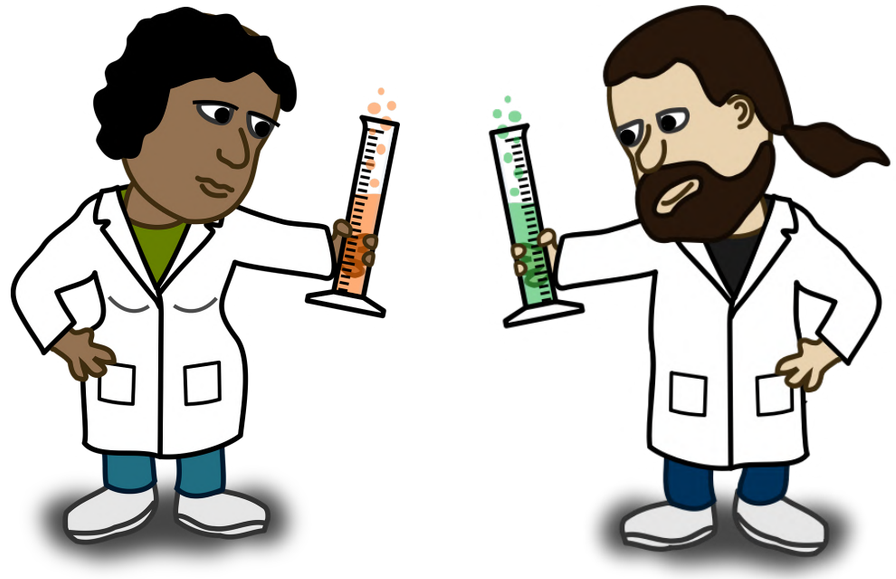
The Computational Stumbling Blocks

Fault-Tolerance and Reproducibility · 20th of May 2023

Lukas Hübner



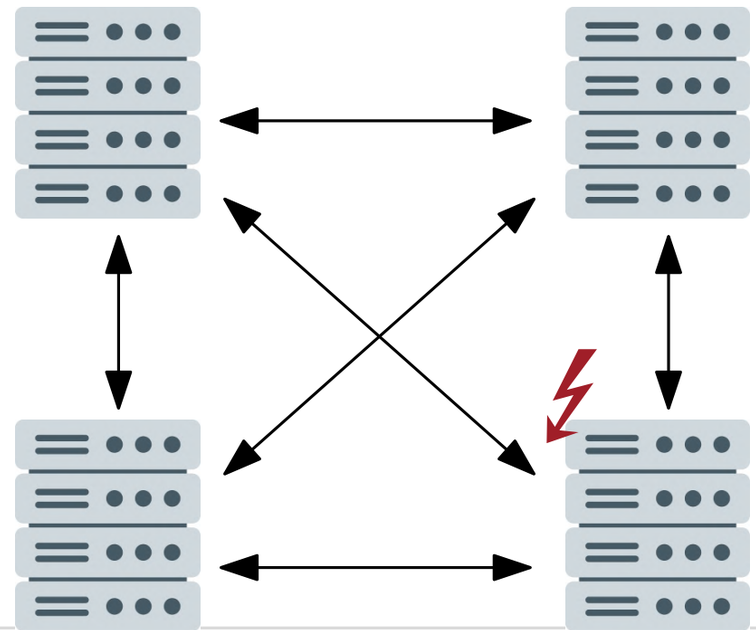
KIT – The Research University in the Helmholtz Association
Heidelberg Institute of Theoretical Studies

www.kit.edu • www.h-its.org

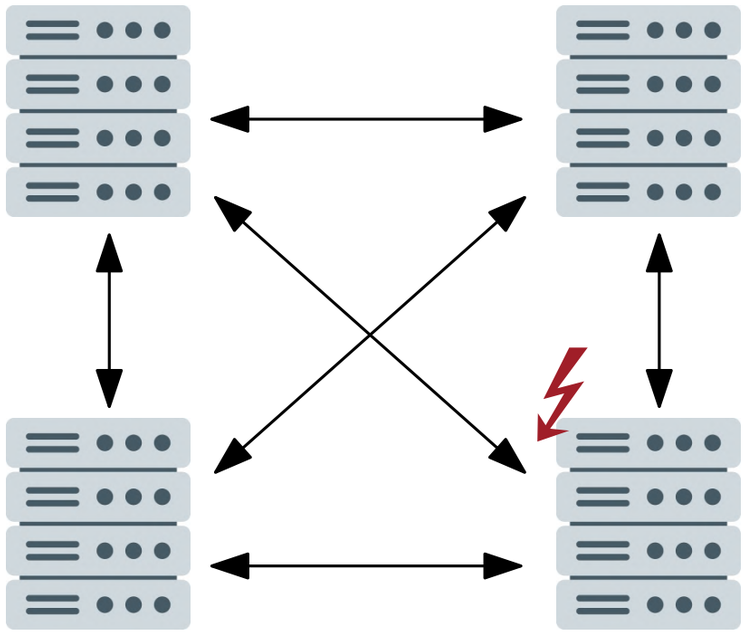
Fault-Tolerance

Fault-Tolerance

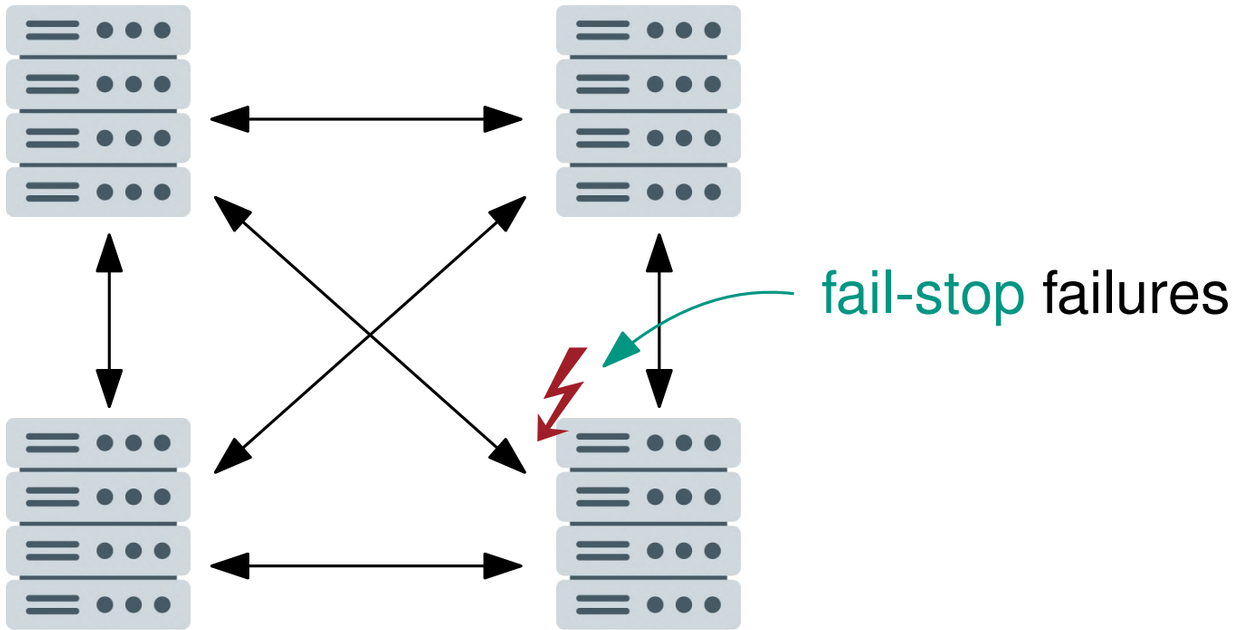
- Some existing supercomputers average over 2 hardware failures/day
- Increasing number of CPUs in supercomputers$\rightarrow$ increased failure probability
- More CPU s$\rightarrow$ more faults $\rightarrow$ more recoveries
- Lowering operational voltage of CPUs conserves energy$\rightarrow$ more faults
- Compute Node failure $\rightarrow$ reload *dynamic* program state and *static* (e.g., input) data



