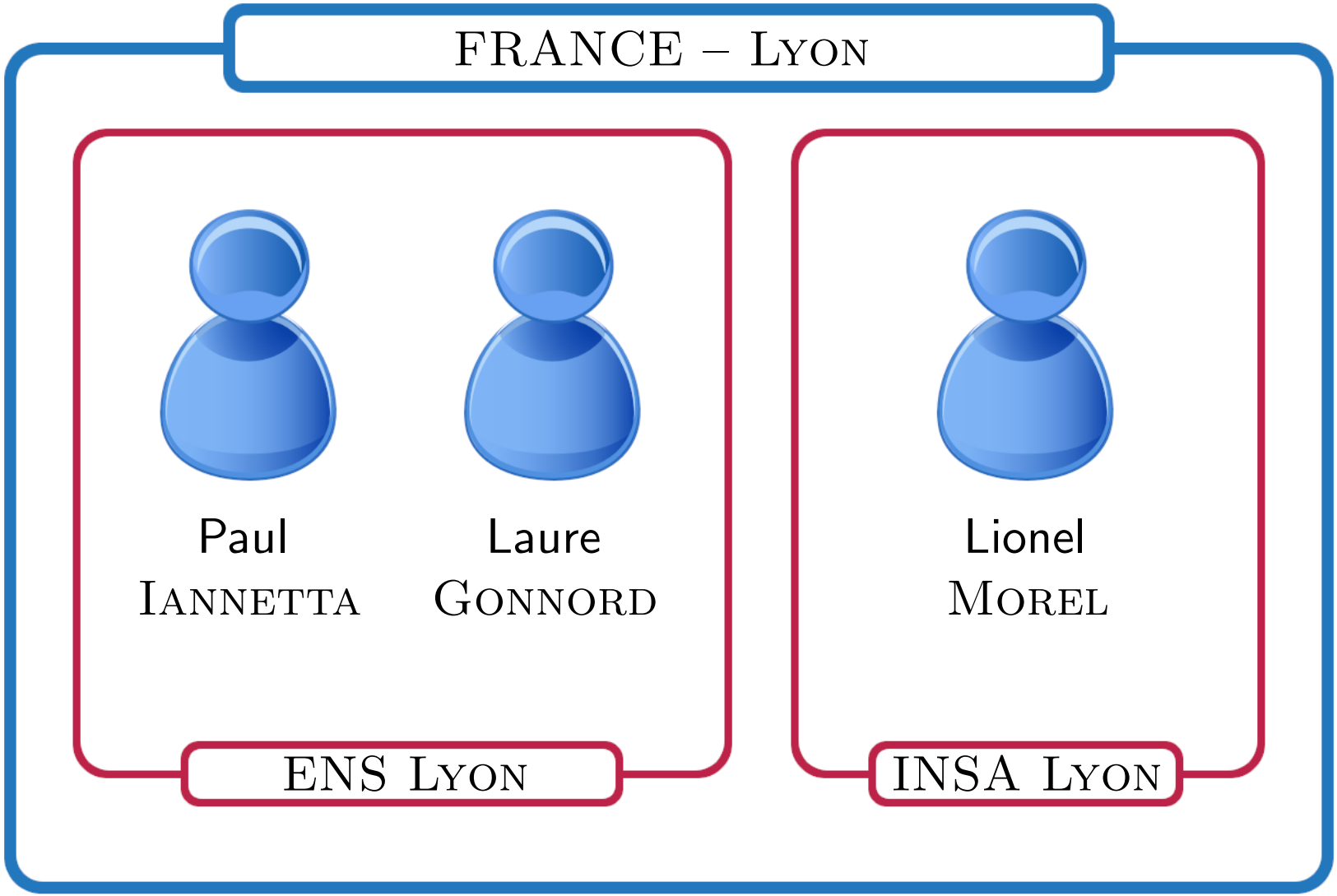
A New Memory Layout for Self-Rebalancing Trees

. . . or fitting trees into arrays

Paul Iannetta
paul.iannetta@ens-lyon.fr

Mar 2, 2021


FRANCE – LYON
Paul
IANNETTA
Laure
GONNORD
ENS LYON
Lionel
MOREL
INSA LYON

Why Trees?

