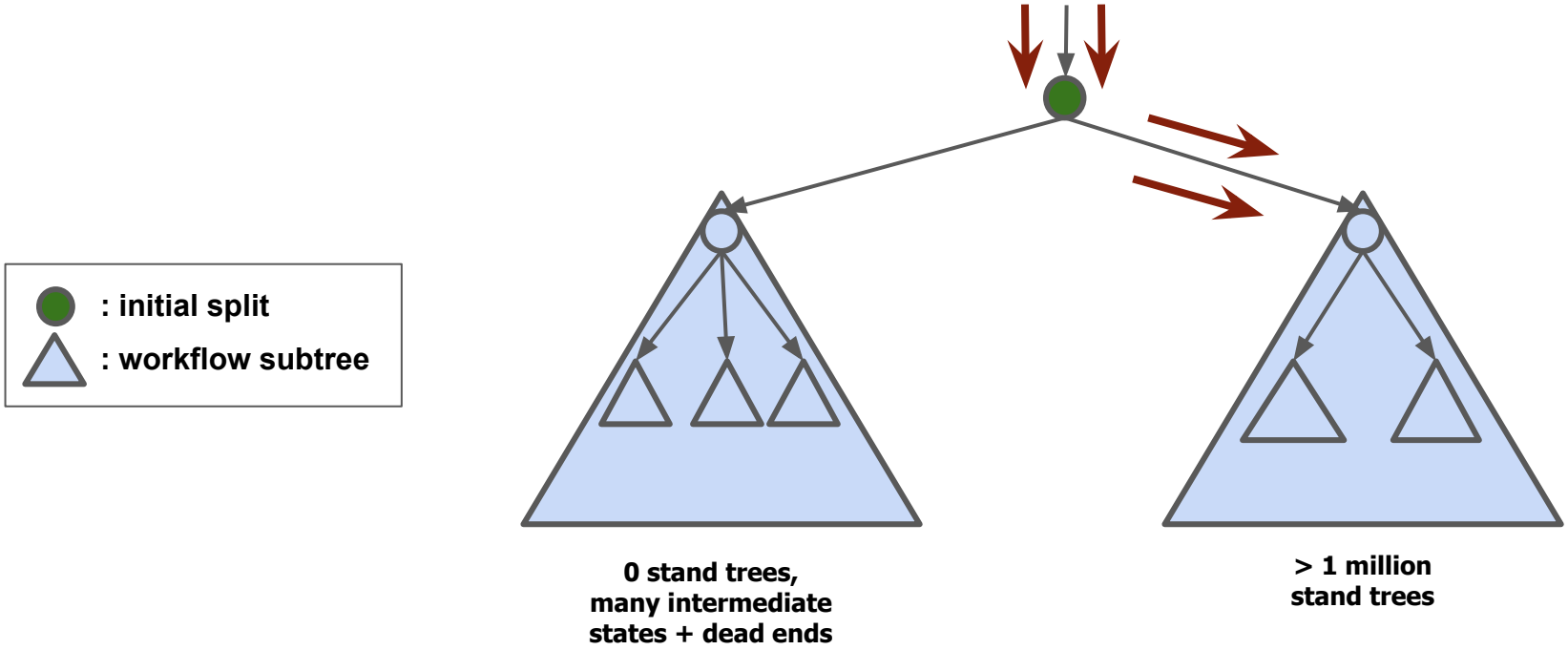
Stopping rule for stand trees is activated. The execution terminates.