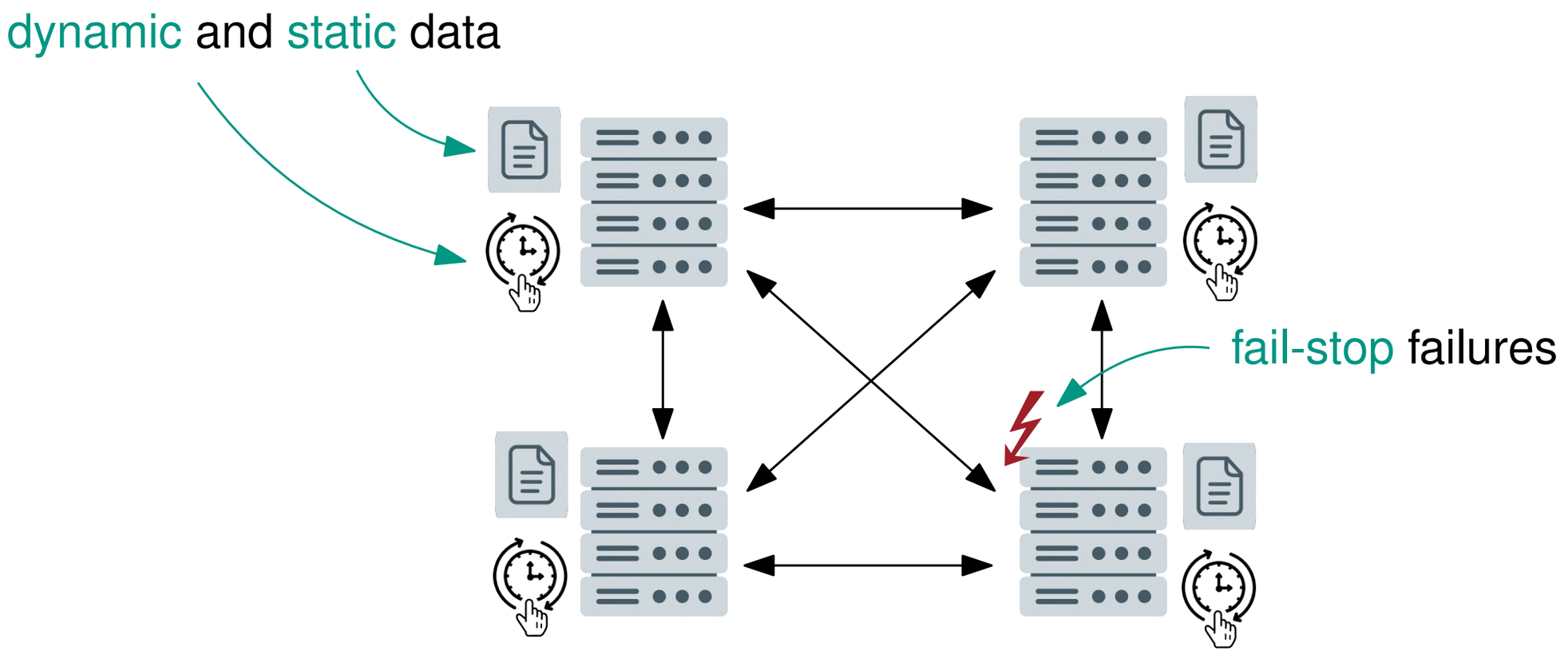
The Basics



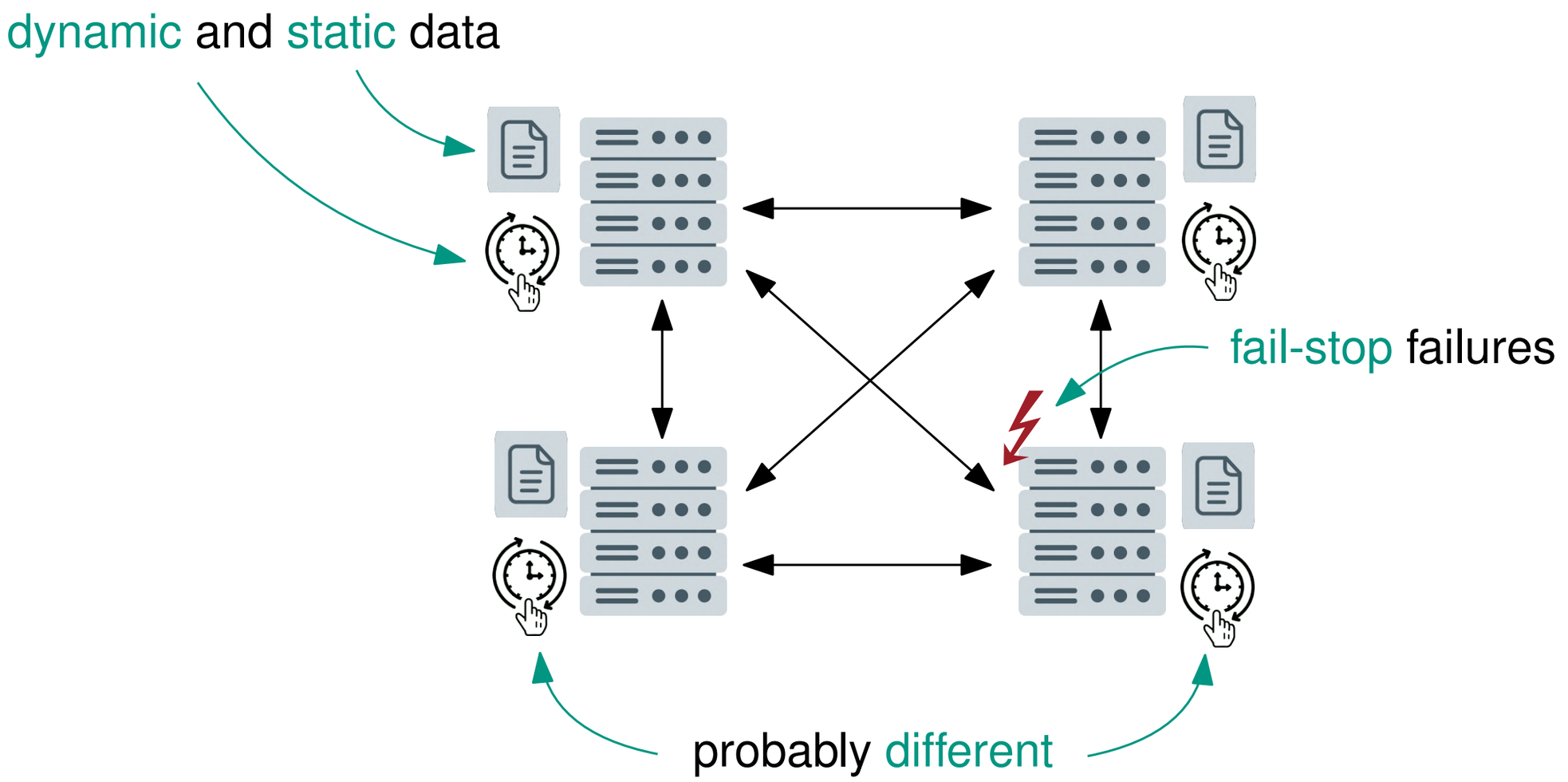
The Basics



The Basics

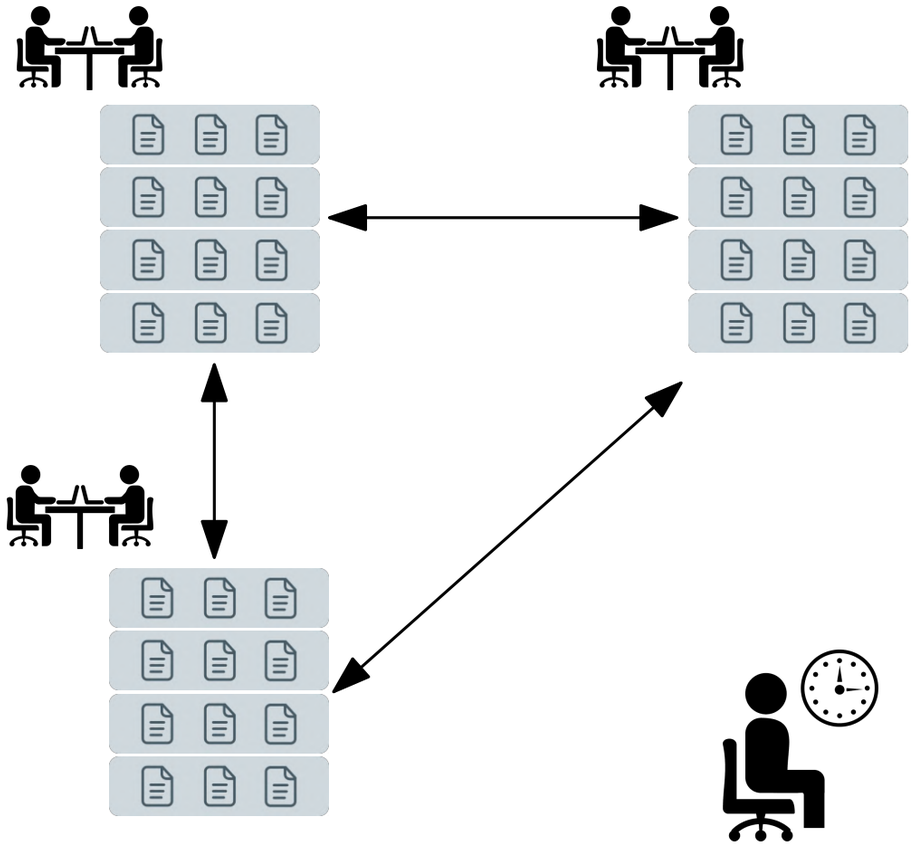


The Basics


dynamic and static data
fail-stop failures
probably different

Shrinking vs Substituting Recovery

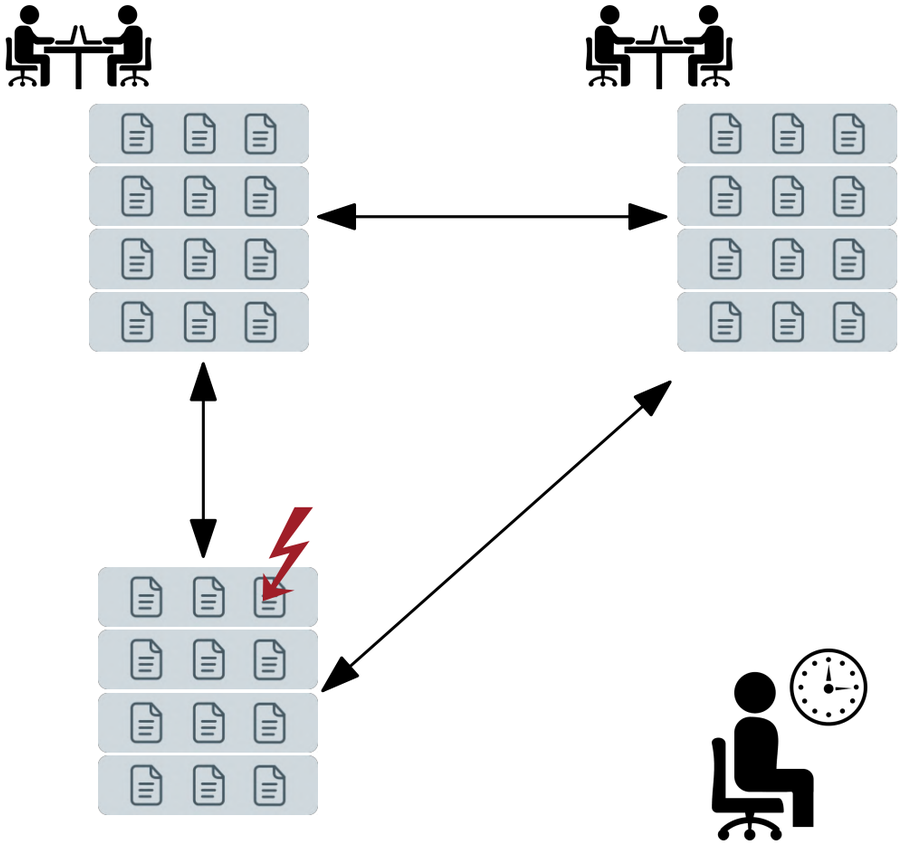
Substituting Recovery



Shrinking Recovery

Shrinking vs Substituting Recovery

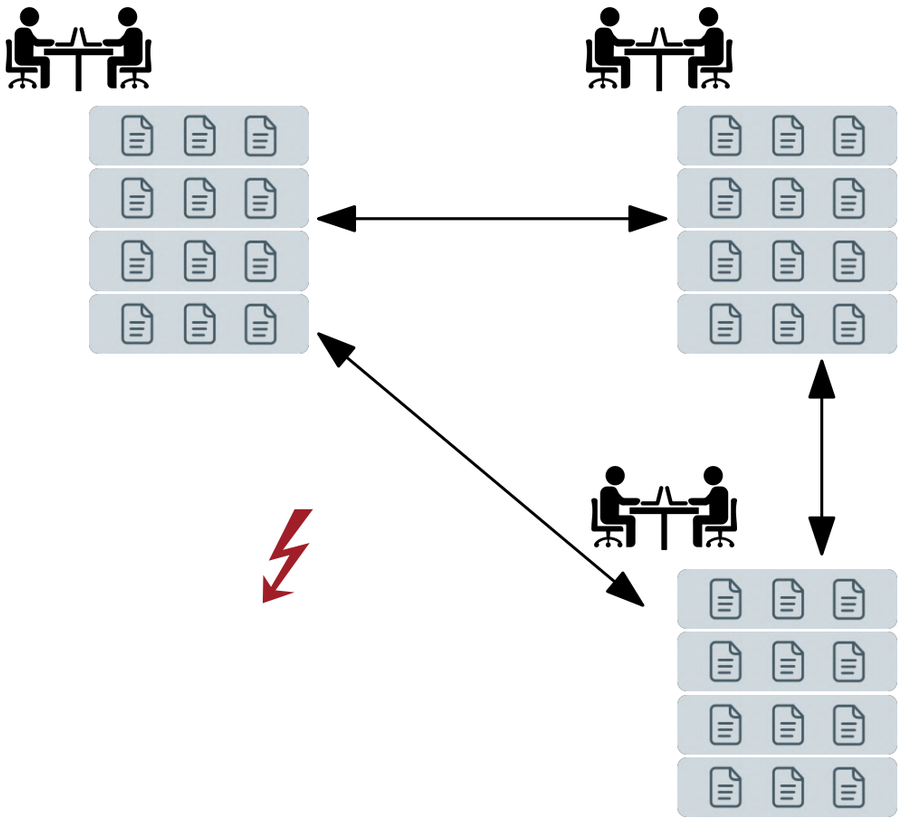
Substituting Recovery



Shrinking Recovery

Shrinking vs Substituting Recovery

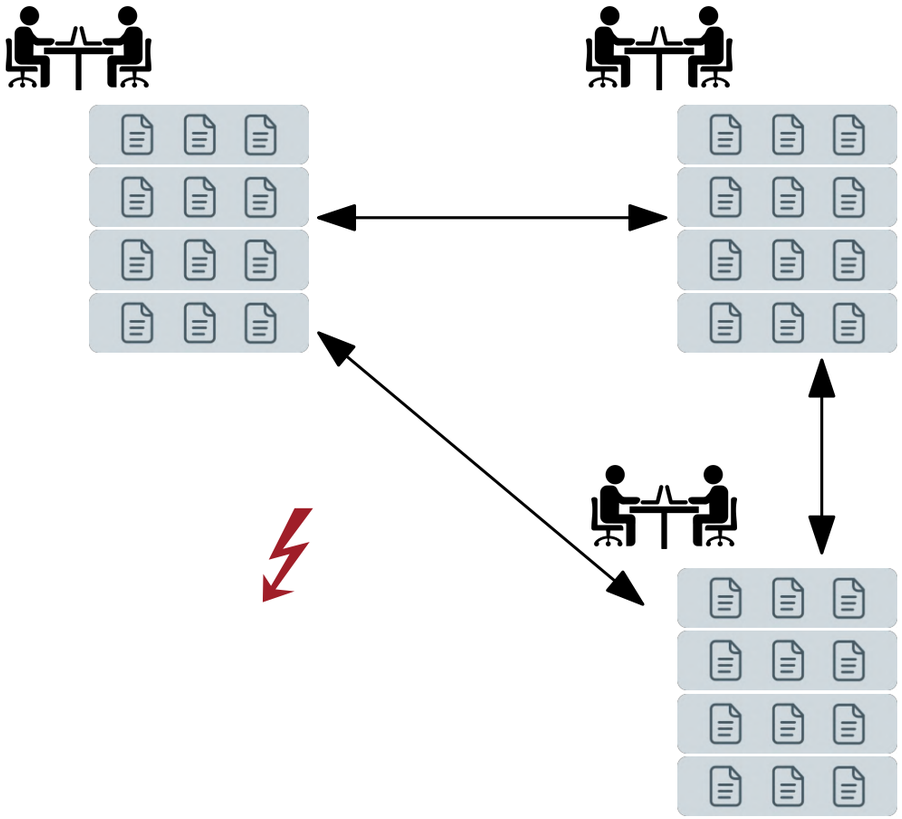
Substituting Recovery



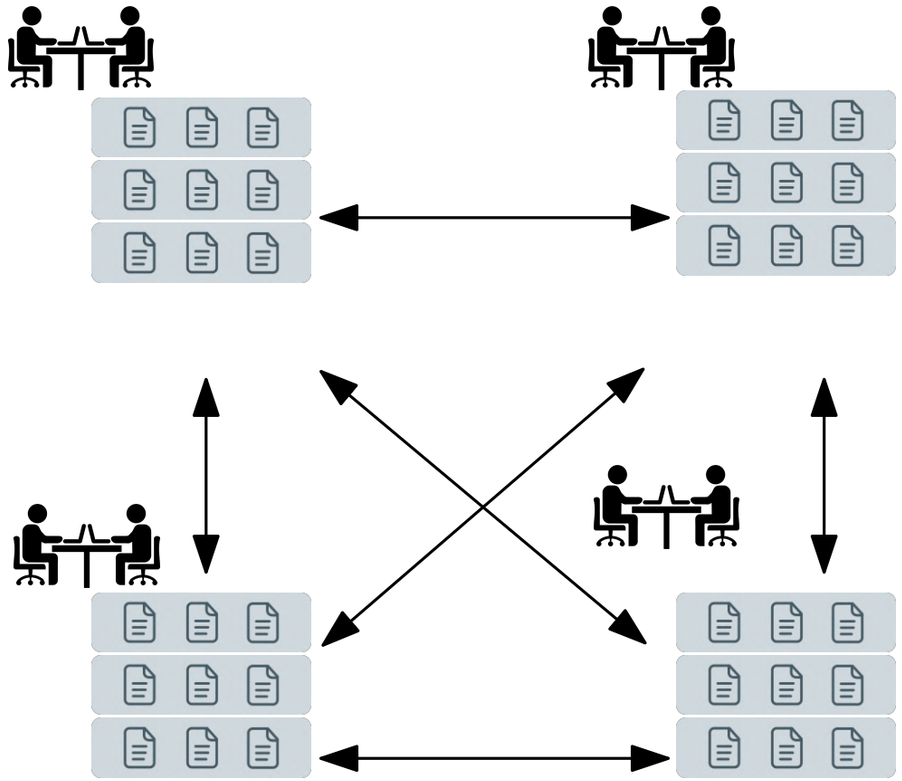
Shrinking Recovery

Shrinking vs Substituting Recovery

Substituting Recovery

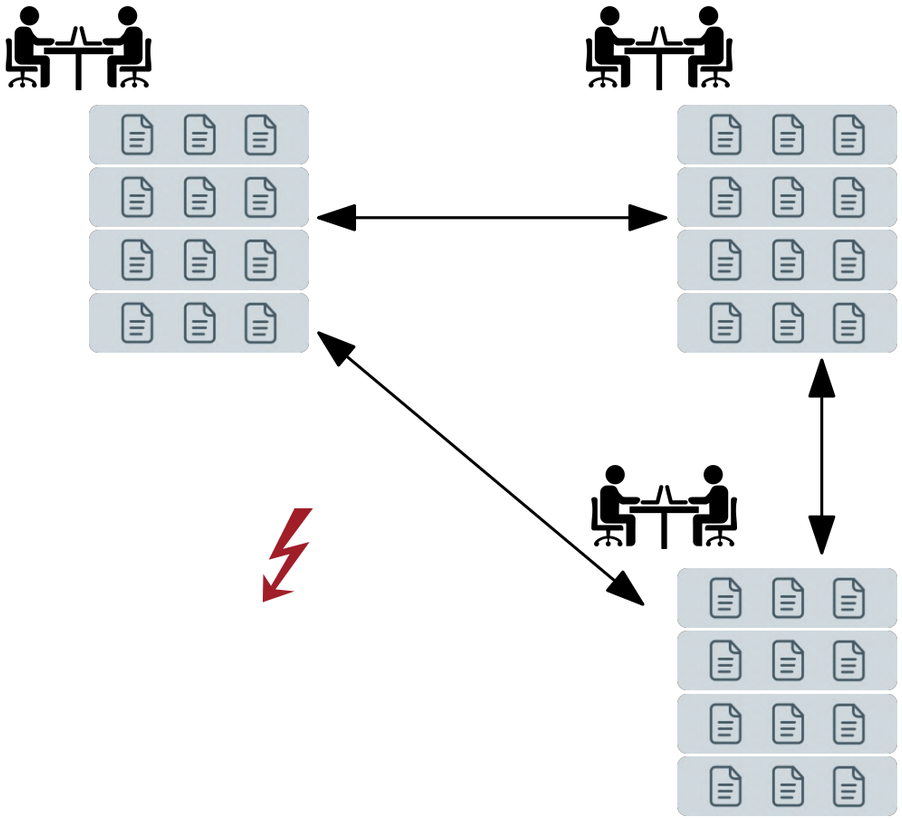


Shrinking Recovery

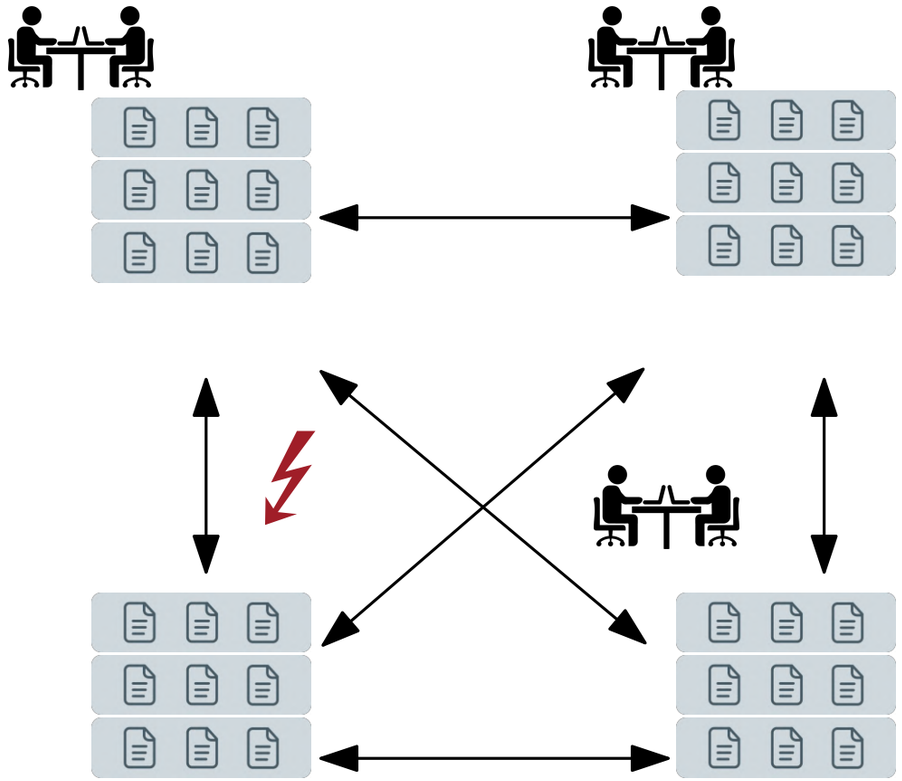


Shrinking vs Substituting Recovery

Substituting Recovery

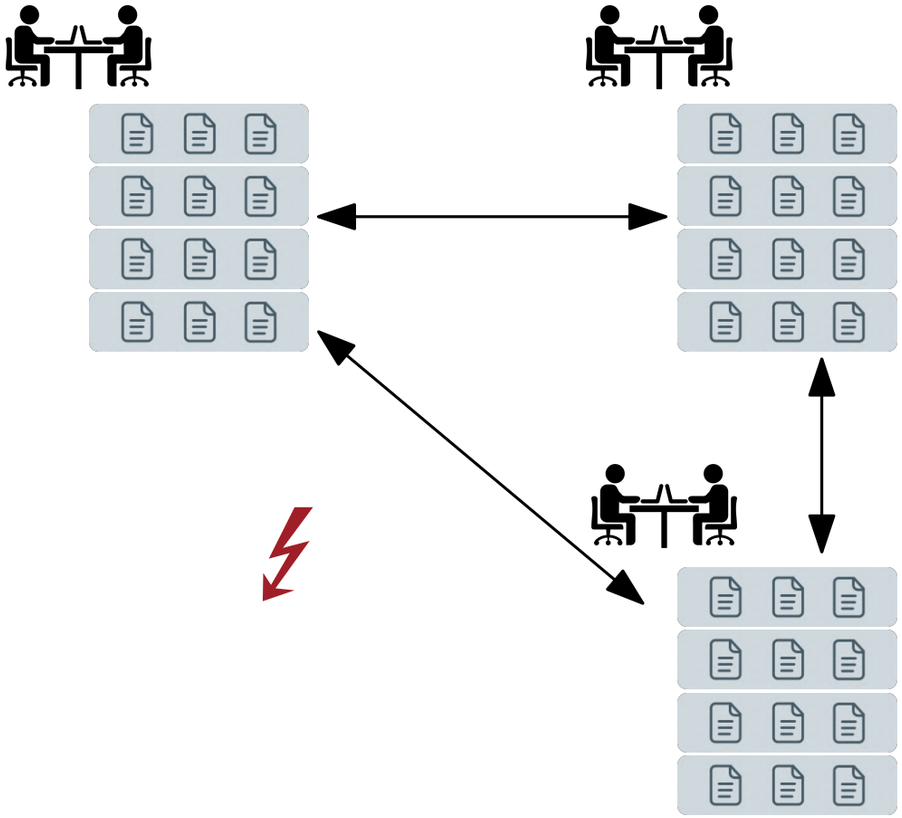


Shrinking Recovery

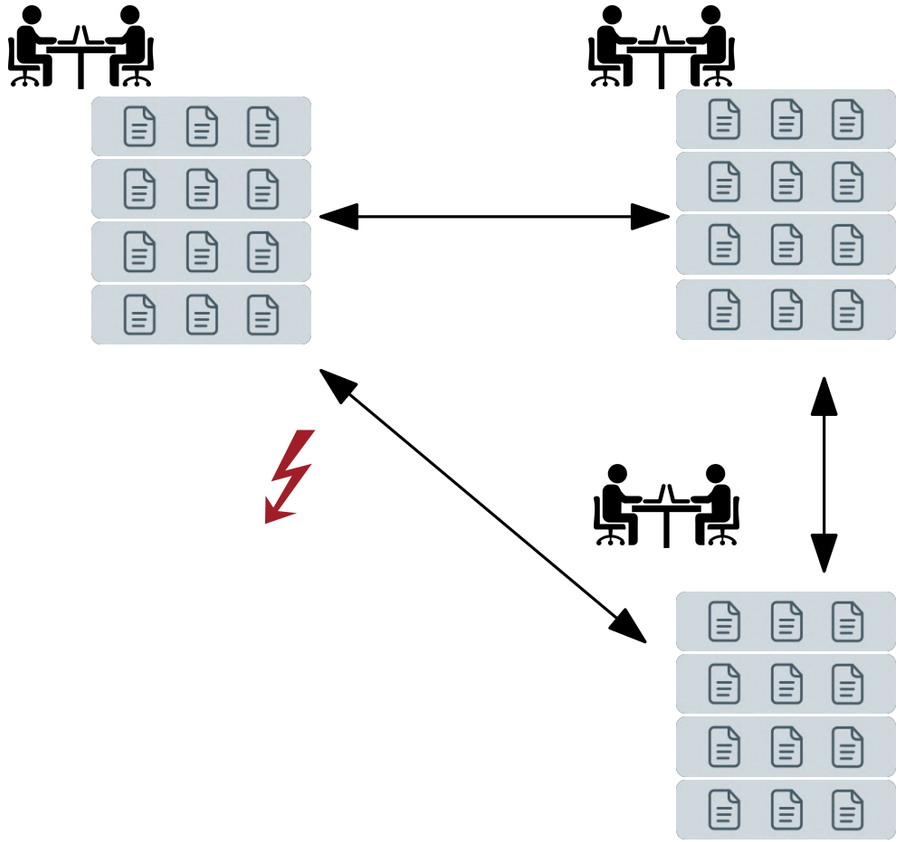


Shrinking vs Substituting Recovery

Substituting Recovery

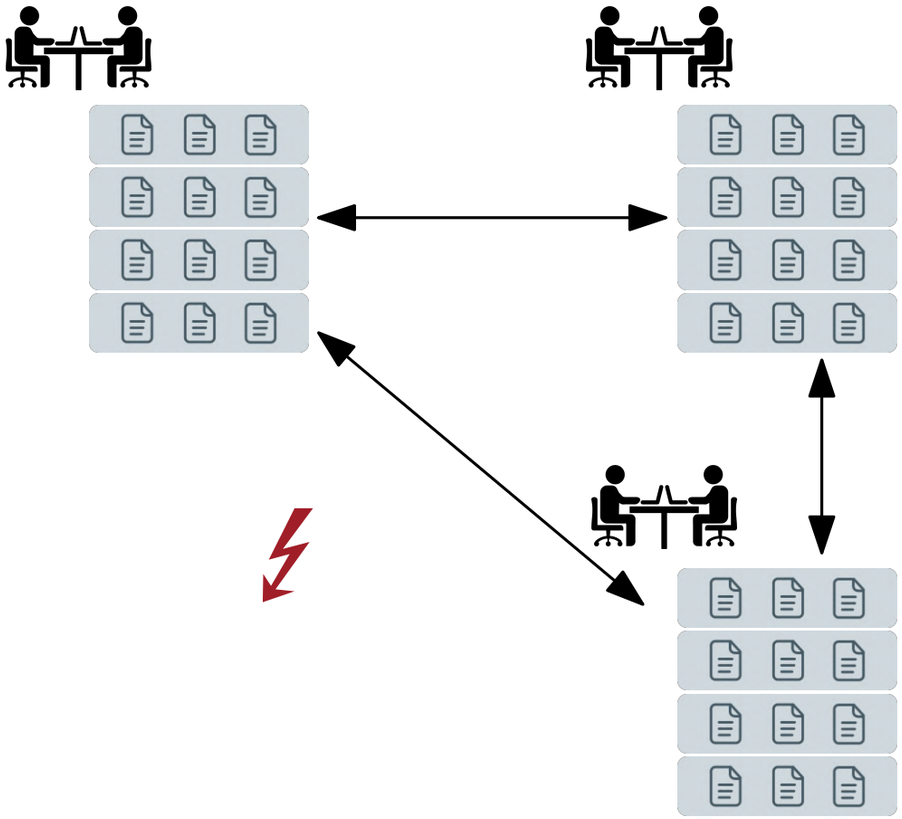