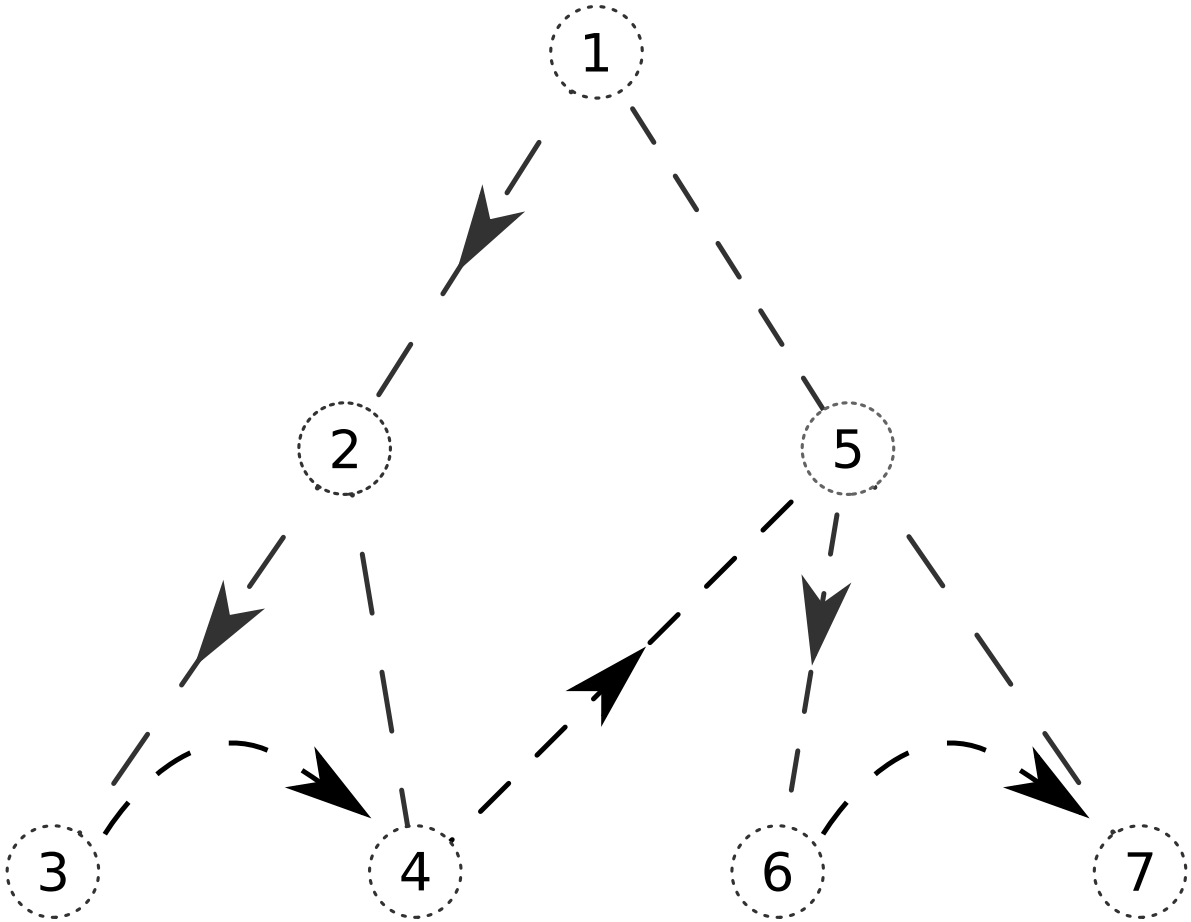
- Balanced trees are everywhere
  - Look-up tables
  - Linux kernel's scheduler
  - Type checkers

- Still needs improvements
  - Not cache-friendly
  - No parallelism

How to Store Trees into Arrays?