Speedup variances

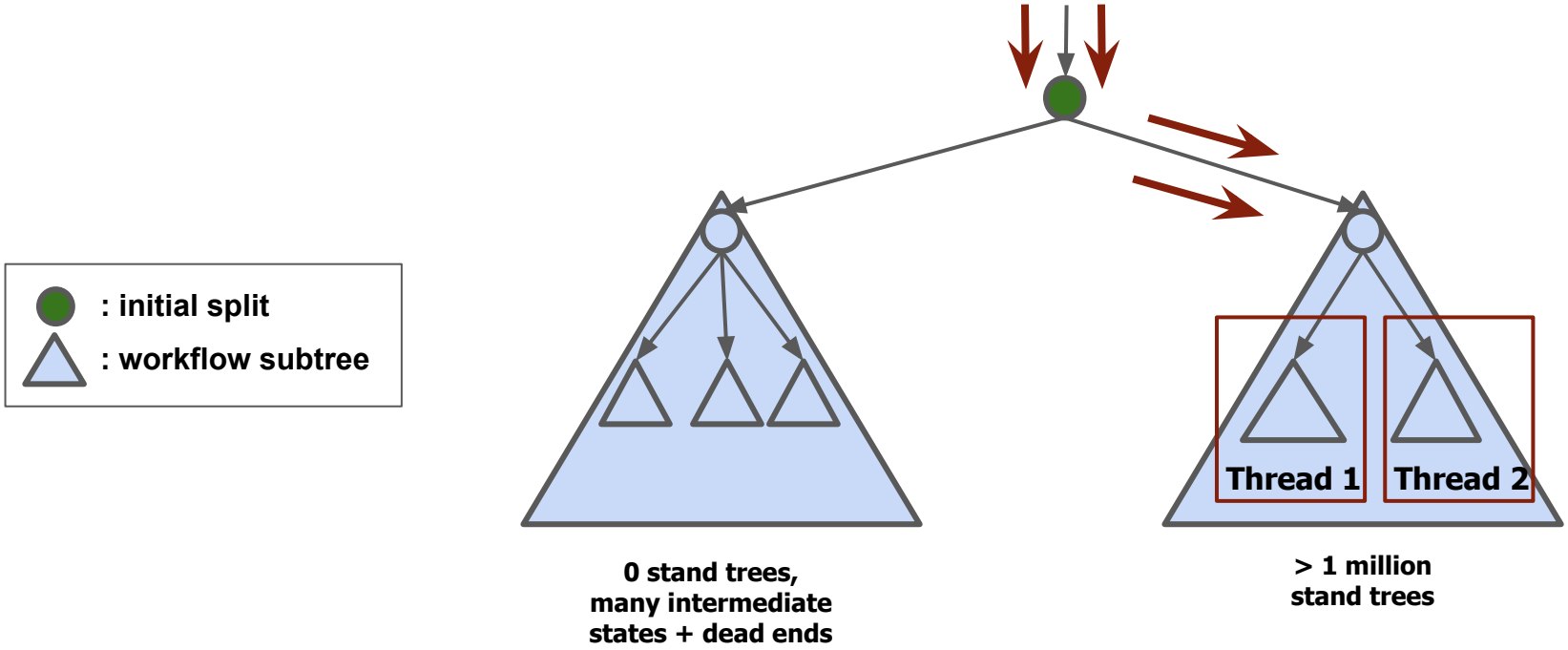
2 threads

b) Speedup plateaus



Speedup variances

b) Speedup plateaus

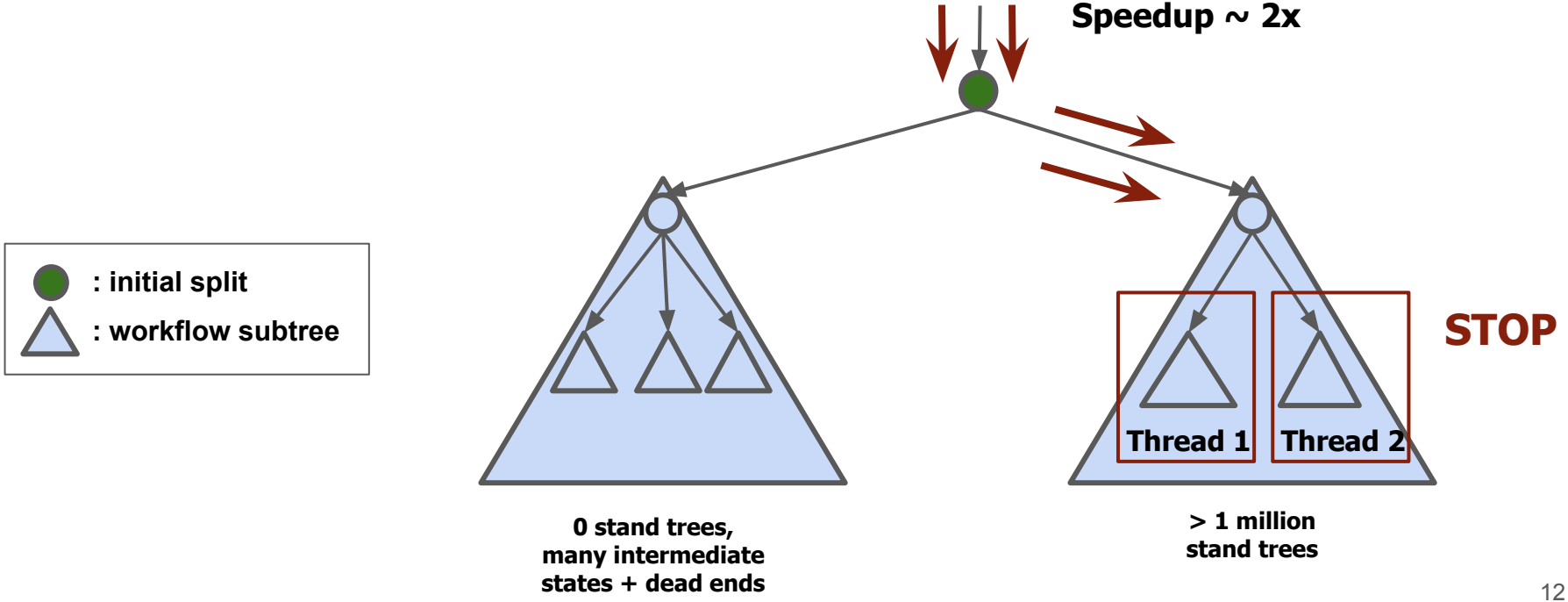