Shrinking Recovery



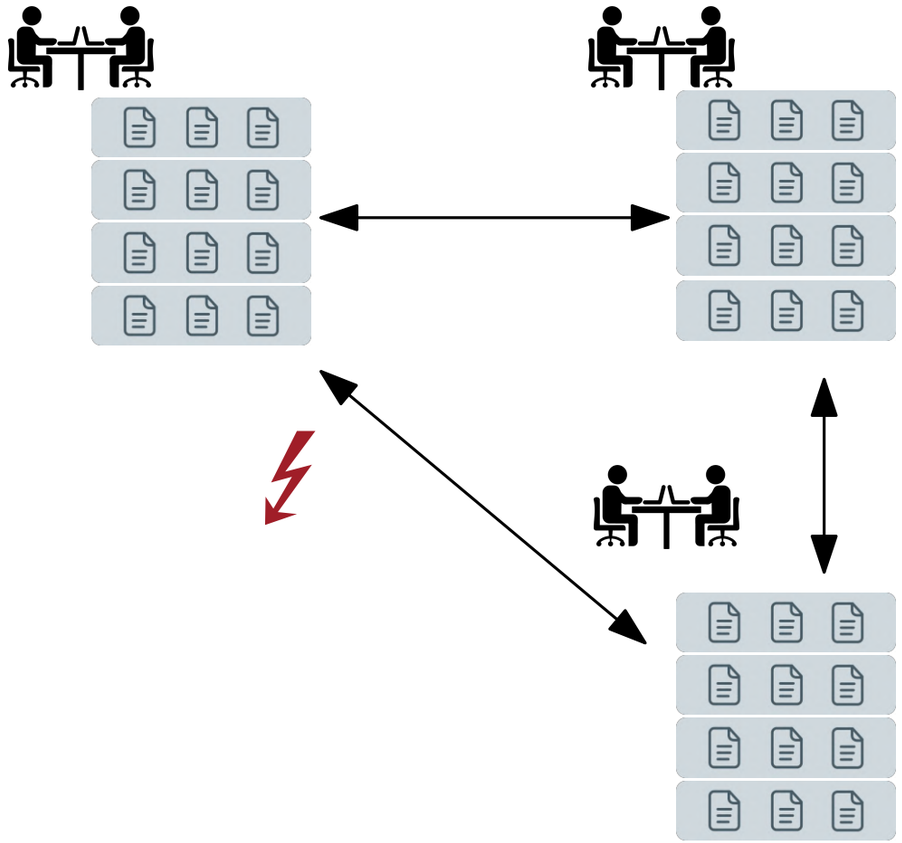
Shrinking vs Substituting Recovery

Substituting Recovery



- Up to $5\,\%$ of nodes idling
- Limited number of failures supported
- Recovery time does not scale

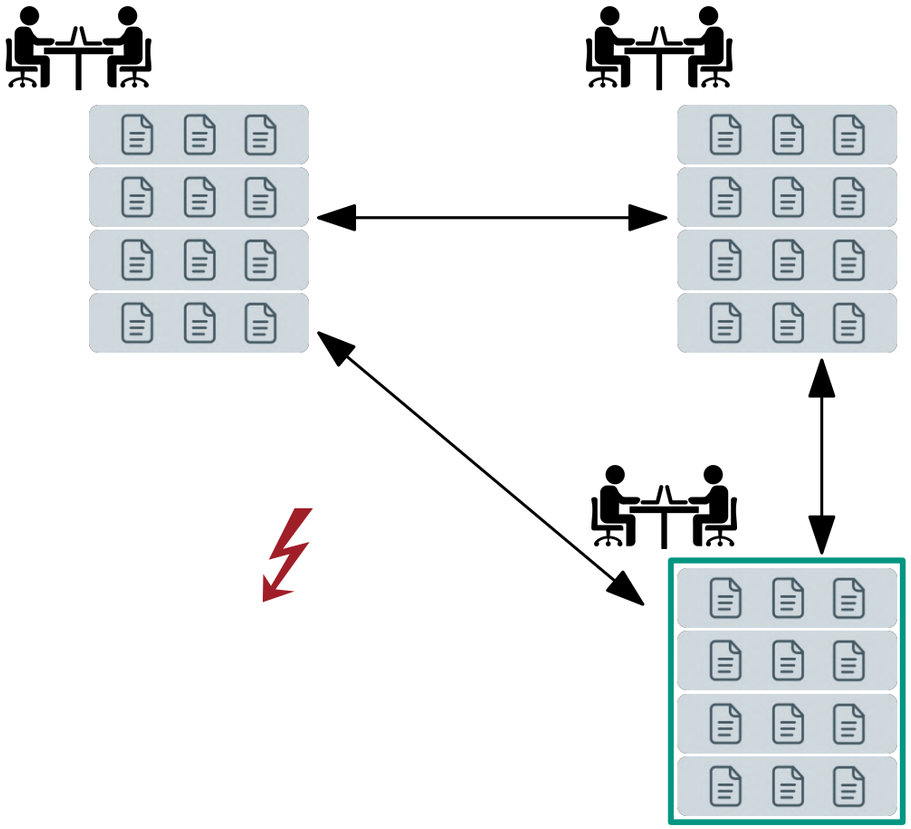
Shrinking Recovery



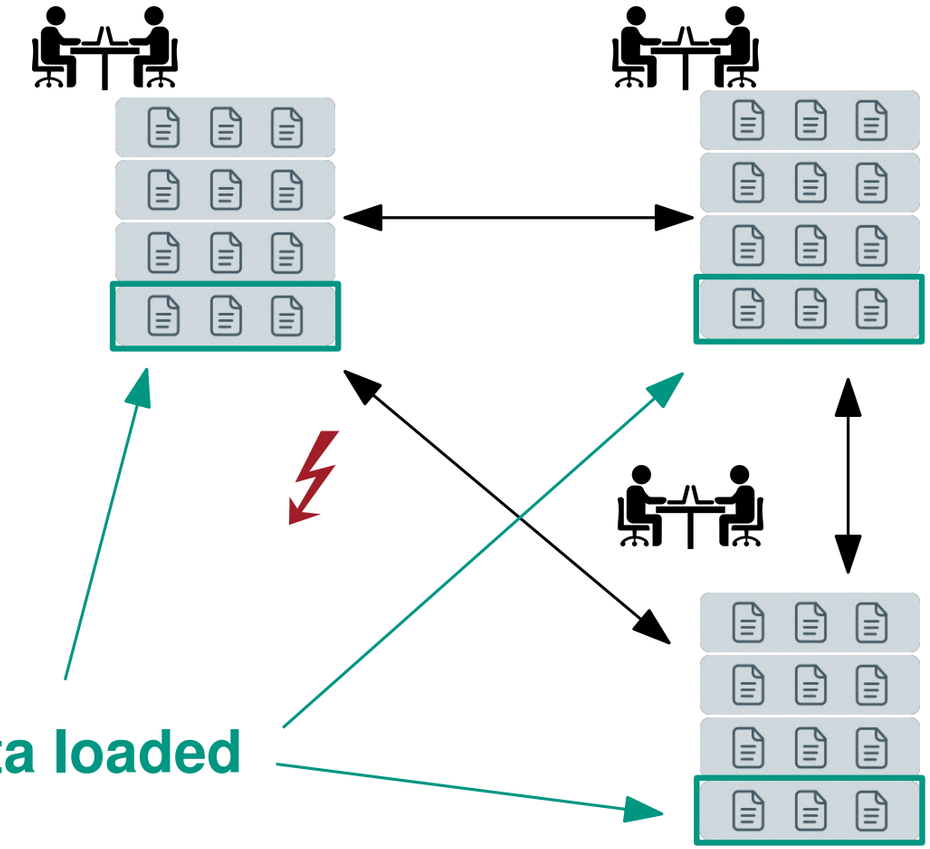
- All nodes participate in computation
- Unlimited number of failures supported
- Recovery time scales with $1/p$

Shrinking vs Substituting Recovery

Substituting Recovery



Shrinking Recovery



Single Node receives all messages

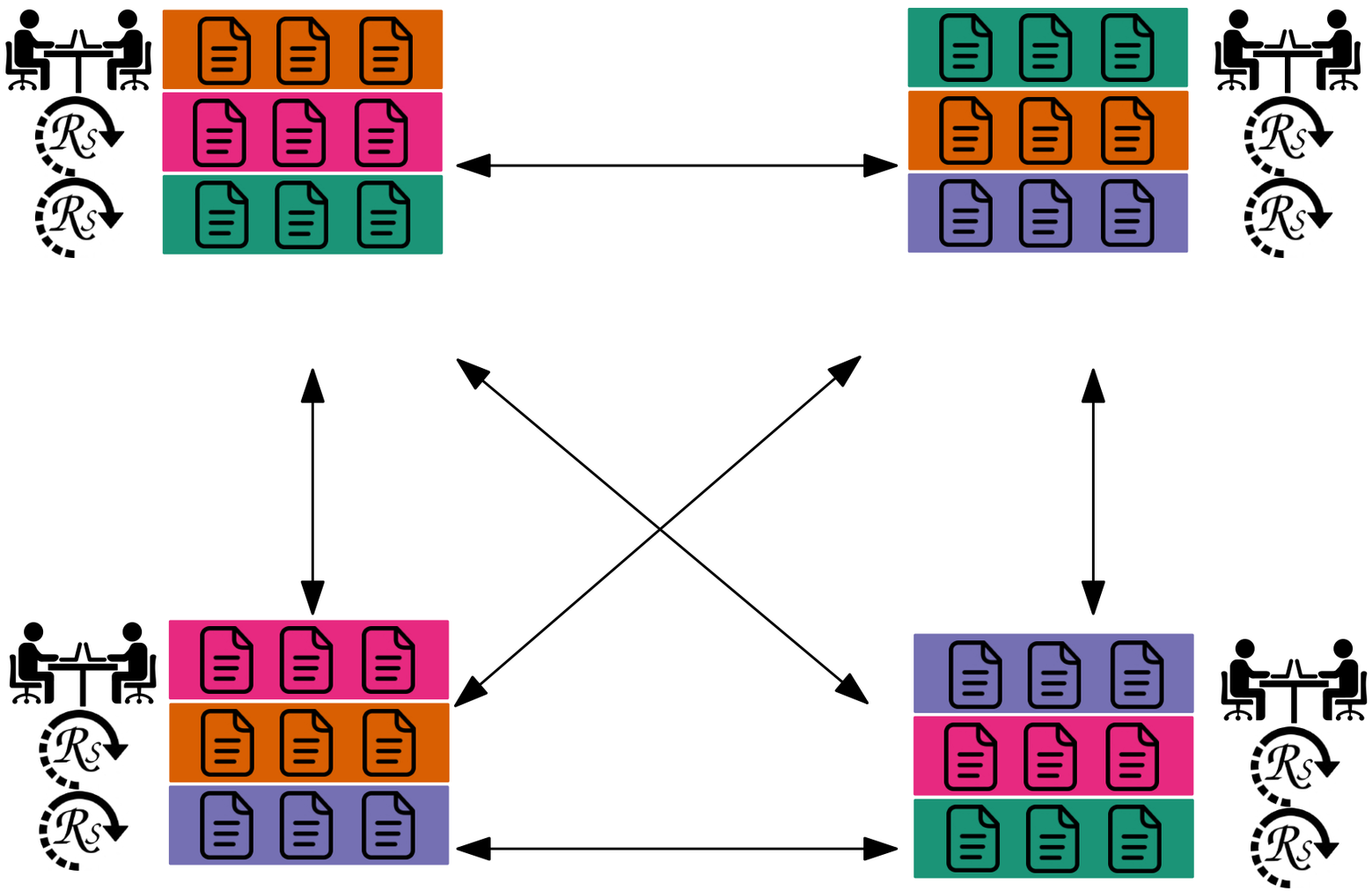
→ **bottleneck**

Design Goals

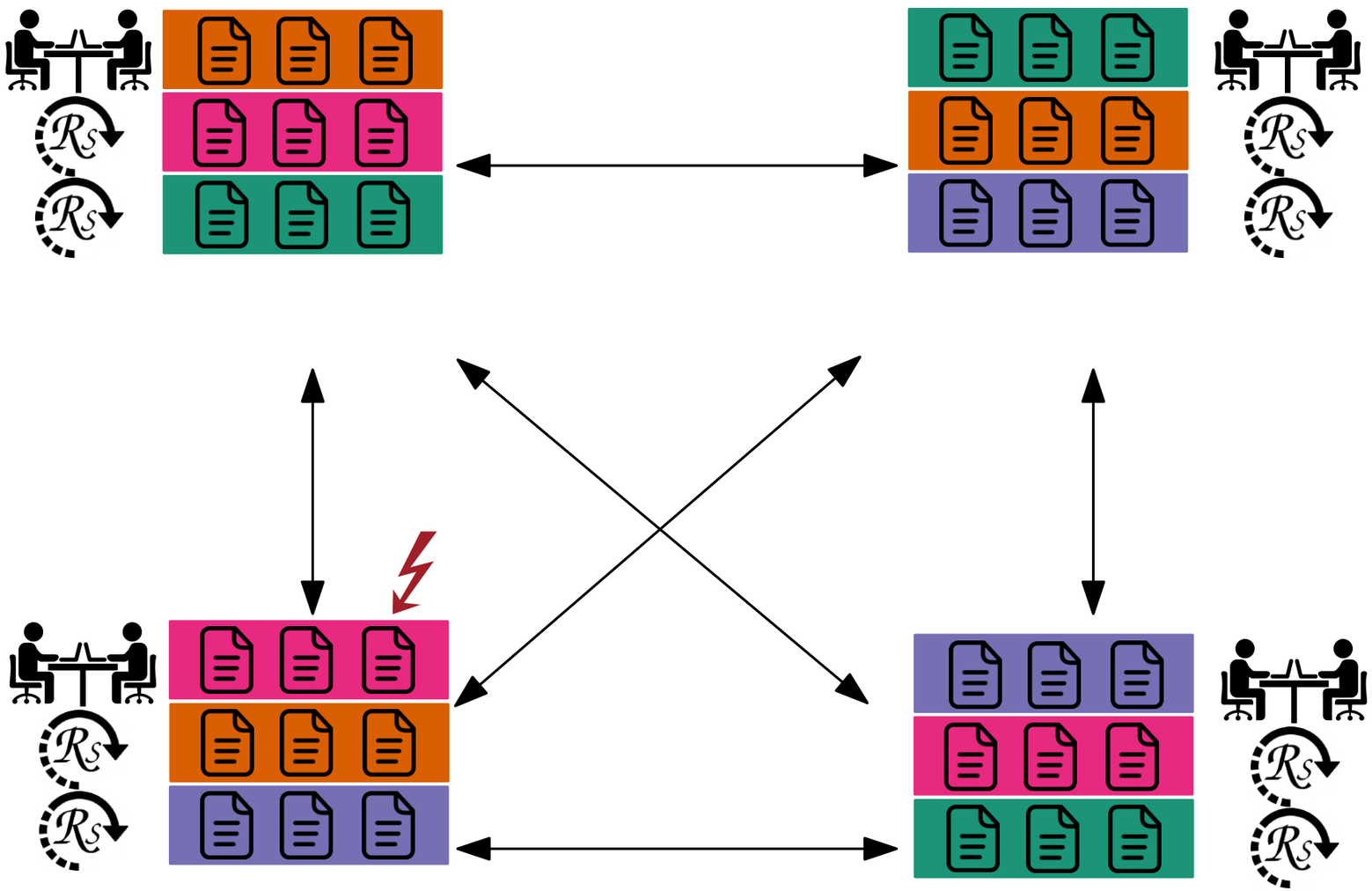
ReStore

- **in-memory** access to the parallel file system is a bottleneck
- **no spare nodes required** spare nodes are wasted resources
- **no checkpointing nodes required** checkpoint nodes are wasted resources
- **scalable recovery** $\in \mathcal{O}(1/p)$ time per failure
- **arbitrary replication level** more flexibility and robustness
- **rapid recovery** because recovery cannot be asynchronous