- Two natural ways to store trees into arrays:
  - Depth-first
  - Breadth-first

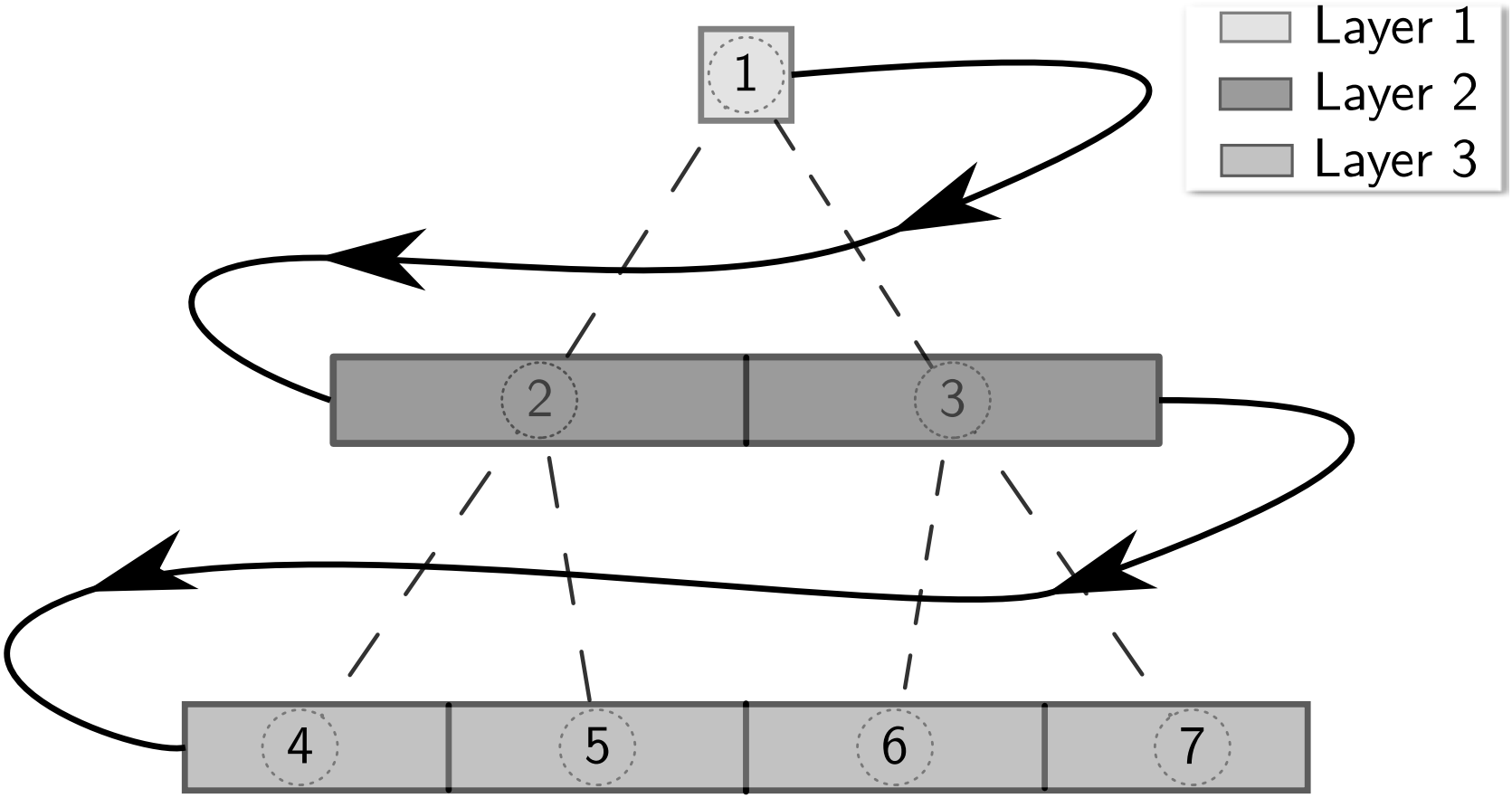
Depth-first Layout



df-array:

1	2	3	4	5	6	7

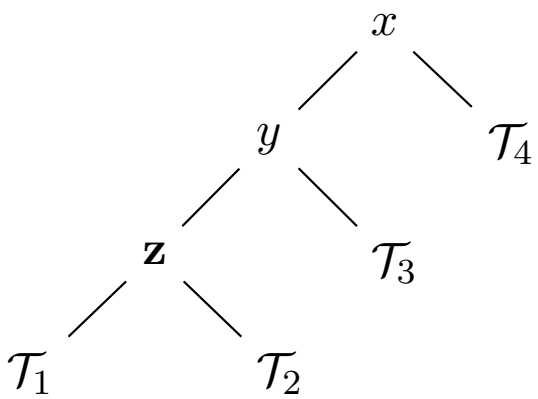
Breadth-first Layout



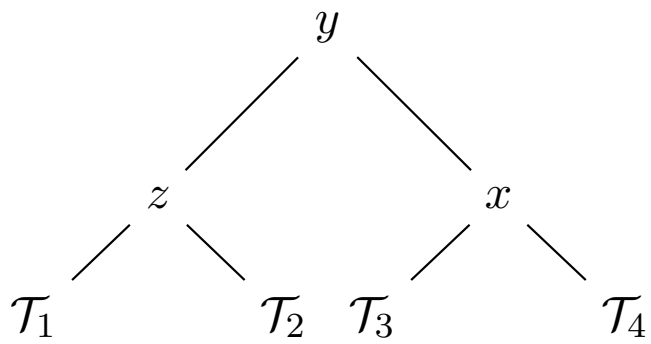
bf-array:

1	2	3	4	5	6	7

What does a Rotation do?