Speedup variances

b) Speedup plateaus

2 threads

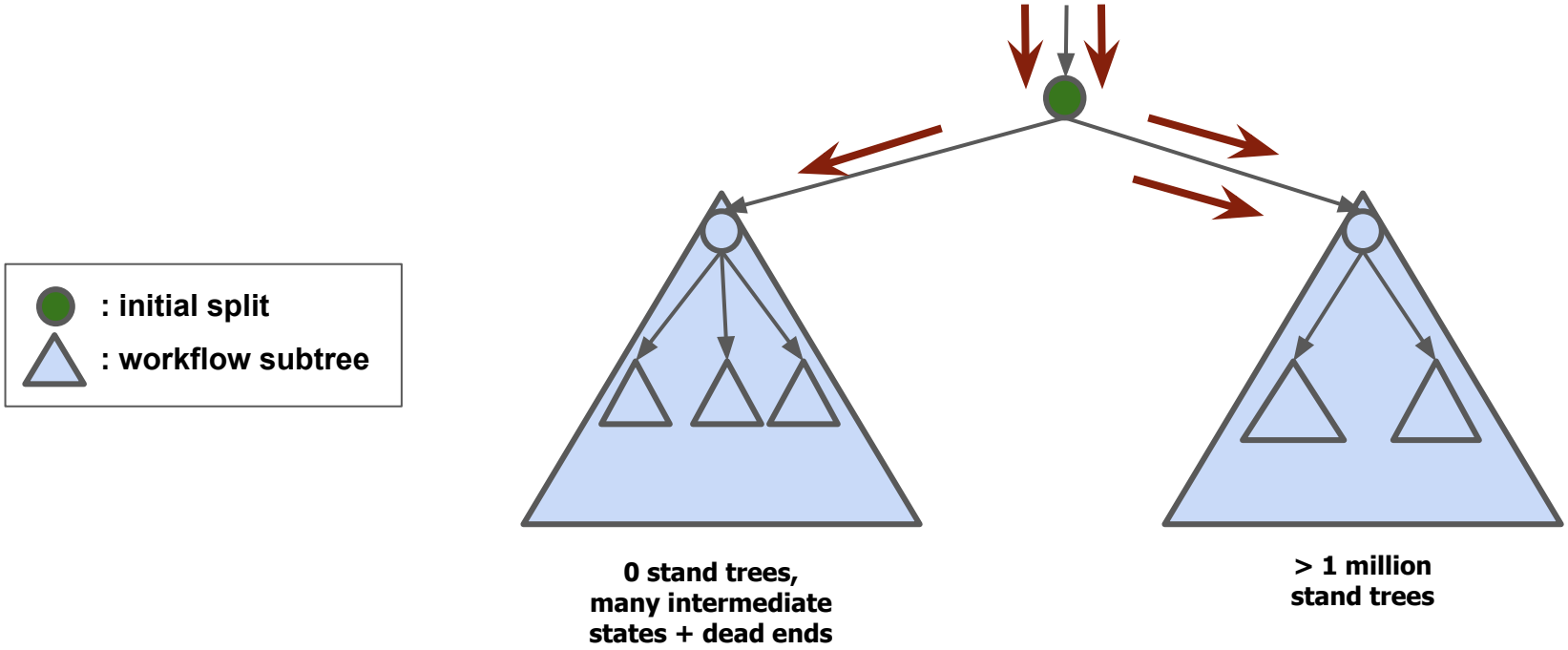
Stopping rule for stand trees is activated.
Speedup ~ 2x