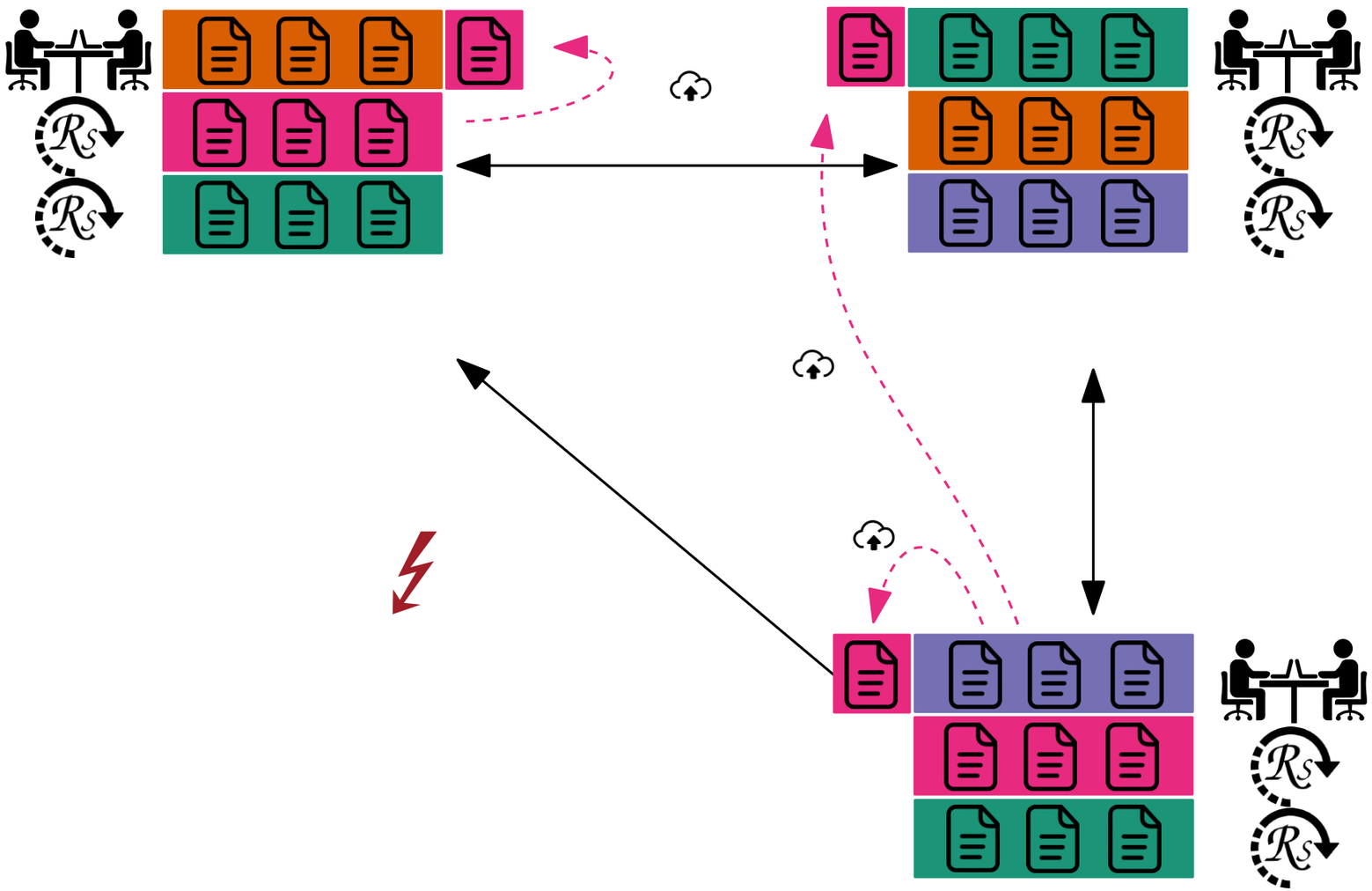
Basic Data Distribution



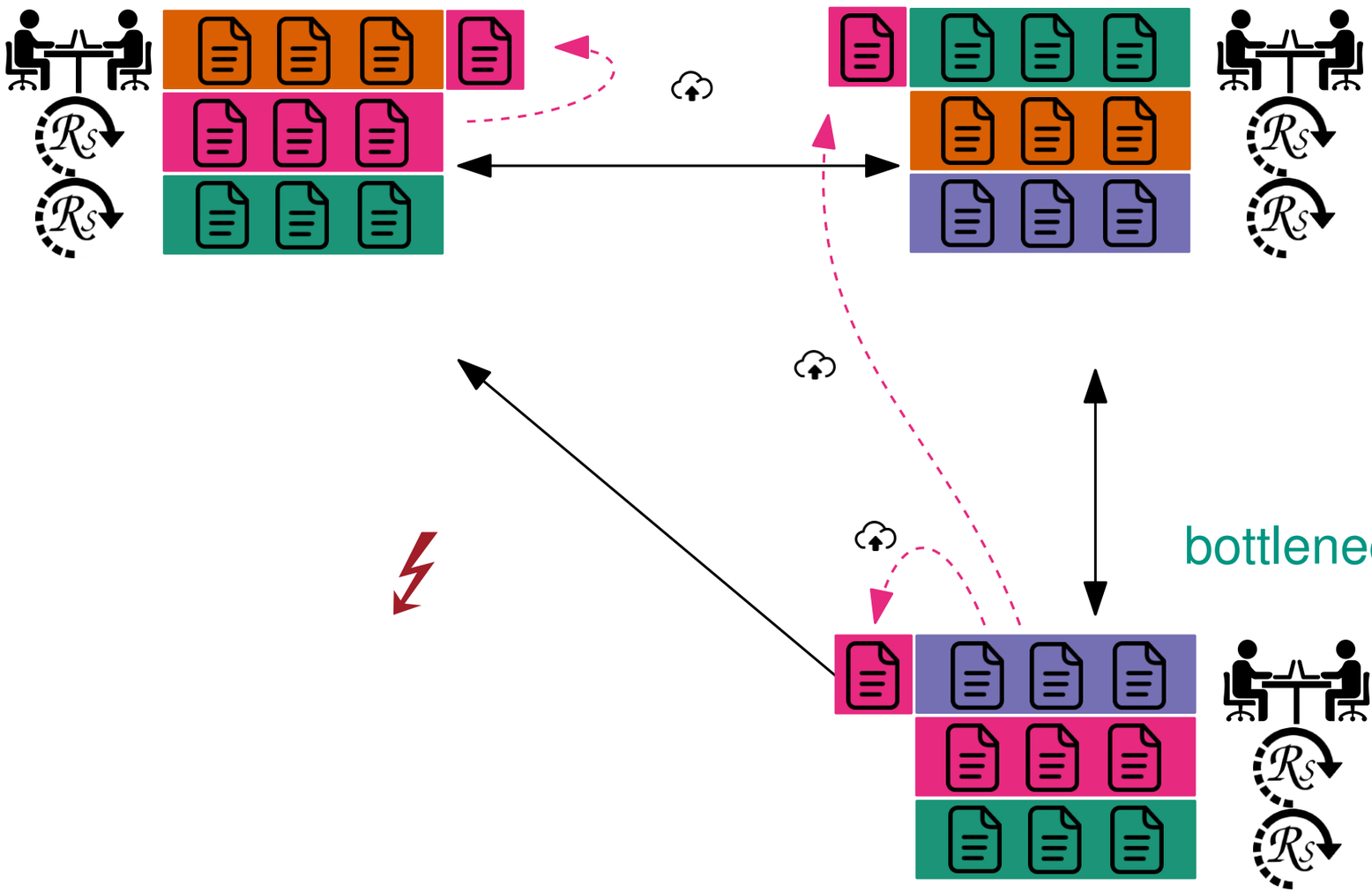
Basic Data Distribution



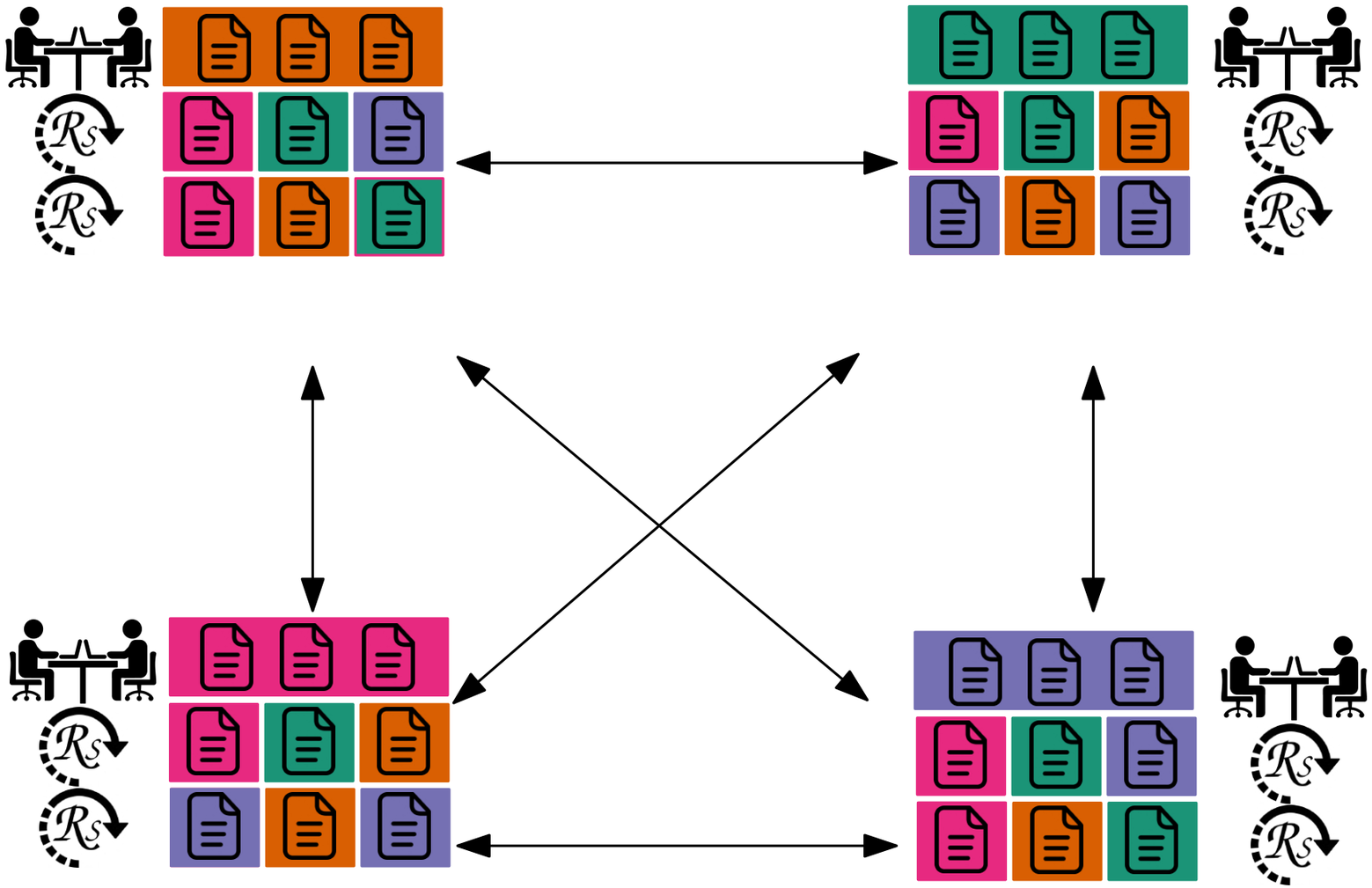
Basic Data Distribution



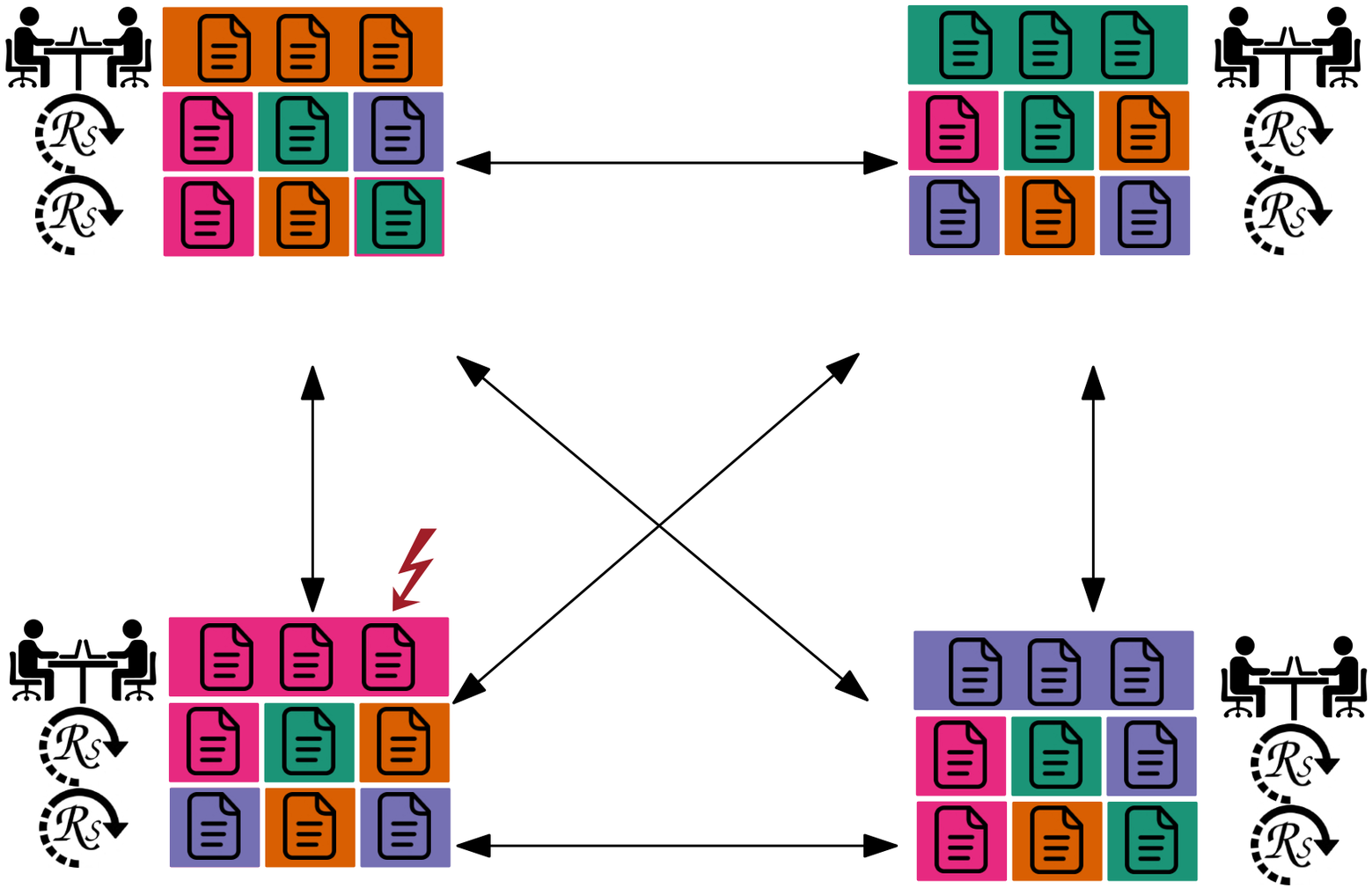
Basic Data Distribution



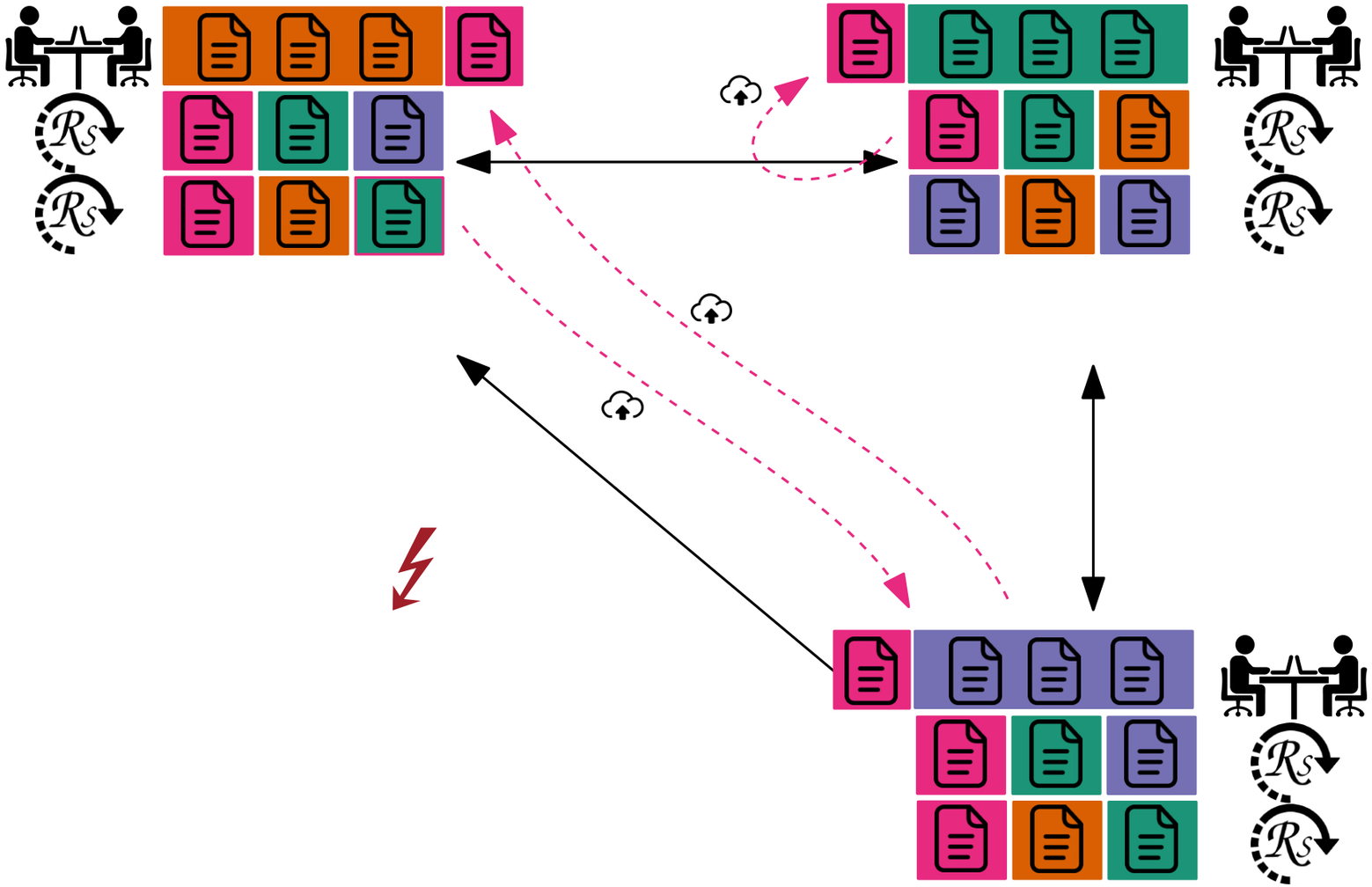
Data Distribution for Faster Recovery



Data Distribution for Faster Recovery

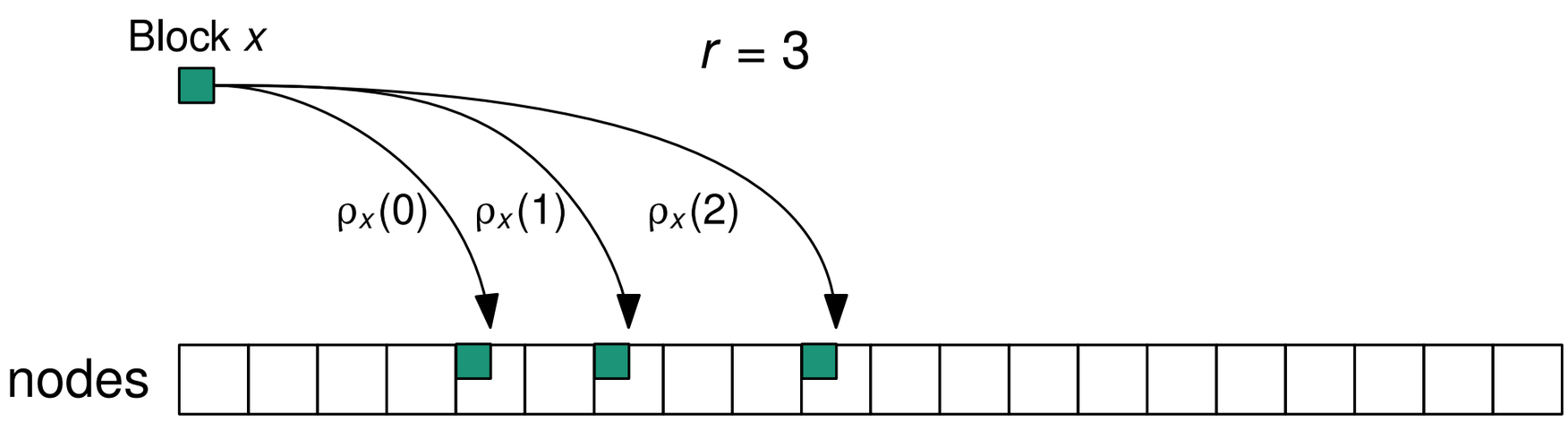


Data Distribution for Faster Recovery



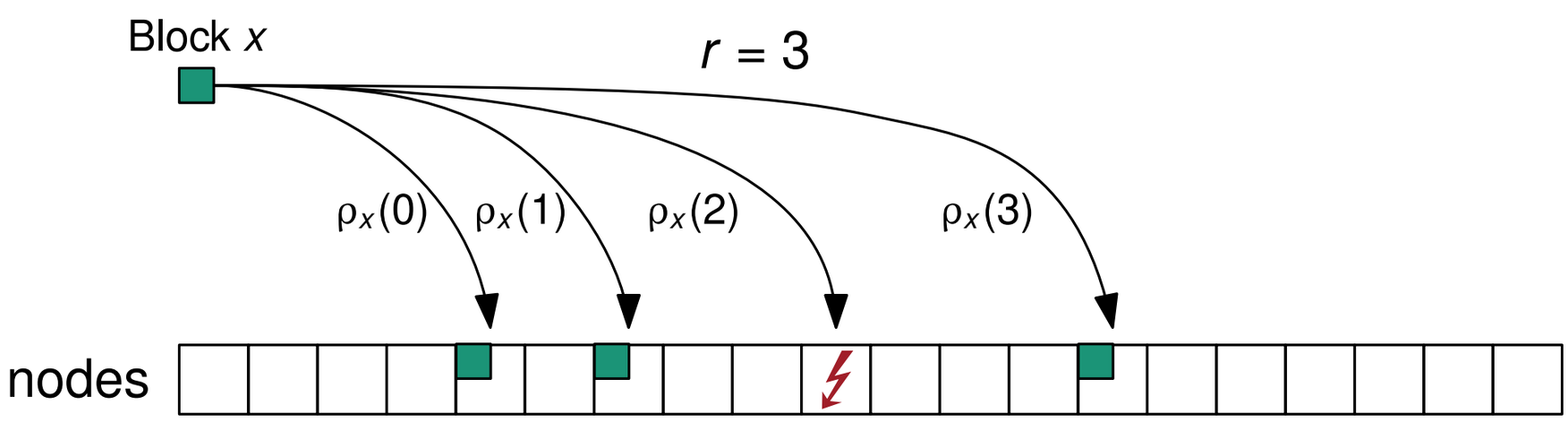
Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$



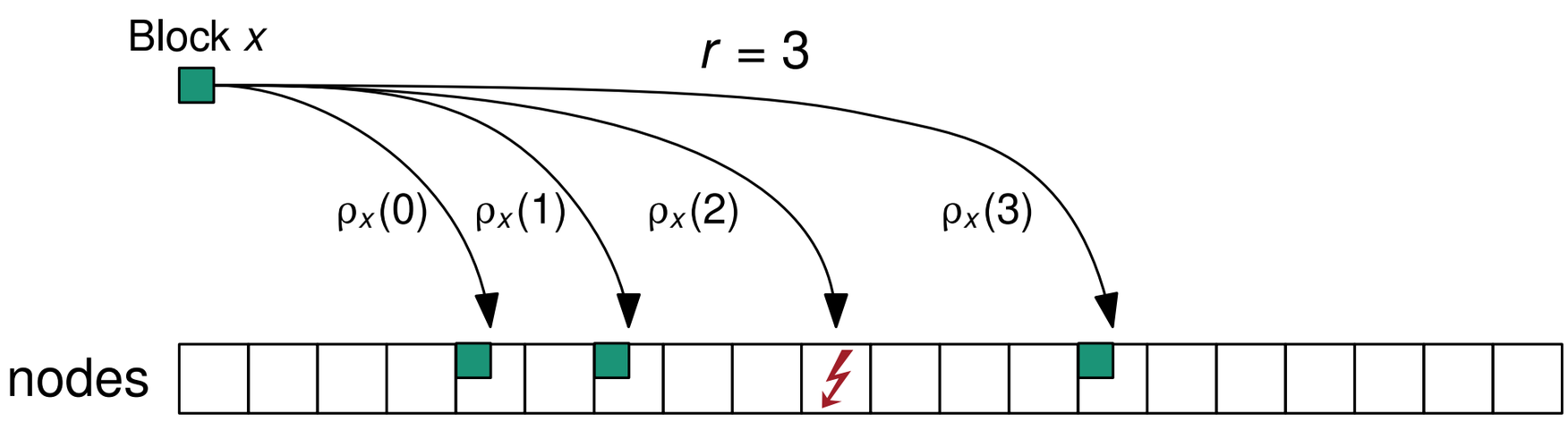
Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$



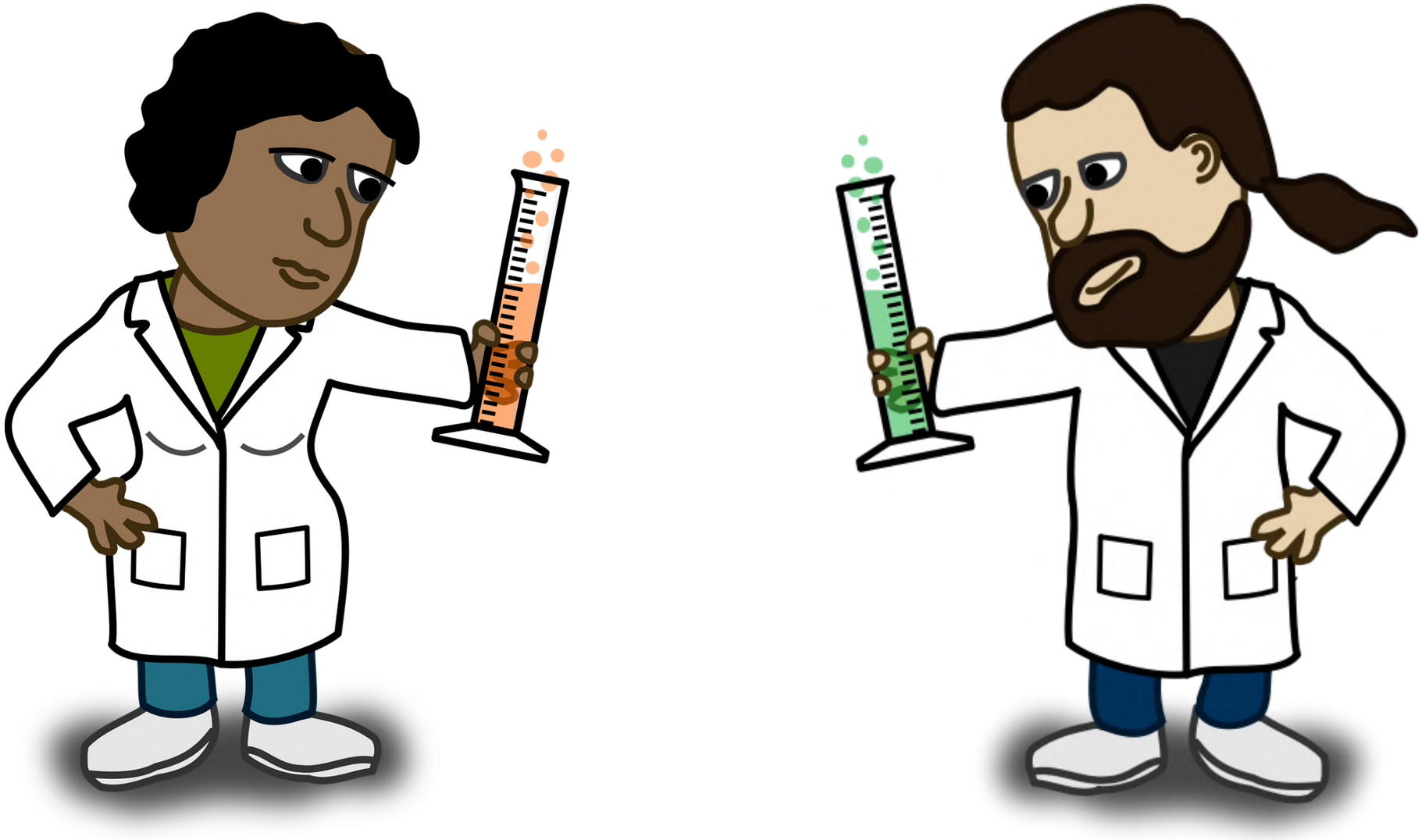
Recovering Replicas After a Node Failure

- **Goal:** Restore lost replicas after a failure; copying only the lost data
- **Idea:** For each block x, draw pseudorandom permutation ρ_x on $[0, p-1]$
- Place copies on $\rho_x(0), \rho_x(1), \ldots$



No need to redistribute any block that did not lose a replica!