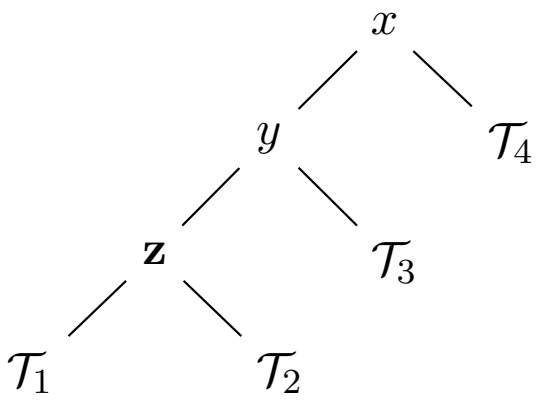


(a) A binary tree T

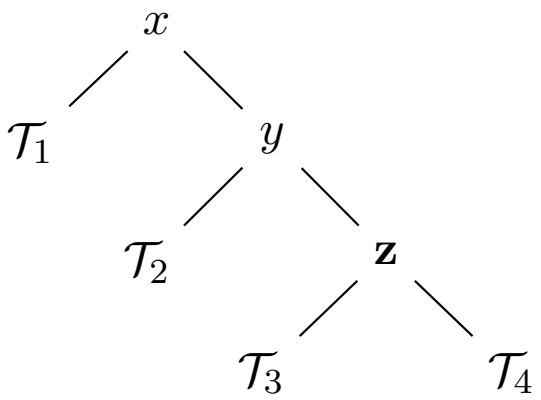


(b) T after a right rotation

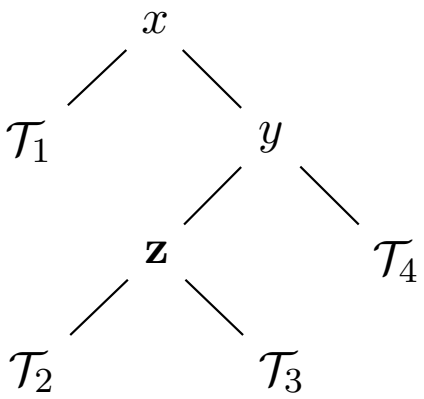
When is a rotation triggered?