Speedup variances

3 threads

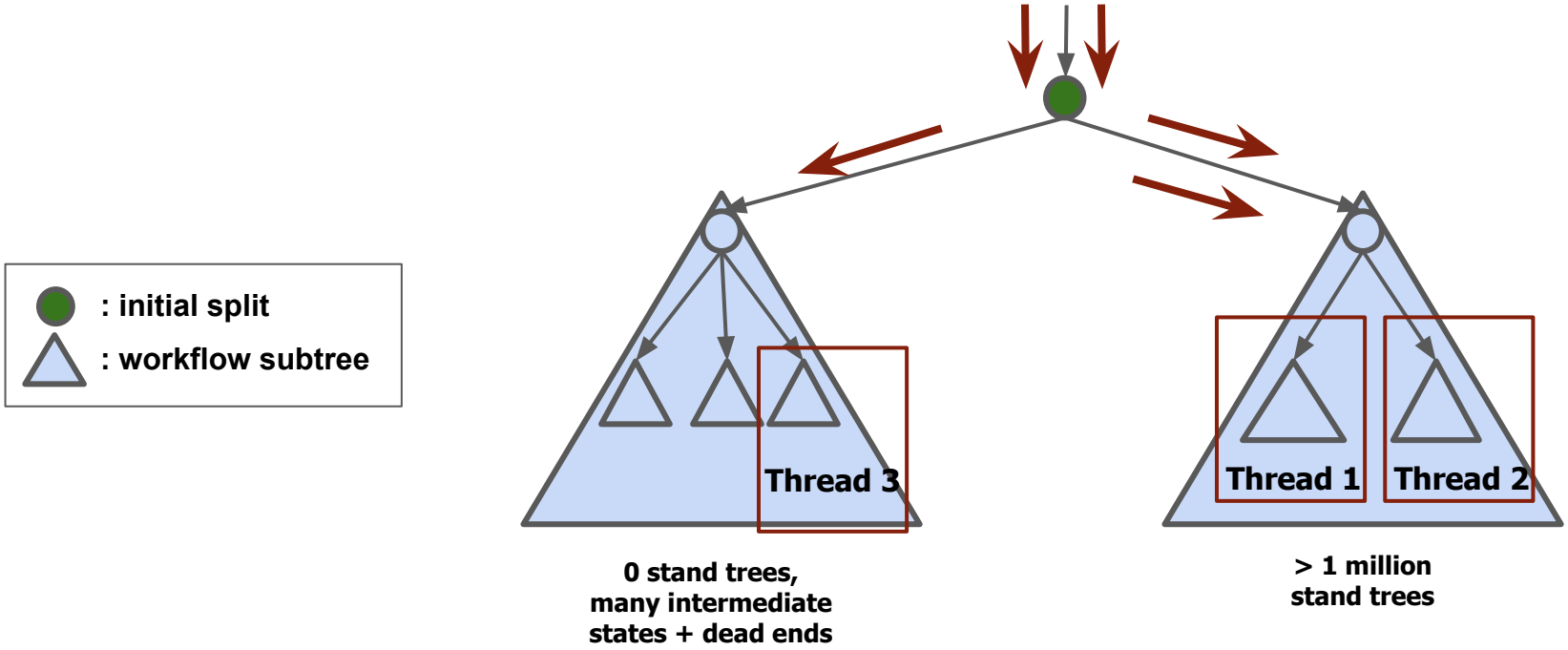
b) Speedup plateaus



Speedup variances

b) Speedup plateaus

3 threads

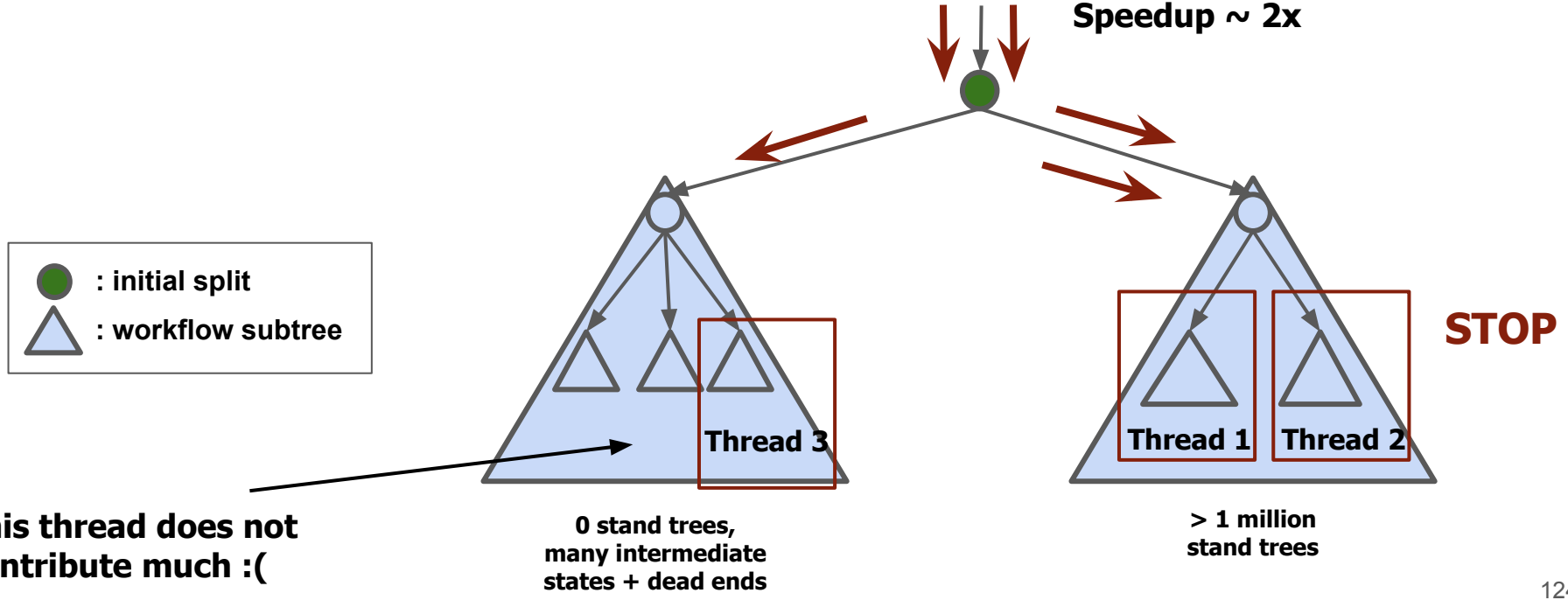