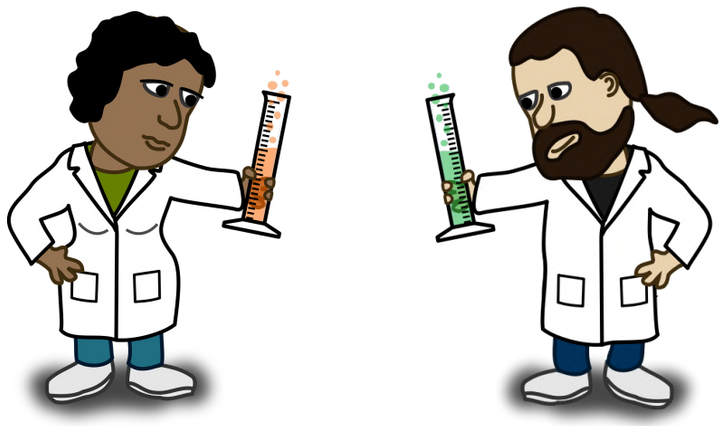
Computational Reproducibility



OpenClipart

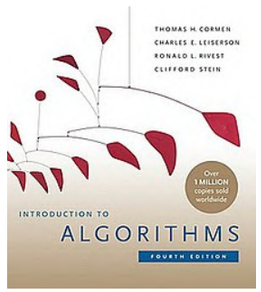
Computational Reproducibility

- Computational Reproducibility: Run program with the same parameters n times $\rightarrow$ same result?!
- Archiving, storing, and sharing the data, as well as providing scripts for reproducing results and figures are not enough.



Computational Reproducibility

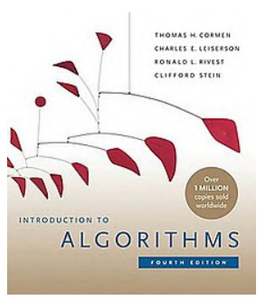
For reproducible results, fix



- algorithm

Computational Reproducibility

For reproducible results, fix

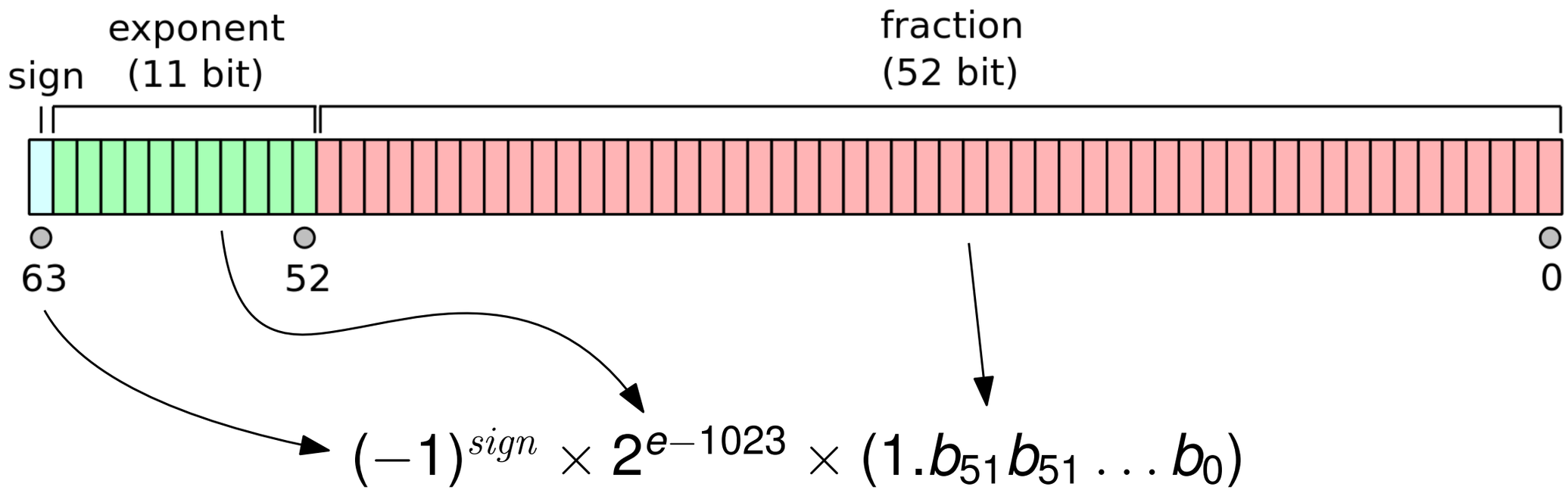


algorithm

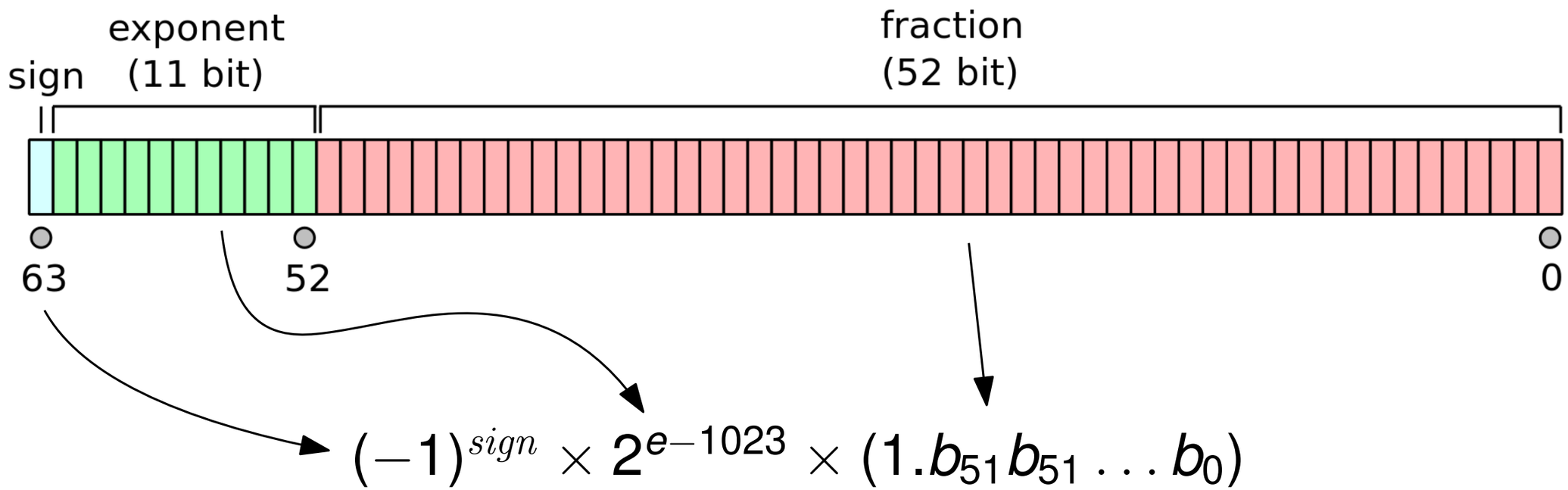
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=us-ascii" />
        <script type="text/javascript">
            function reDo() {top.location.reload();}
            if (navigator.appName == 'Netscape') {top.onresize = reDo;}
            dom=document.getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