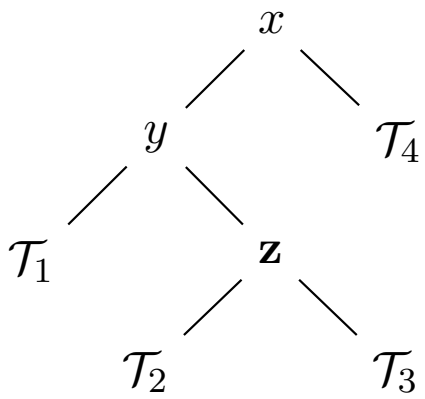
(a) Needs a L-rotation



(b) Needs a R-rotation

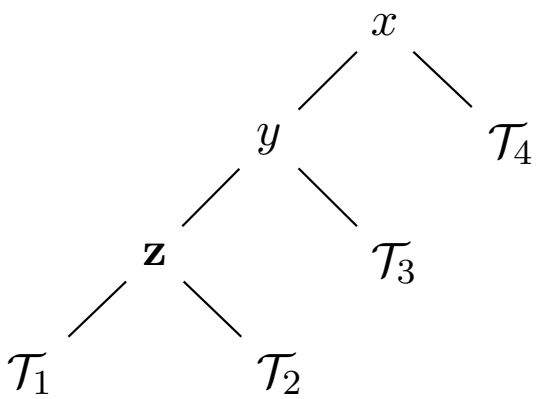


(c) Needs a LR-rotation

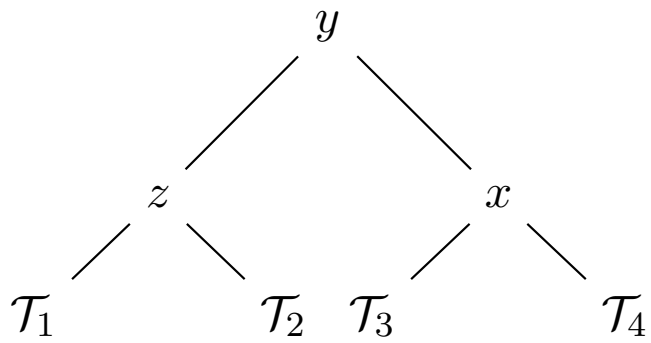


(d) Needs a RL-rotation

Decomposition of the Right Rotation into Array Operations

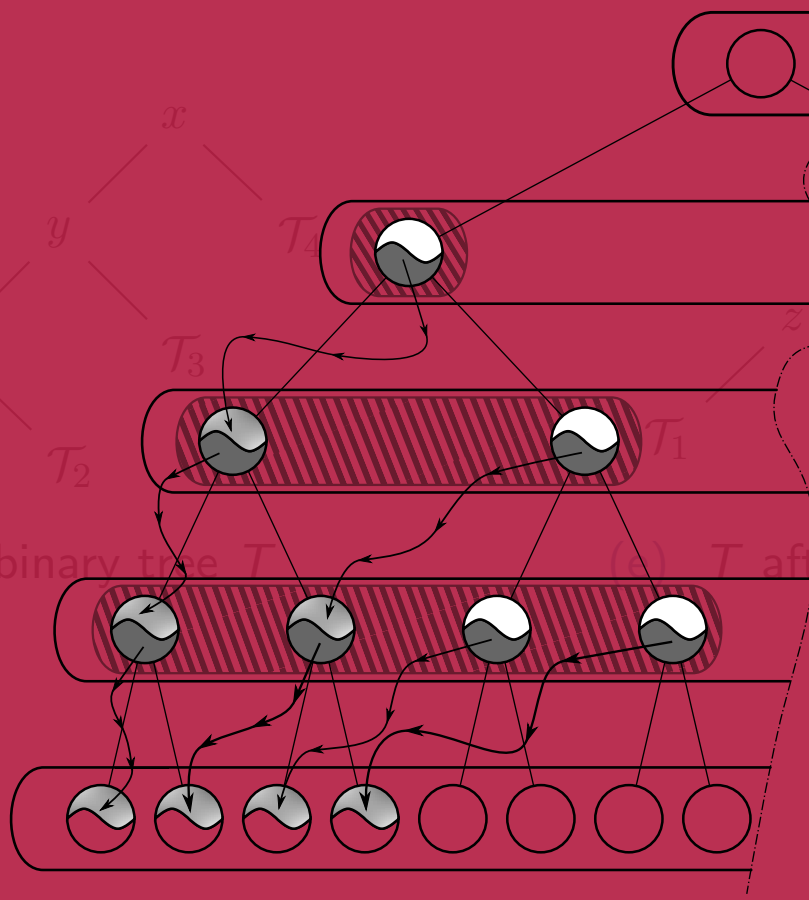


(a) A binary tree T

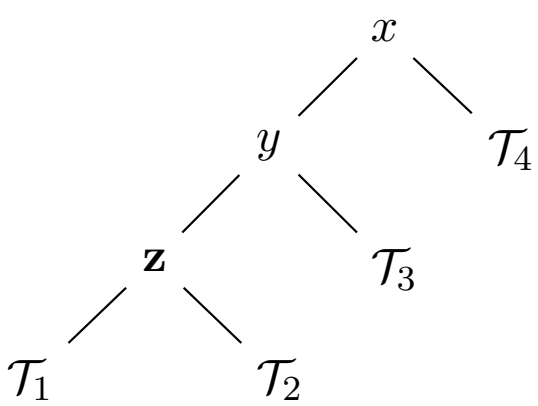


(e) T after a right rotation

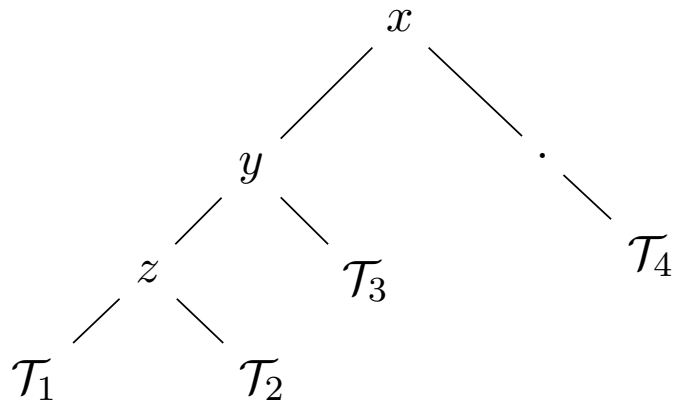
Decomposition of the Right Rotation into Array Operations