Speedup variances

b) Speedup plateaus

3 threads

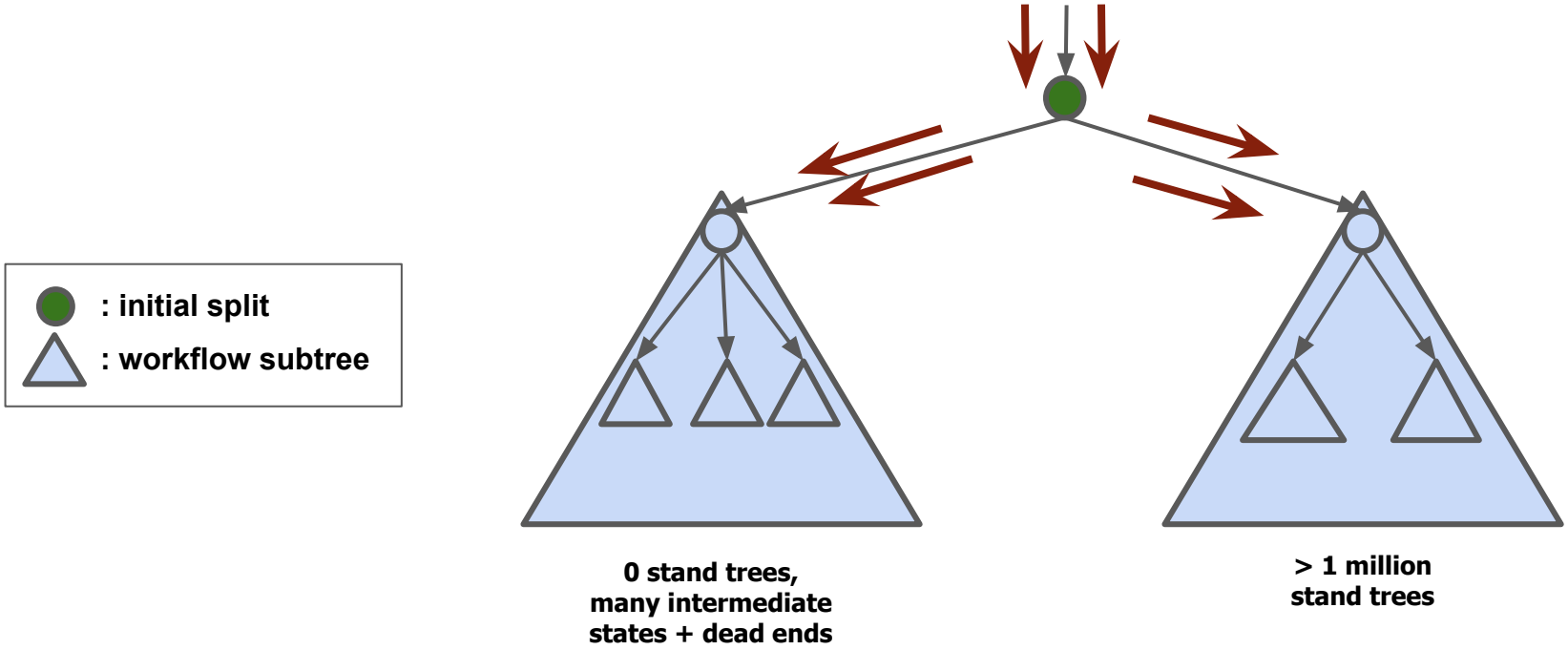
Stopping rule for stand trees is activated.
Speedup ~ 2x