source code

Floating-Point Numbers



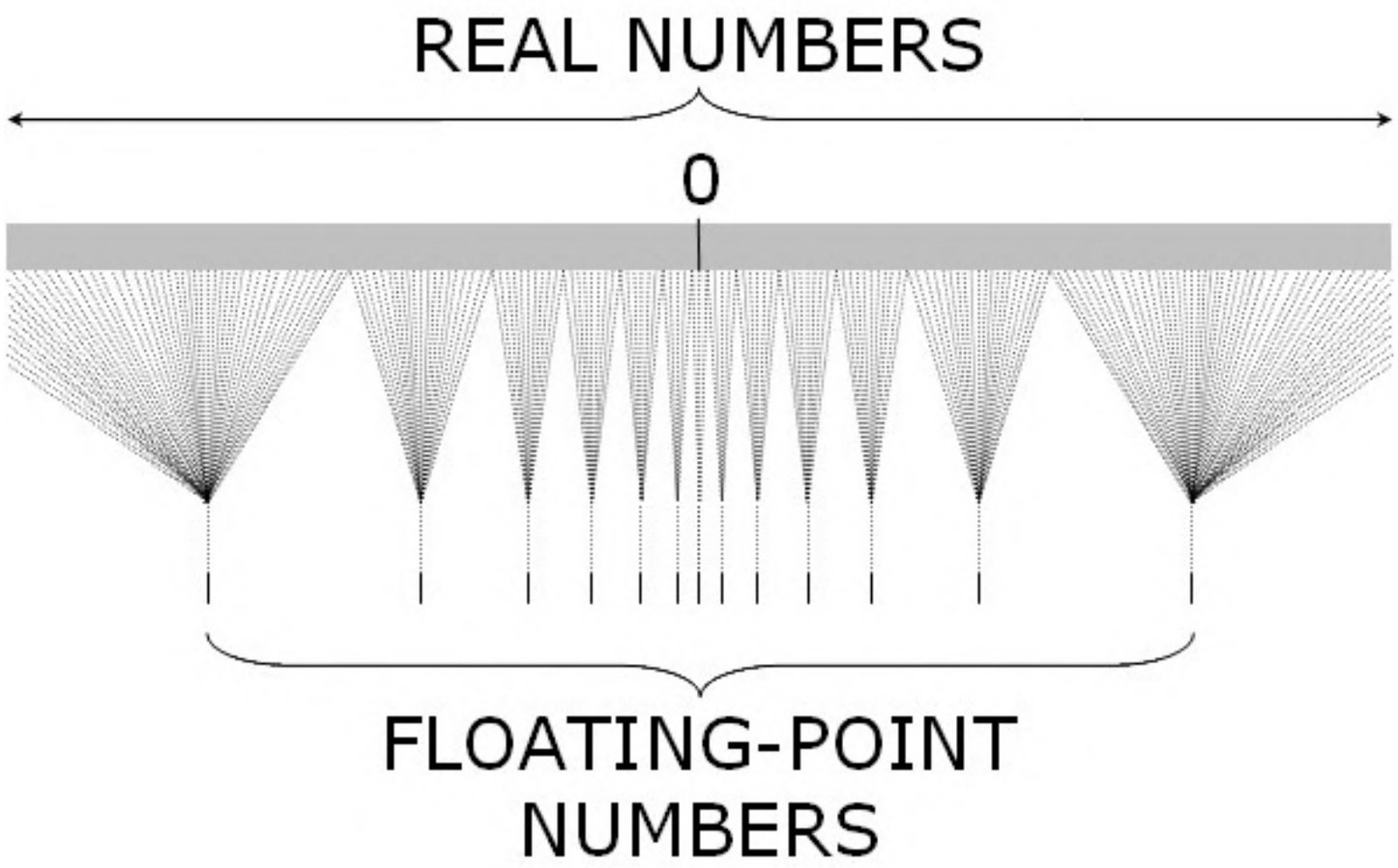
Floating-Point Numbers



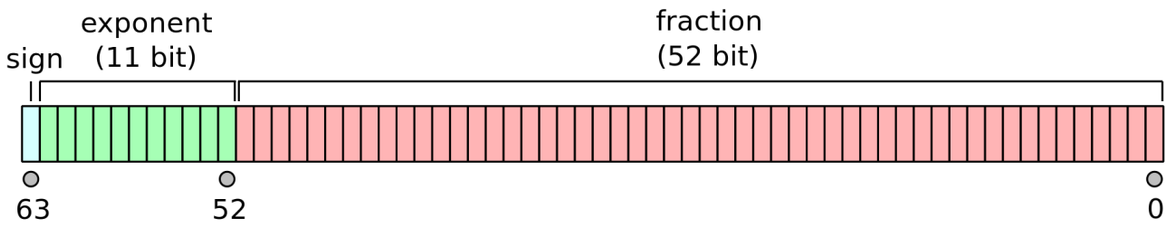
15 to 17 decimal digits

Floating-Point Numbers

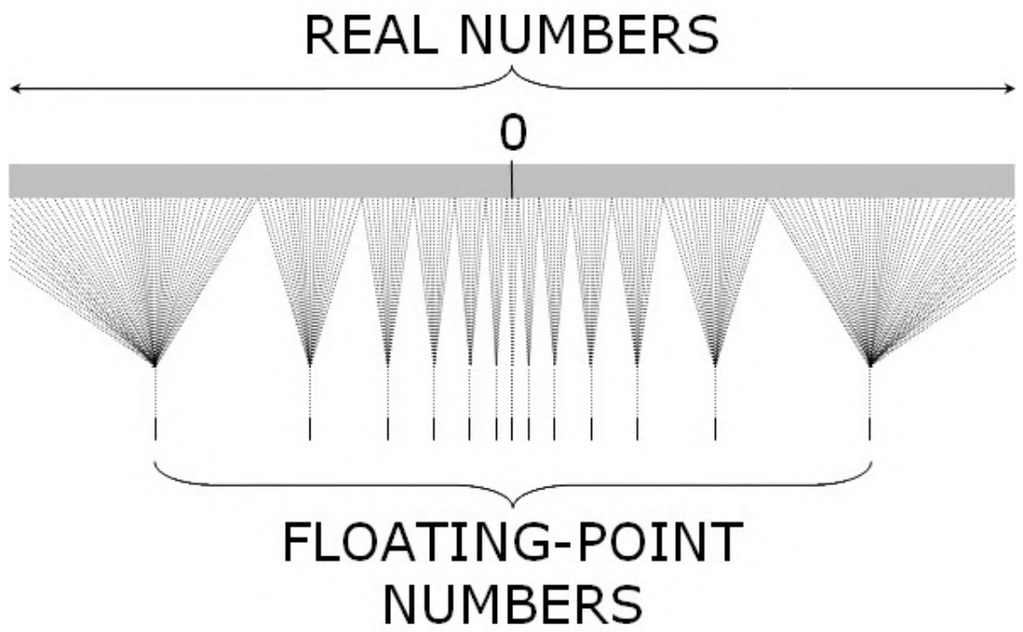
Floating-Point numbers are an imperfect mapping of the infinite real numbers to a finite number of machine values!



Floating-Point Numbers



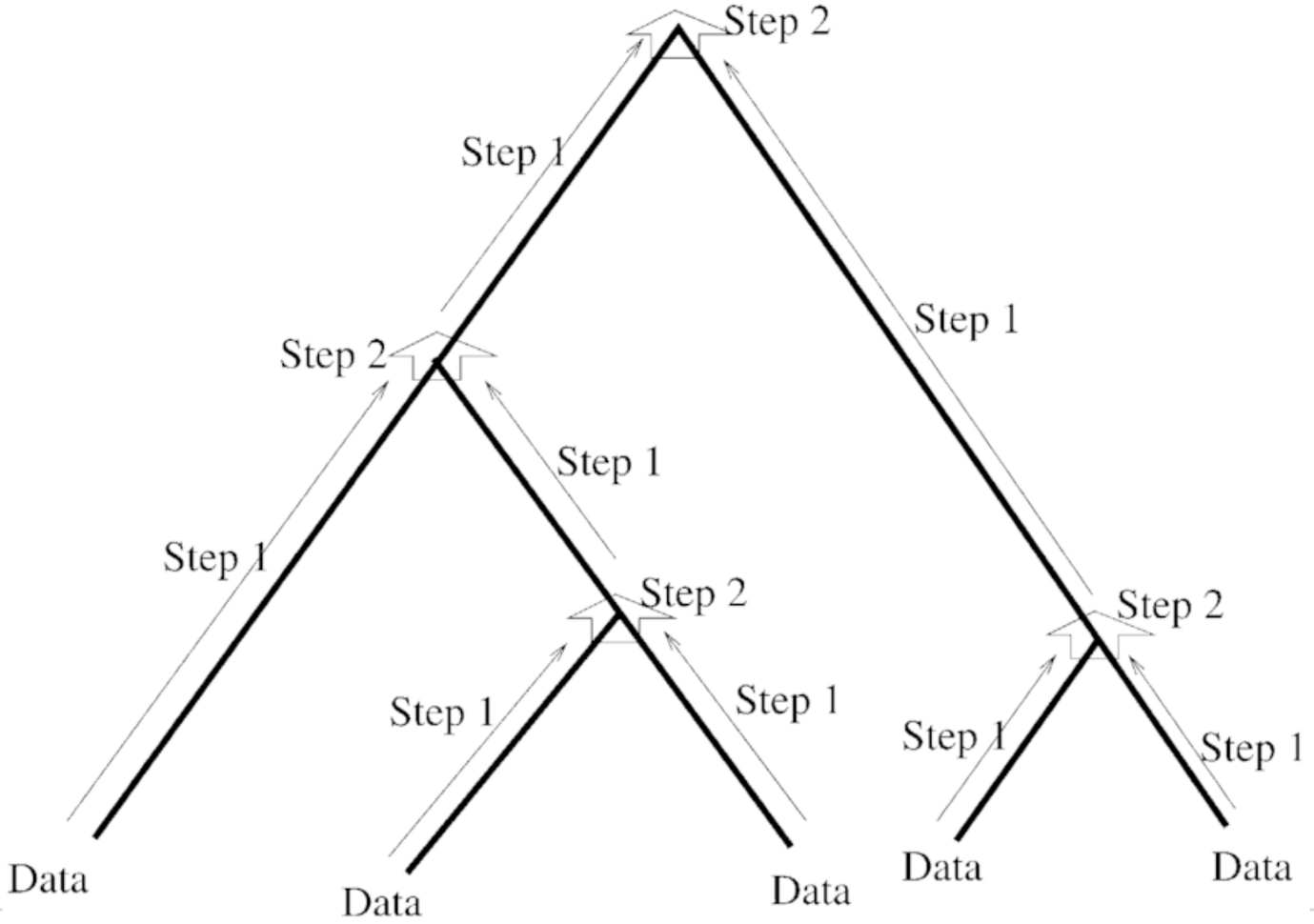
$$(-1)^{sign} \times 2^{e-1023} \times (1.b_{51}b_{51}\ldots b_0)$$



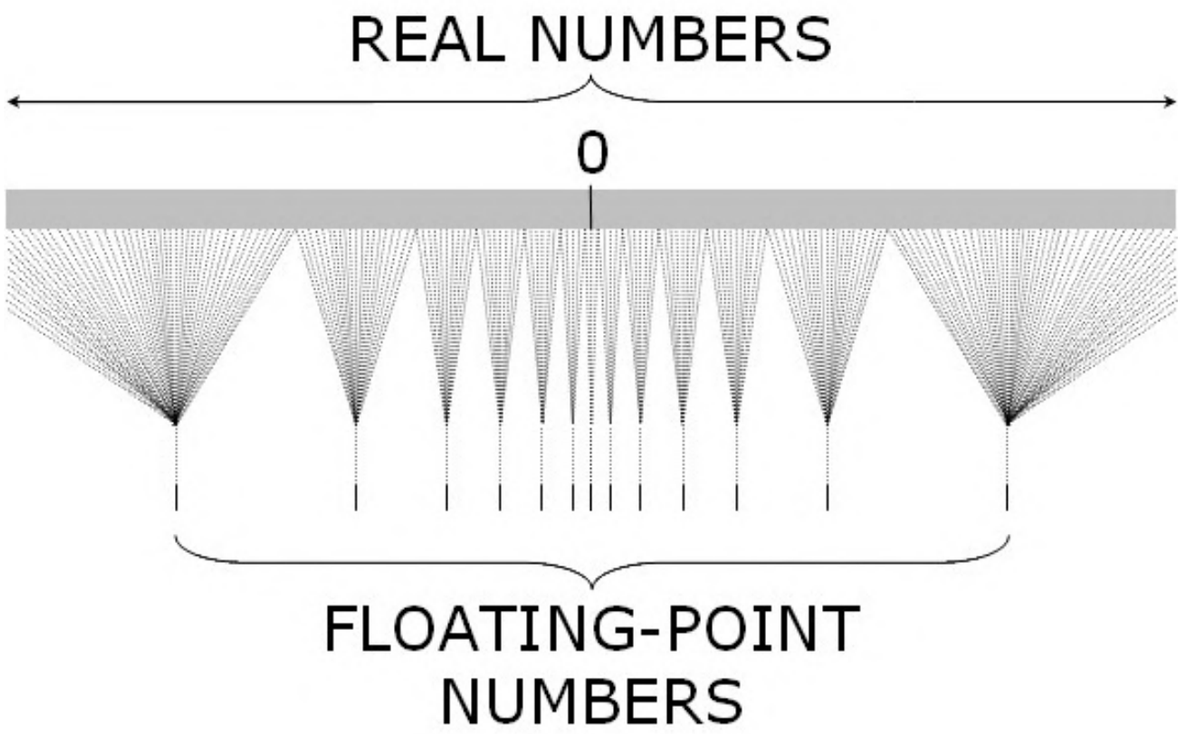
- $2^{52} + 0.2 = 2^{52}$ (next number after 2^{52} is $2^{52} + 1$)
- $1 + 1/2^{54} = 1$ (next number is $1 + 1/2^{52}$)
- Between 2^n and 2^{n+1} there are always 2^{52} evenly spaced values

Floating-Point Math in Phylogenetics

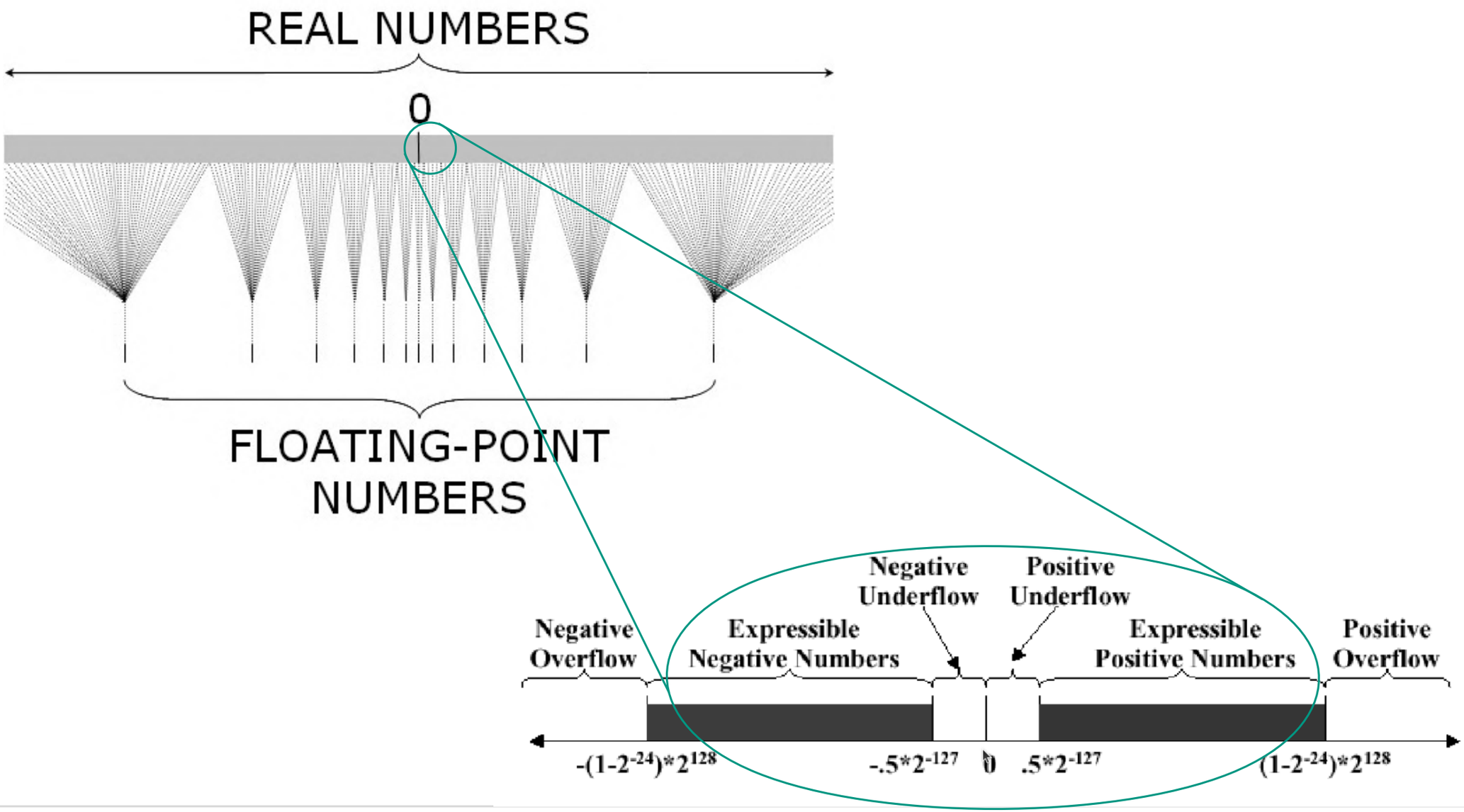
- Likelihood of a tree: multiply probabilities ($\in [0, 1]$) along a tree
- $\rightarrow$ values get smaller and smaller as we approach the root of the tree