Decomposition of the Right Rotation into Array Operations

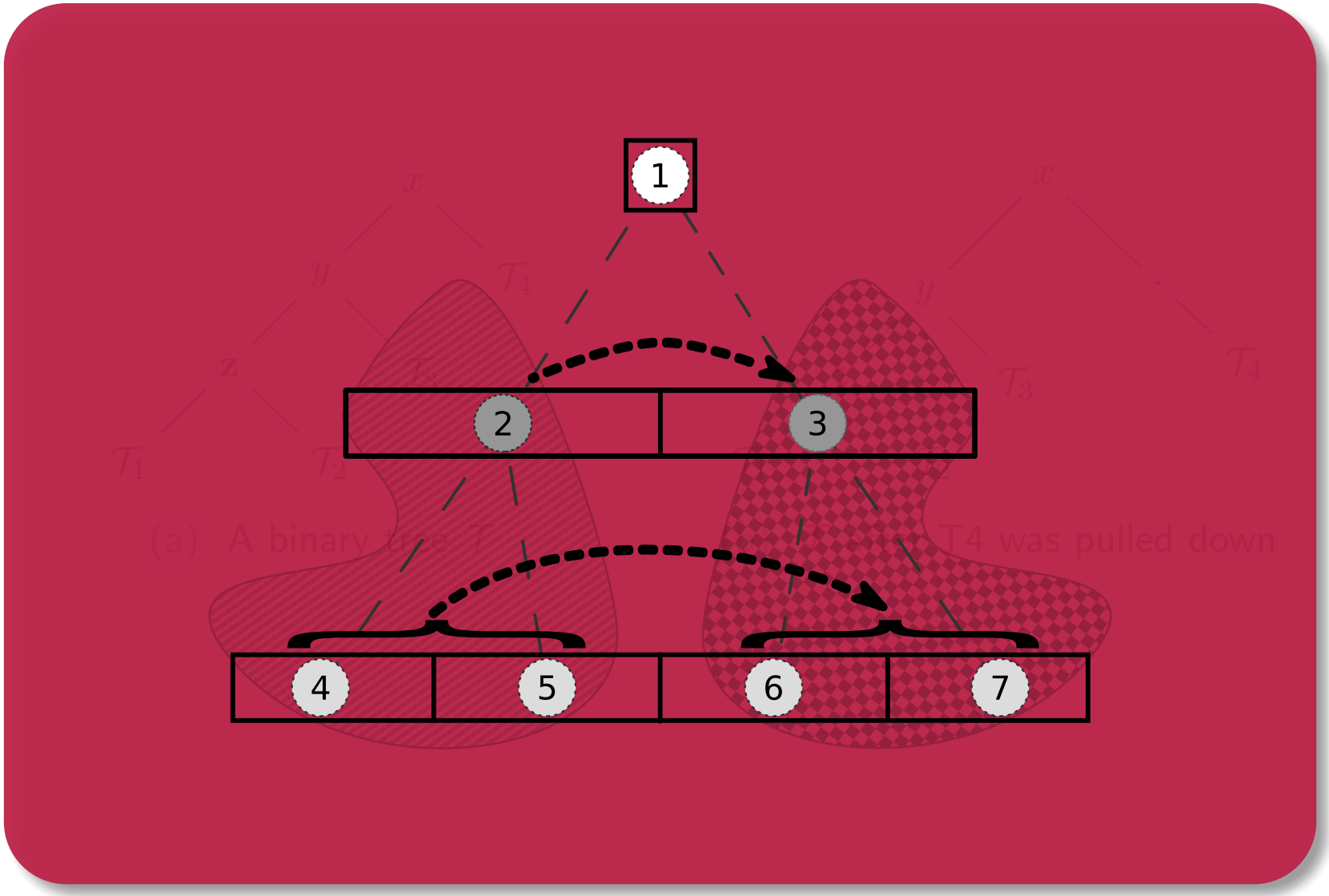


(a) A binary tree T

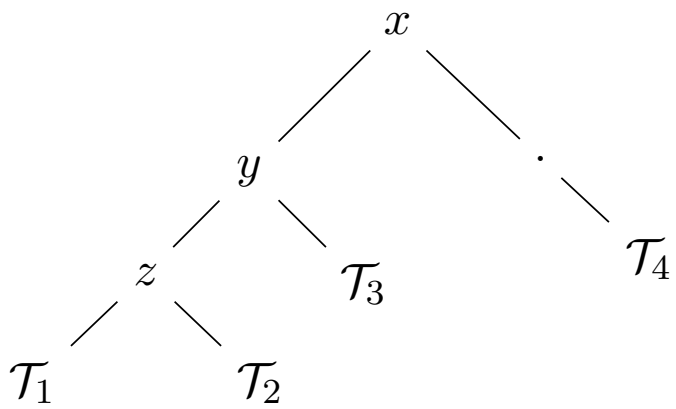


(b) T after T4 was pulled down

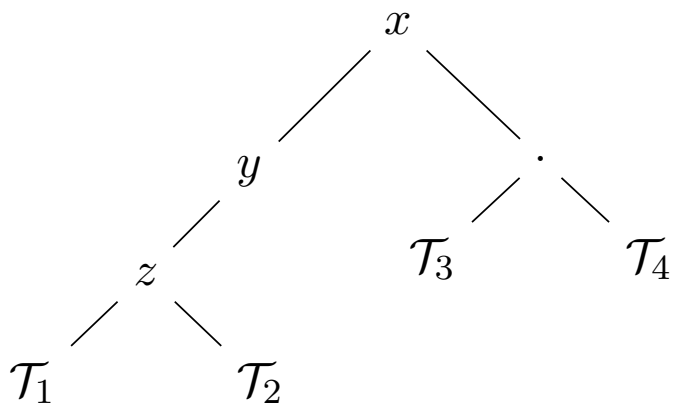
Decomposition of the Right Rotation into Array Operations