Speedup variances

4 threads

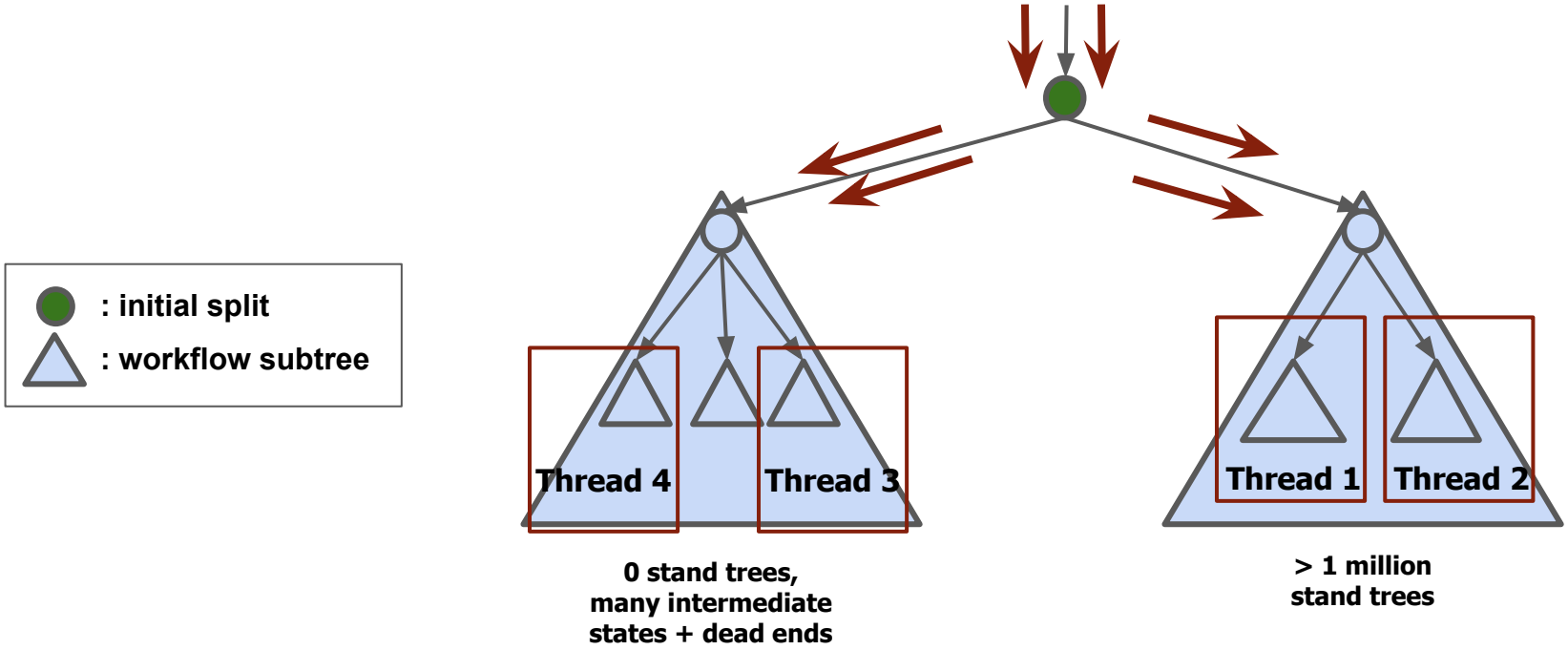
b) Speedup plateaus



Speedup variances

b) Speedup plateaus

4 threads



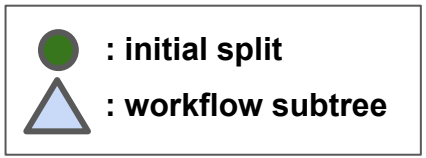
Speedup variances

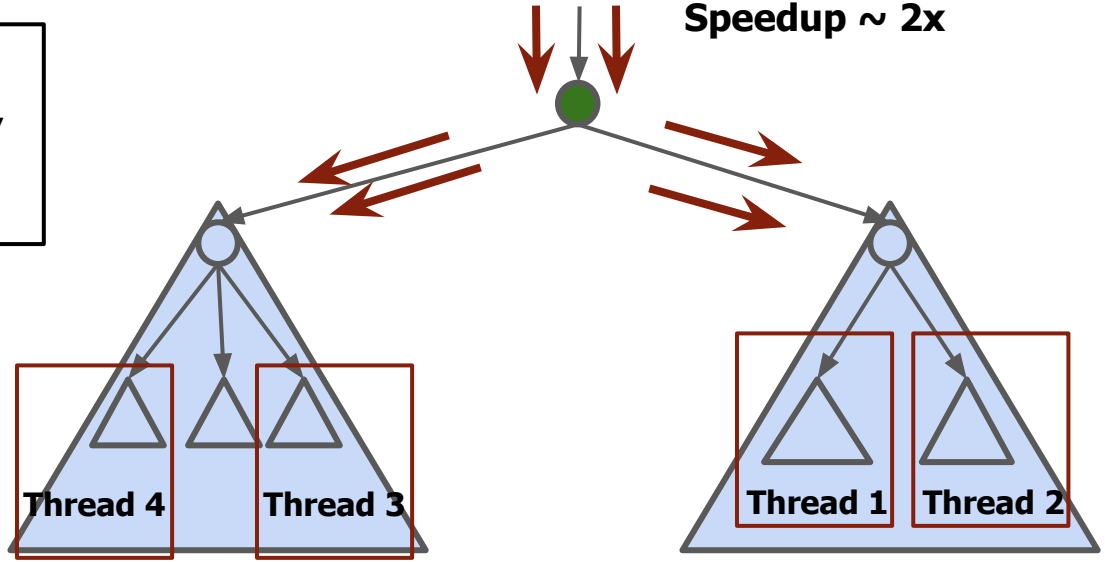
b) Speedup plateaus

Example datasets:
sim-data-1511 , sim-data-1792, sim-data-1795

4 threads

Stopping rule for stand trees is activated.
Speedup ~ 2x





These two threads do not contribute much :(

Conclusion

Summary

- **Gentrius** enumerates **all trees** on **stand**
- The size of stand is important, because it quantifies the uncertainty of phylogenetic inference
- We designed, implemented and tested a parallelization scheme for Gentrius algorithm
- To avoid difficulties, we used a **thread pooling** scheme
- We achieve **linear speedups** up to **16 threads**
- In some **rare cases**, we achieve **super-linear** speedups

Availability

- Parallel Gentrius is available under GNU GPL:
  - https://www.github.com/togkousa/iqtree2/tree/terragen

- Manuscript *(to be published)*:
  - Togkousidis, A., Chernomor, O., Stamatakis. A., (2023). Parallel Inference of Phylogenetic Stands with Gentrius. *IEEE Proceedings*

Thank you