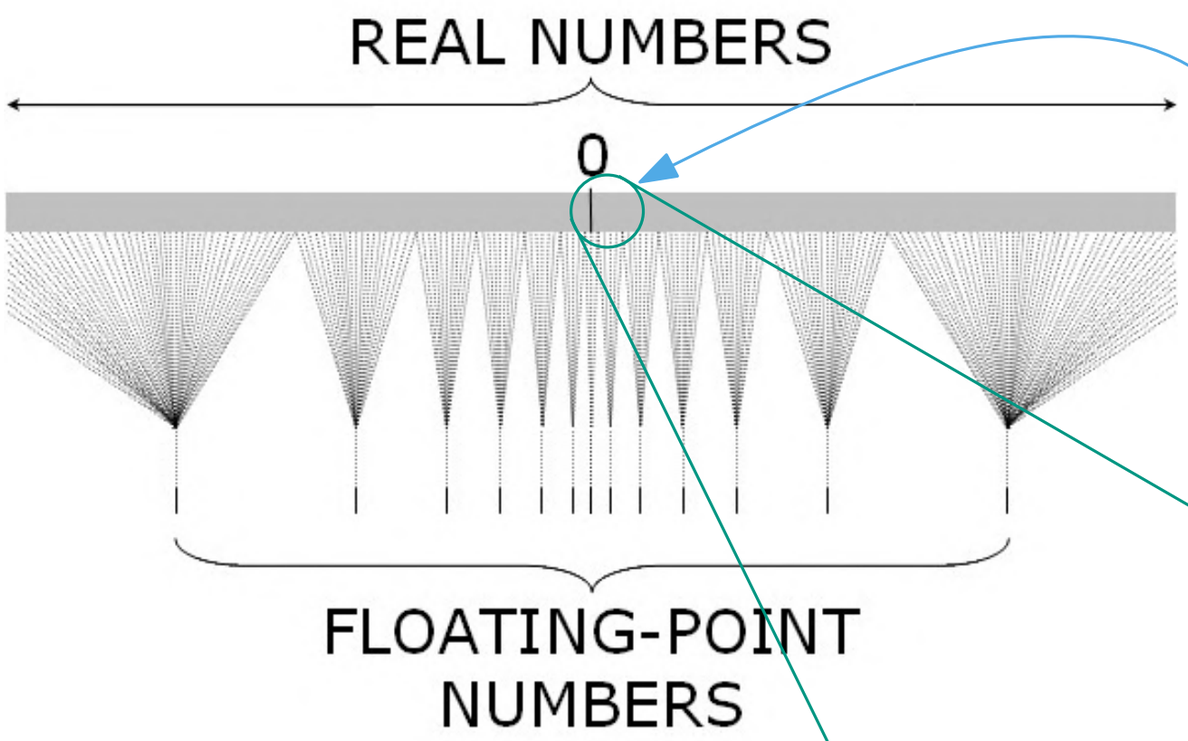
Floating-Point Math in Phylogenetics



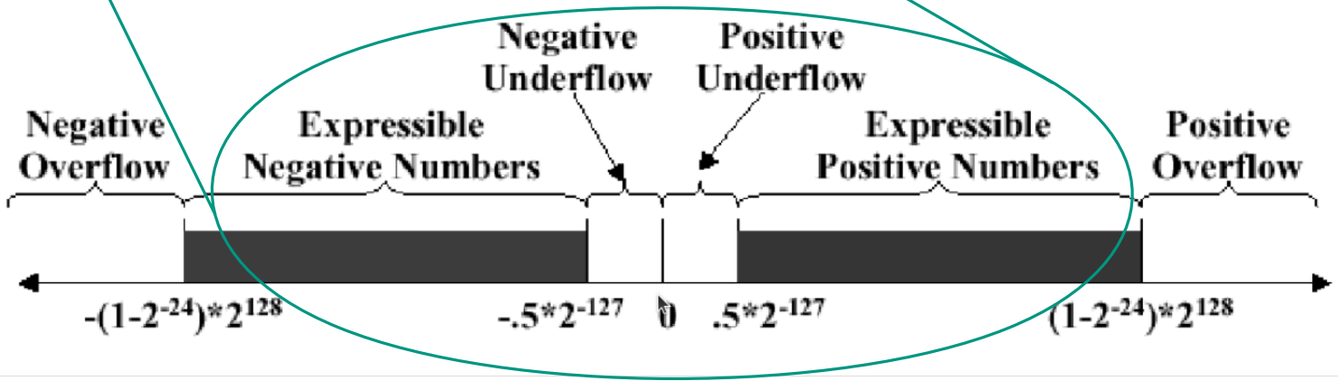
Floating-Point Math in Phylogenetics



Floating-Point Math in Phylogenetics



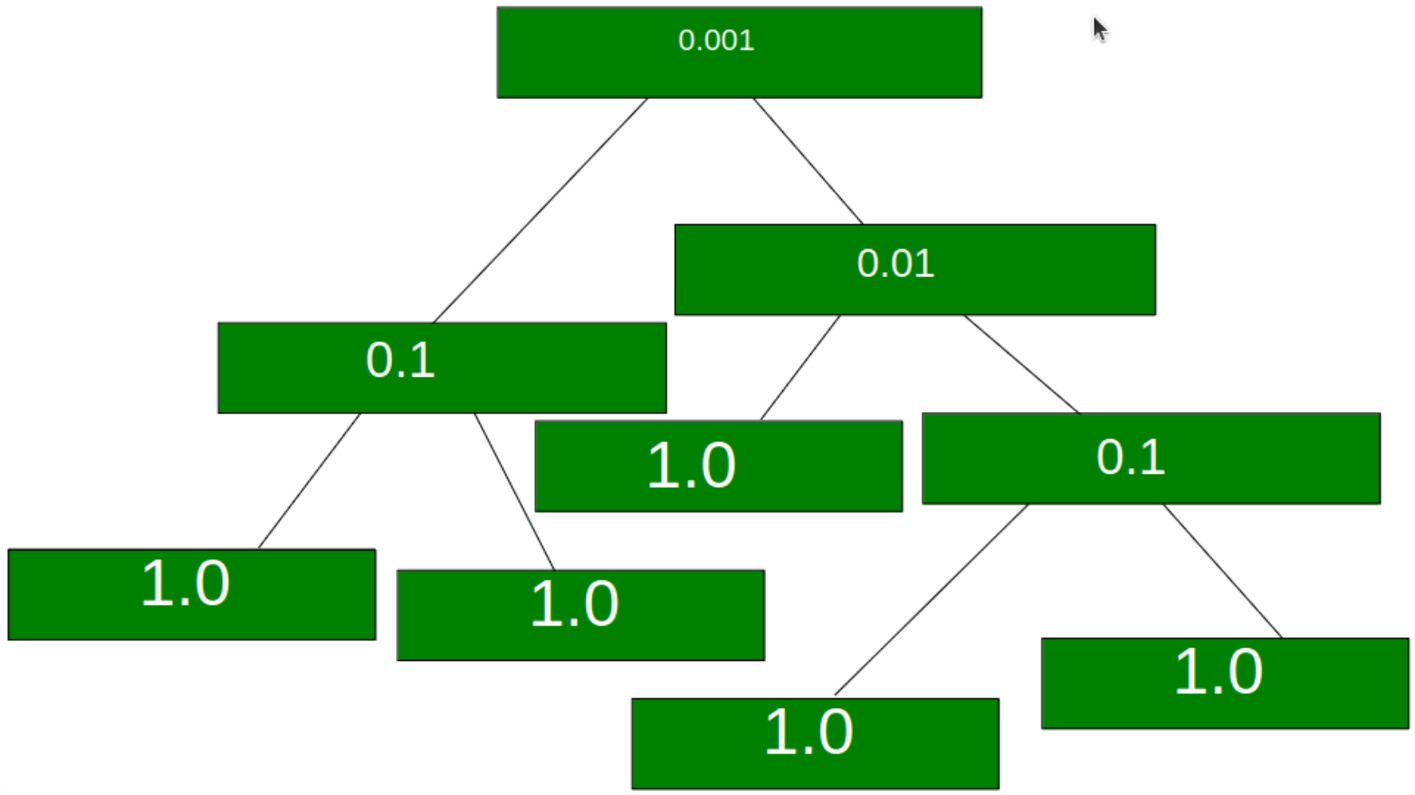
Values too small → underflow (round to zero)



How to Avoid Underflow?

Typical approach

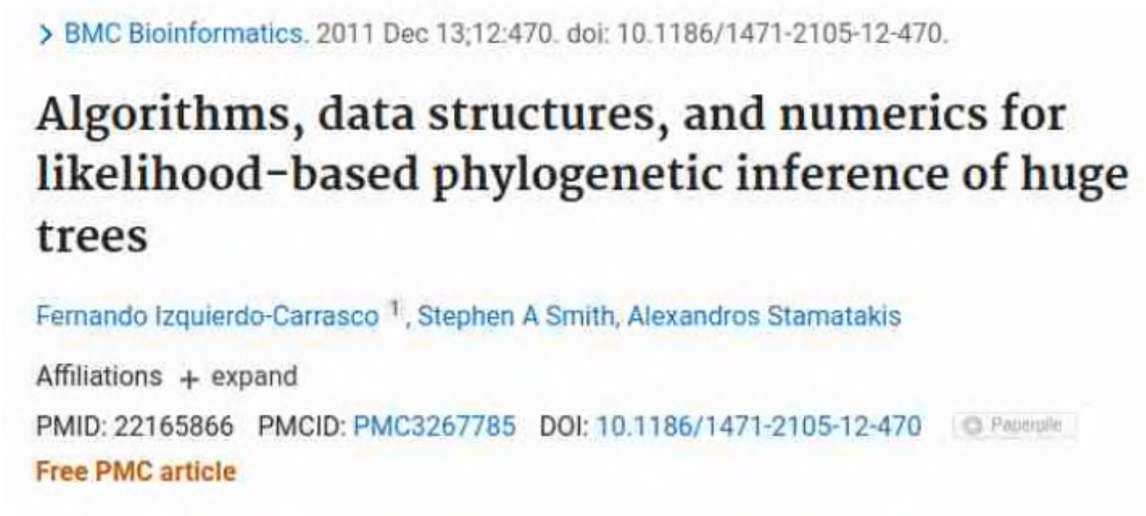
1) Check if values are too small
2) If so multiply with some large number
3) Undo those scaling multiplications (somehow) in the end
4) for likelihood this undoing is easy



Values in conditional likelihood vectors get smaller and smaller as we move to the root → this needs to be handled!

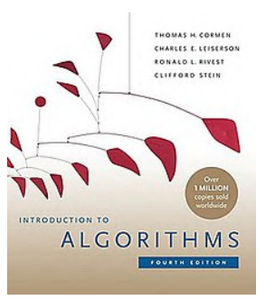
What Went Wrong?

- DNA models without rate heterogeneity: Scaling approach worked fine
  $\rightarrow$ check if all 4 conditional likelihoods at a given CLV and site are smaller than a minimum and multiply with large number
- DNA models with rate heterogeneity: doesn't always work
  - Range of values is too large because of distinct rate categories
  - Solution: Scale rate categories individually
  - Slower :-/



Computational Reproducibility

For reproducible results, fix



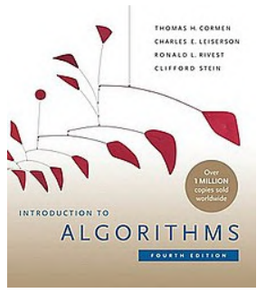
algorithm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=us-ascii" />
        <script type="text/javascript">
            function reDo() {top.location.reload();}
            if (navigator.appName == 'Netscape') {top.onresize = reDo;}
            dom=document.getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

source code

Computational Reproducibility

For reproducible results, fix



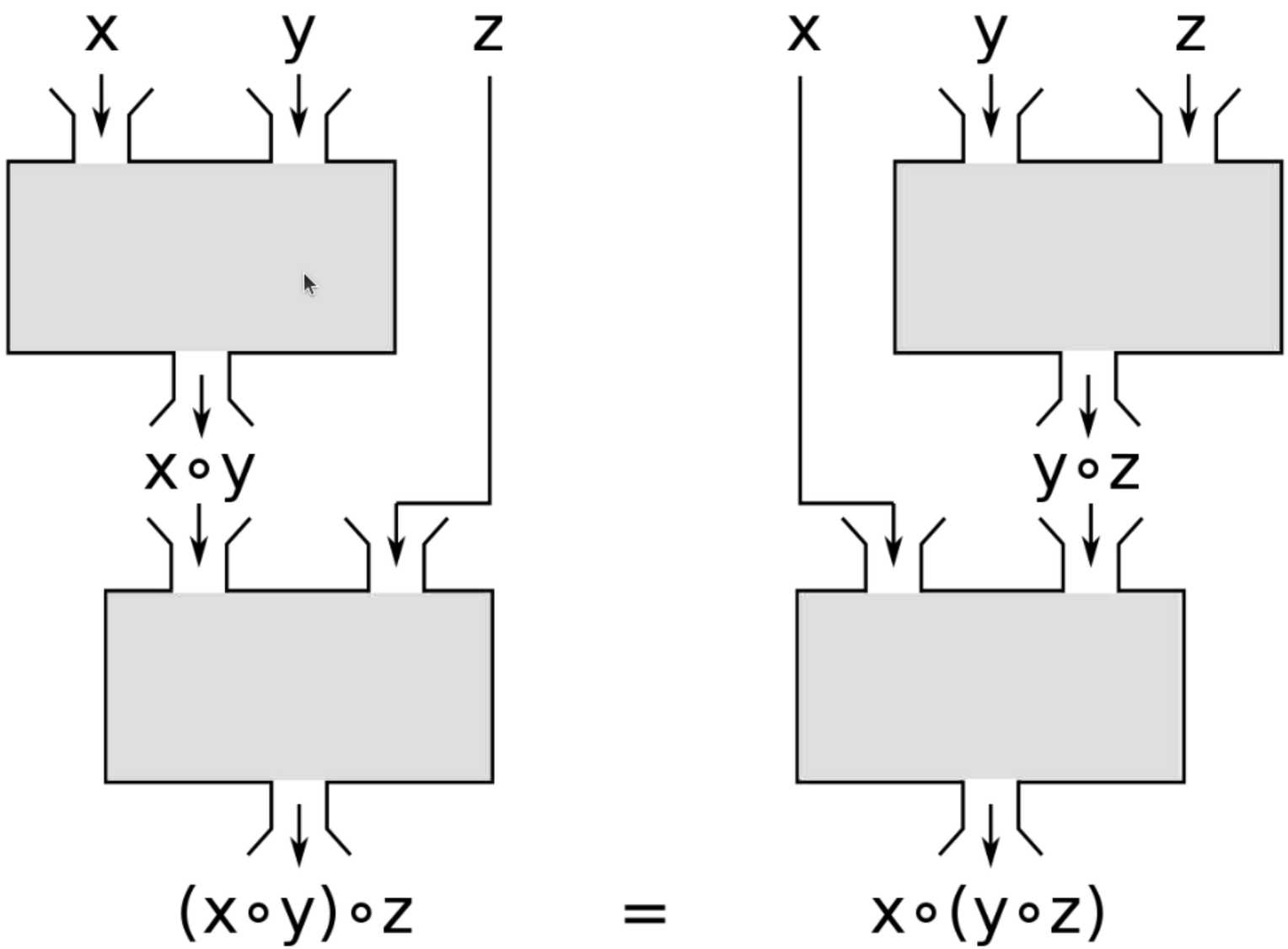
algorithm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
xhtml">
    <head>
        <meta http-equiv="Content-
Type" content=
        "text/html; charset=us-
ascii" />
        <script type="text/
javascript">
            function reDo() {top.
location.reload();}
            if (navigator.appName ==
'Netscape') {top.onresize = reDo;}
            dom=document.
getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

source code

binary

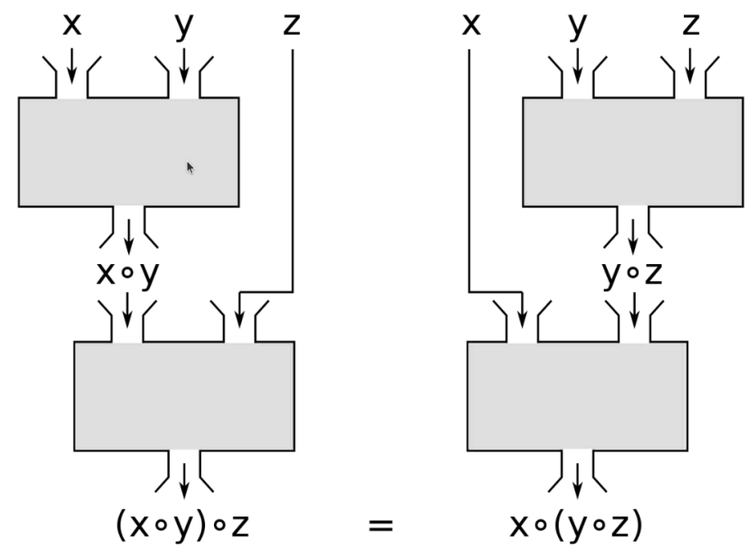
Floating-Point Math is Non-Associative



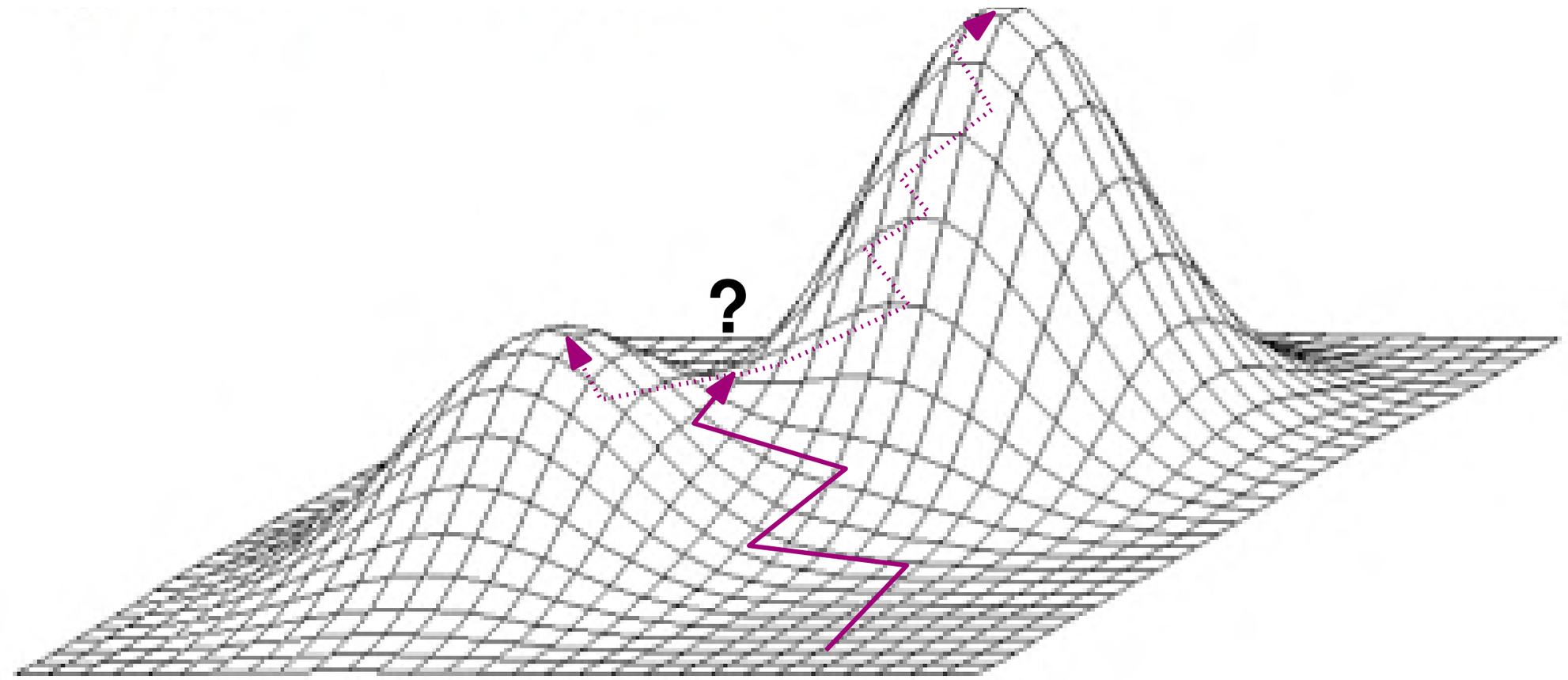
- $(a + b) + c \neq a + (b + c)$
- different round-off errors

Floating-Point Math is Non-Associative

- We reorder code
  - Manually
  - Compiler optimizes code
  - CPUs reorder instructions
  - Number of threads/processes change reduction algorithm
- We round values at different points in time
  - 80-bit floating-point registers get flushed to 64-bit memory or regular registers
  - Different SIMD instruction sets



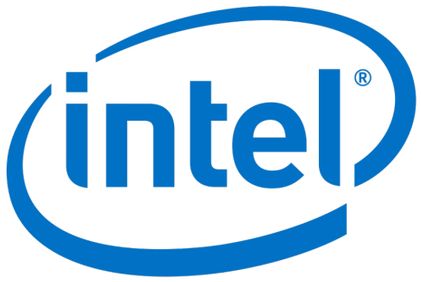
Floating-Point Math is Non-Associative



Small differences in the Likelihood computation can lead the tree searches to diverge.