Decomposition of the Right Rotation into Array Operations

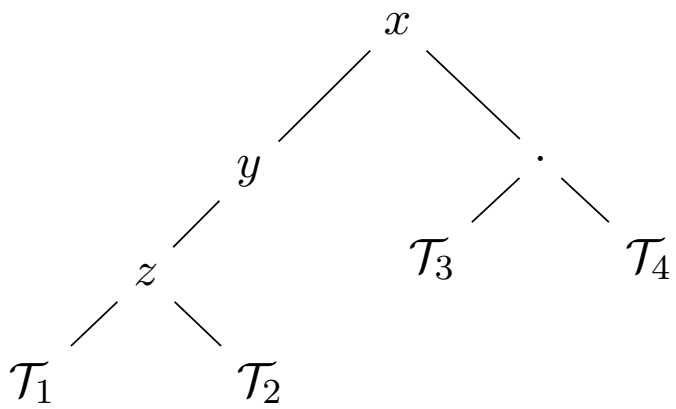


(b) T after T4 was pulled down

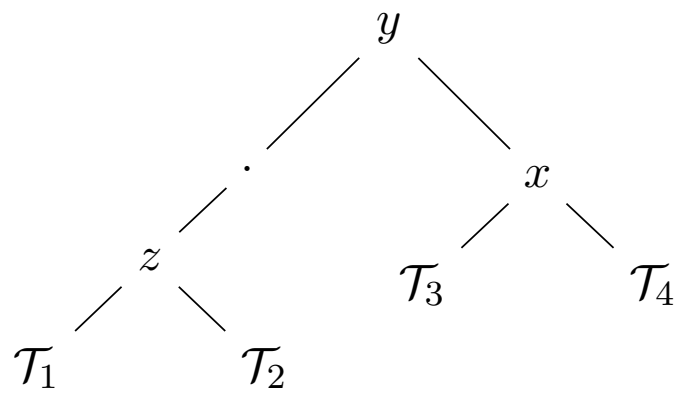


(c) ... and T3 shifted

Decomposition of the Right Rotation into Array Operations

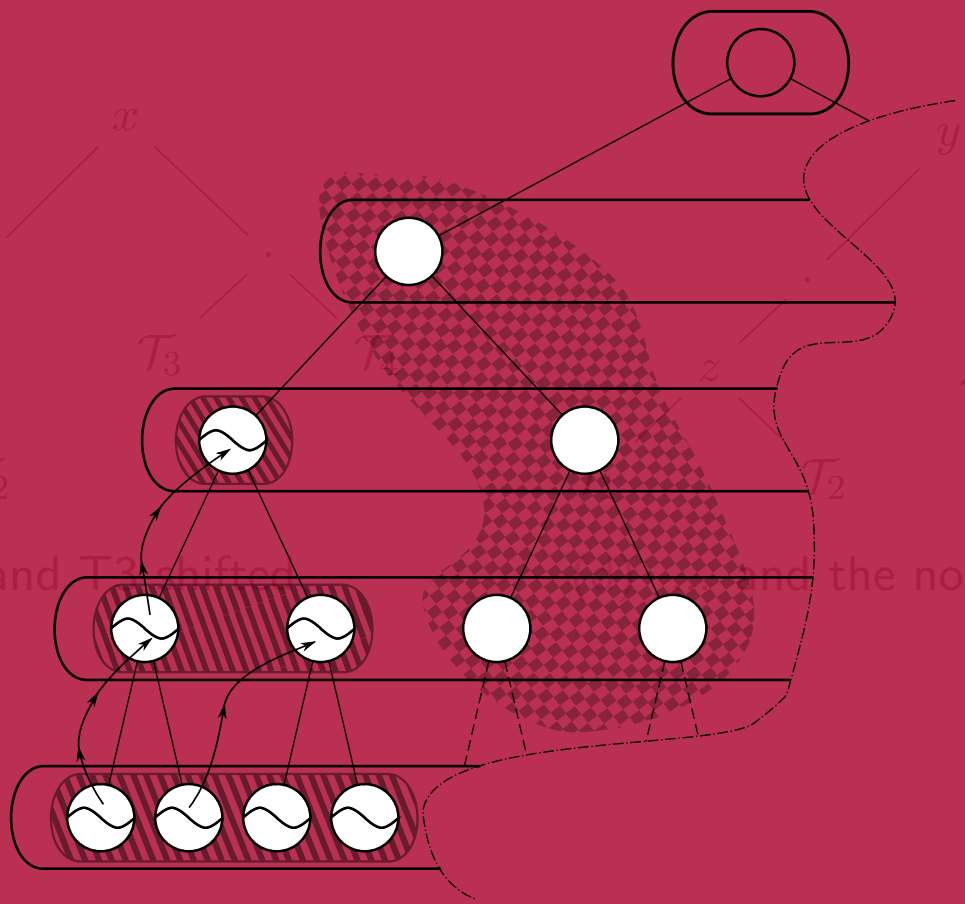


(c) ... and T3 shifted

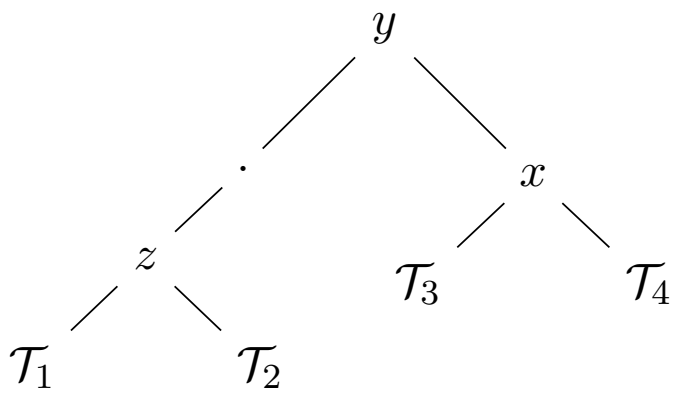


(d) ... and the nodes relabeled

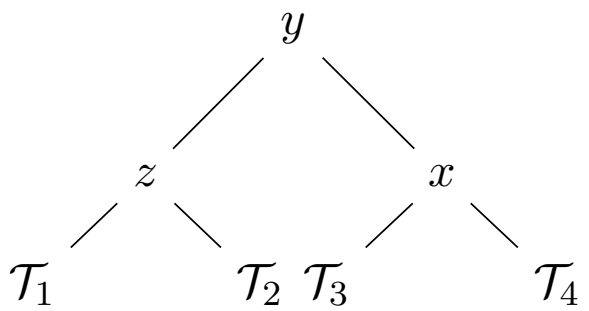
Decomposition of the Right Rotation into Array Operations



Decomposition of the Right Rotation into Array Operations



(d) ... and the nodes relabeled



(e) ... and z pulled up

Benefits of the decomposition

All the rotations can be decomposed into basic operations:

	Right	Left	Right-left	Left-right
1.	pull down $\mathcal{T}_4$	pull down $\mathcal{T}_1$	pull down $\mathcal{T}_1$	pull down $\mathcal{T}_4$
2.	shift right $\mathcal{T}_3$	shift left $\mathcal{T}_2$	shift left $\mathcal{T}_2$	shift right $\mathcal{T}_3$
3.	pull up z	pull up z	pull up $\mathcal{T}_2$	pull up $\mathcal{T}_2$
4.	relabel x, y, z	relabel x, y, z	relabel x, y, z	relabel x, y, z

Benchmarks

		avl-tree		avl-bf	
size	d (%)	t (ms)	misses (%)	t (ms)	misses (%)
64	55	1.8	45	1.7	41
512	33	1.9	48	1.8	43
1024	33	1.7	48	1.9	43
65536	13	31.2	17	47.7	36
524288	17	517.1	46	648.1	60
1048576	13	1192.3	47	1509.4	63
2097152	06	3027.4	47	3779.8	63

Future directions

- Benchmark the model where the operations use parallel code
- Study the effects of compression
- Find ways to leverage more compiler optimizations

Conclusion

- A new memory layout
- Parallelisaton opportunities
- `https://gitlab.inria.fr/paiannet/calv`