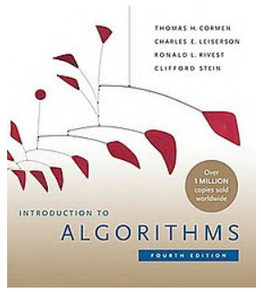
Compiler Optimizations

- In RAxML uses matrix exponential function from the book Numerical Recipes in C
- Especially the Intel `icc` compiler was very aggressive when optimizing this function
  $\rightarrow$ numerical breakdown
- Solution: Disable compiler optimization for respective subroutine



Computational Reproducibility

For reproducible results, fix



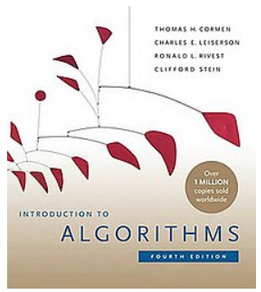
algorithm

source code

binary

Computational Reproducibility

For reproducible results, fix



algorithm

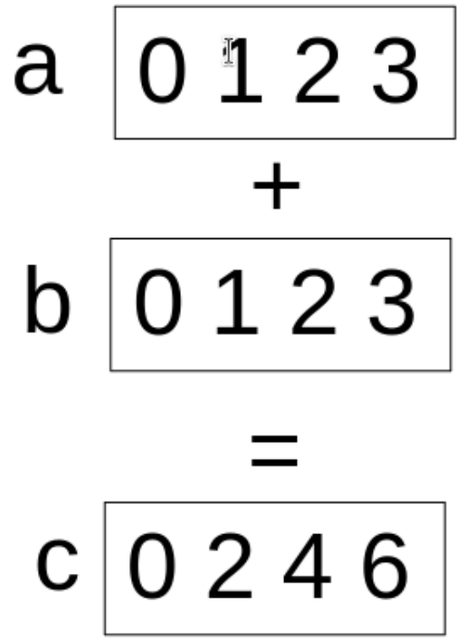
source code

binary

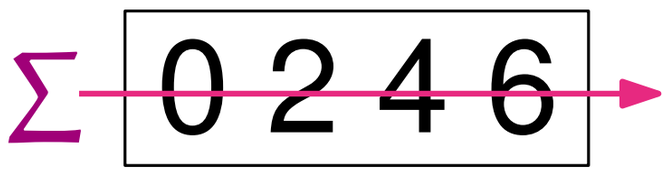
hardware

SIMD Instructions

- Single Instruction Multiple Data
- Parallelism inside a single CPU
- Different versions: SSE3, AVX, AVX2

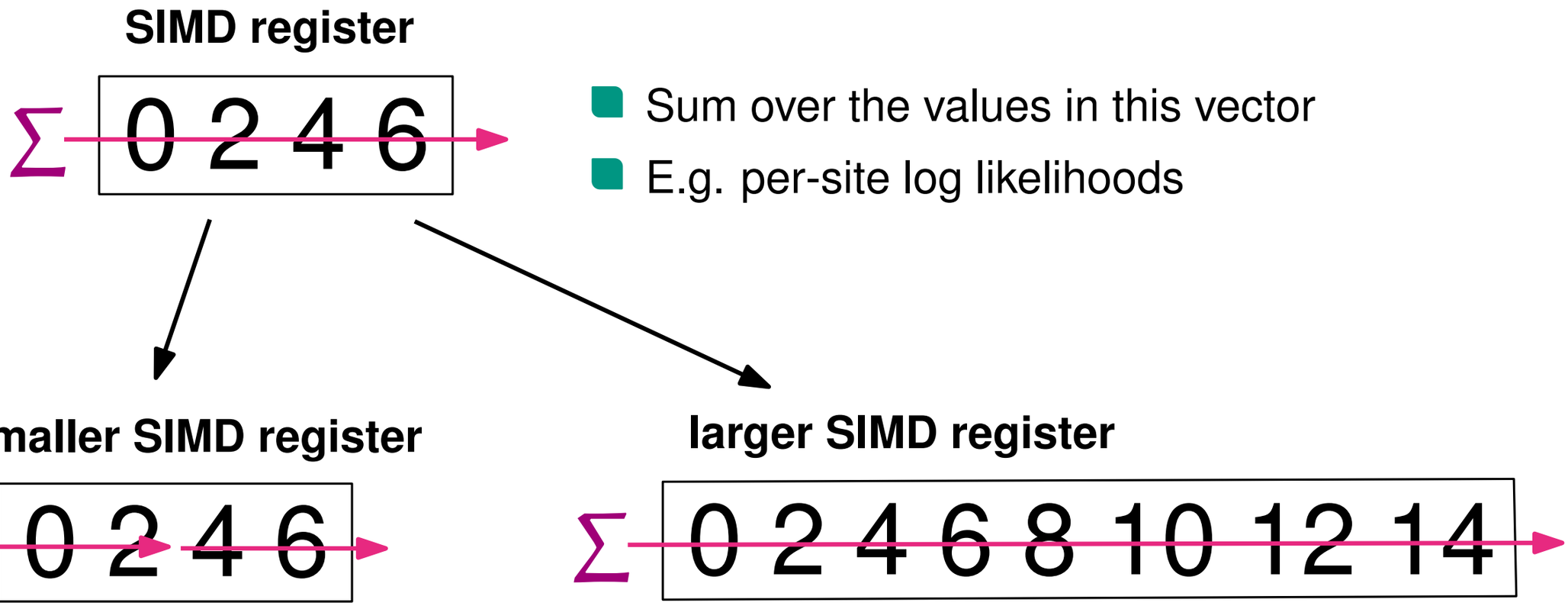


SIMD Instructions: Horizontal Add



- Sum over the values in this vector
- E.g. per-site log likelihoods

SIMD Instructions: Horizontal Add

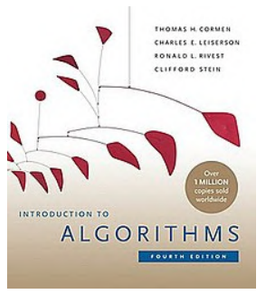


- Sum over the values in this vector
- E.g. per-site log likelihoods

- Different SIMD versions have different register widths
- Leads to different round-off errors
- Effects even visible on easy datasets

Computational Reproducibility

For reproducible results, fix



algorithm

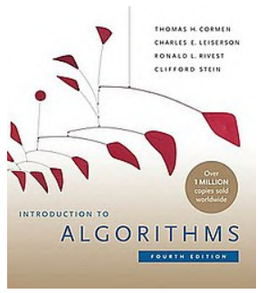
source code

binary

hardware

Computational Reproducibility

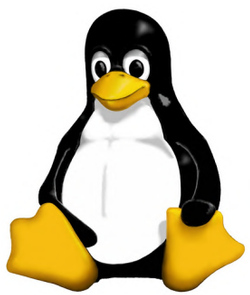
For reproducible results, fix



algorithm

source code

binary

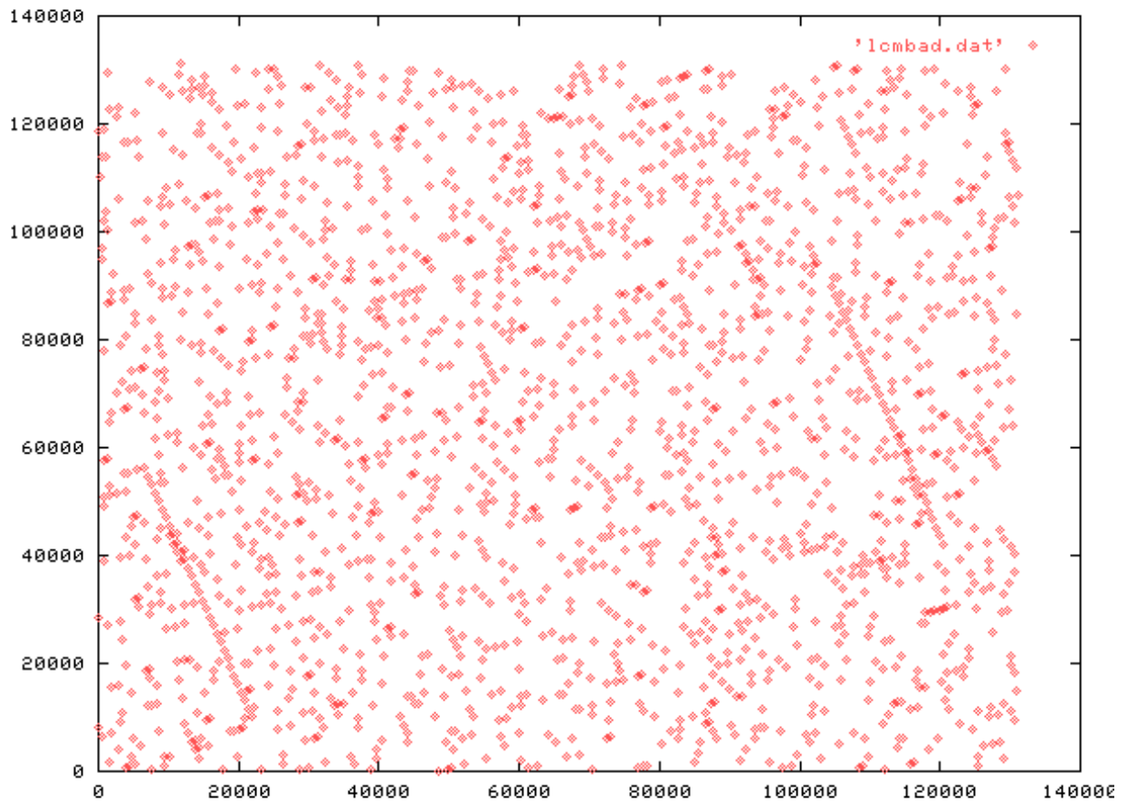


software environment

hardware

Pseudorandom Number Generators

- Example: Linear Congruential Generator $X_{n+1} = (a \cdot X_n) + c \mod m$
- Generates sequence of number deterministically
- $\rightarrow$ Same seed leads to same sequence of numbers
- Sequence “looks” random



Pseudorandom Number Generators

A Critical Assessment of Storytelling: Gene Ontology Categories and the Importance of Validating Genomic Scans

Pavlos Pavlidis,*[1] Jeffrey D. Jensen,[2] Wolfgang Stephan,[3] and Alexandros Stamatakis[1]

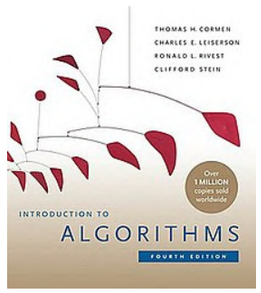
- Results not reproducible on different machine; even with same random-number generator seed

→ **different Random Number Generator library lead to different sequence of numbers**

- Take-home message: Version control for external libraries, too.

Computational Reproducibility

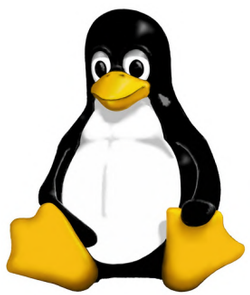
For reproducible results, fix



algorithm

source code

binary

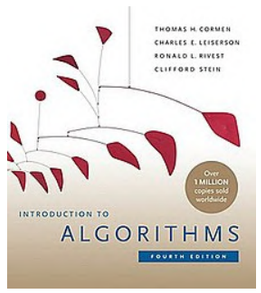


software environment

hardware

Computational Reproducibility

For reproducible results, fix

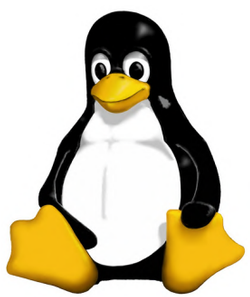


algorithm

source code

binary

That's it, right?

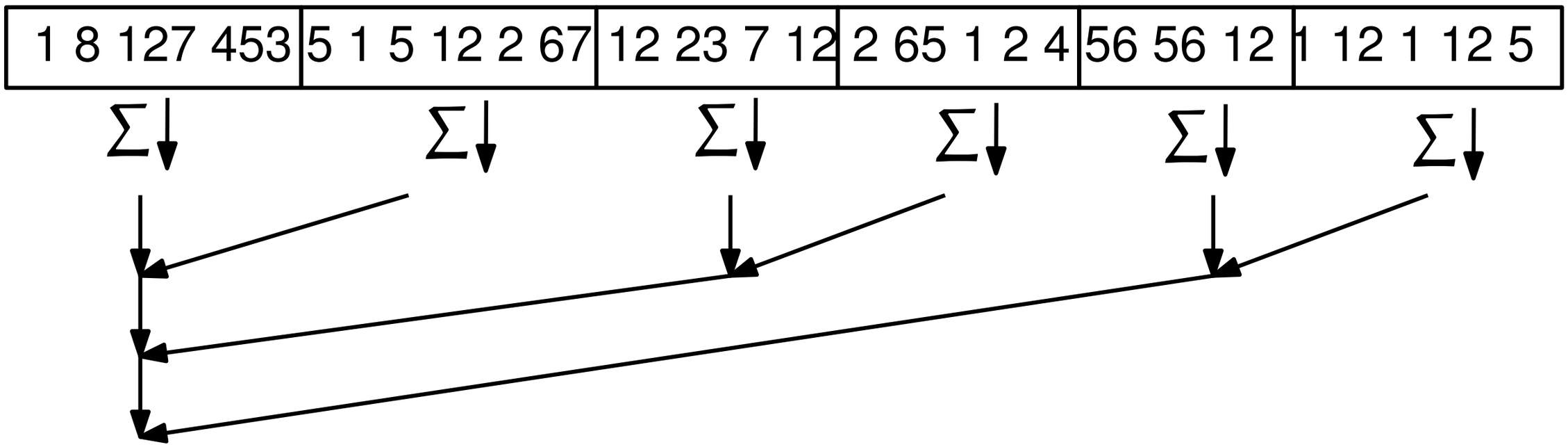


software environment

hardware

Multi-Thread and Multi-Processor Reduce

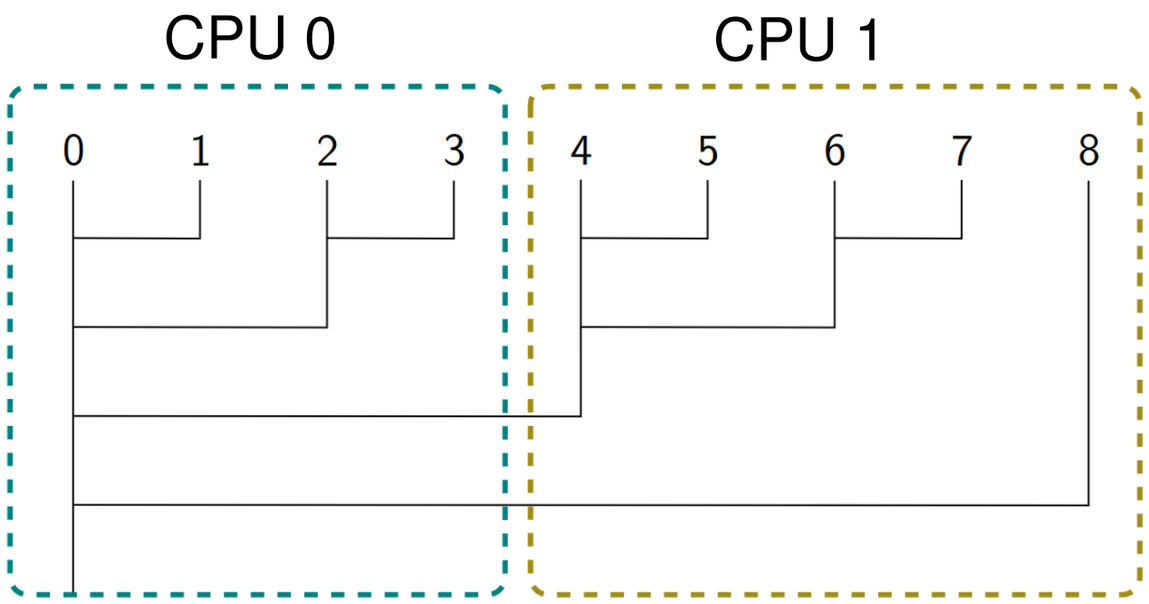
- Running the same binary on the same cluster with different number of processes can lead to different results
- We observed this on real data
- We have to sum numbers stored across multiple CPUs

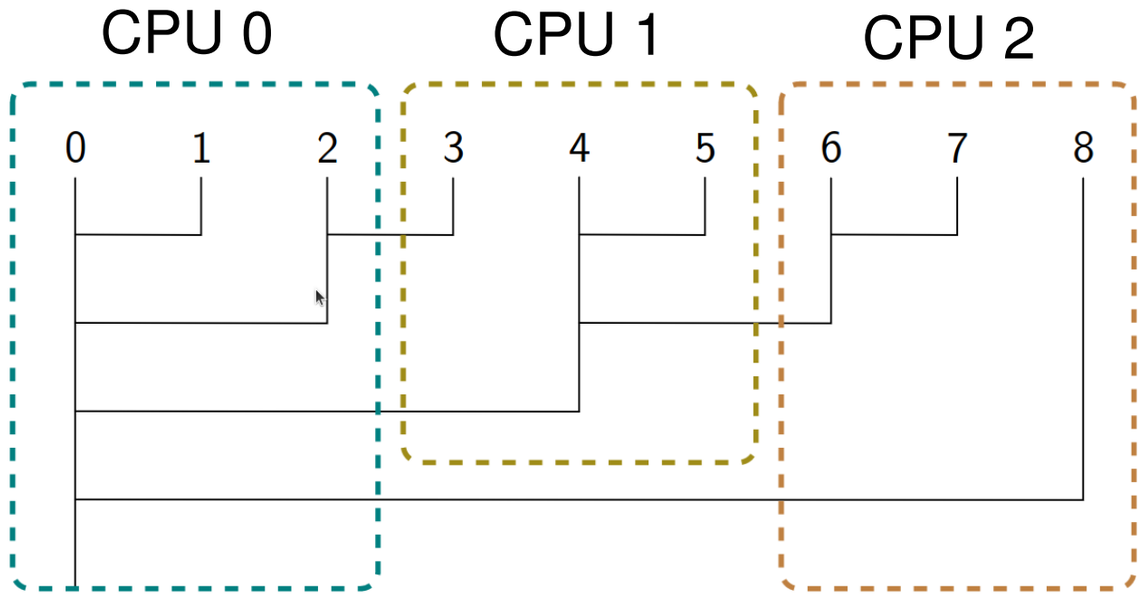


→ **different number of CPUs results in different round-off errors**

Multi-Thread and Multi-Processor Reduce

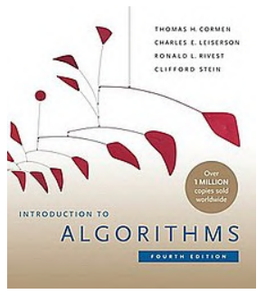
- Idea: Do local summation as a tree, too. Send intermediate results over network
- Same order of summation $\rightarrow$ same round-off error
- Slower than local summation + reduce
- We verified reproducibility under different number of CPUs with RAxML-NG





Computational Reproducibility

For reproducible results, fix



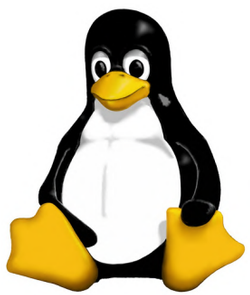
algorithm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/
xhtml">
    <head>
        <meta http-equiv="Content-
Type" content=
        "text/html; charset=us-
ascii" />
        <script type="text/
javascript">
            function reDo() {top.
location.reload();}
            if (navigator.appName ==
'Netscape') {top.onresize = reDo;}
            dom=document.
getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

source code

binary

Bonus: Measurement of Runtime



software environment

hardware

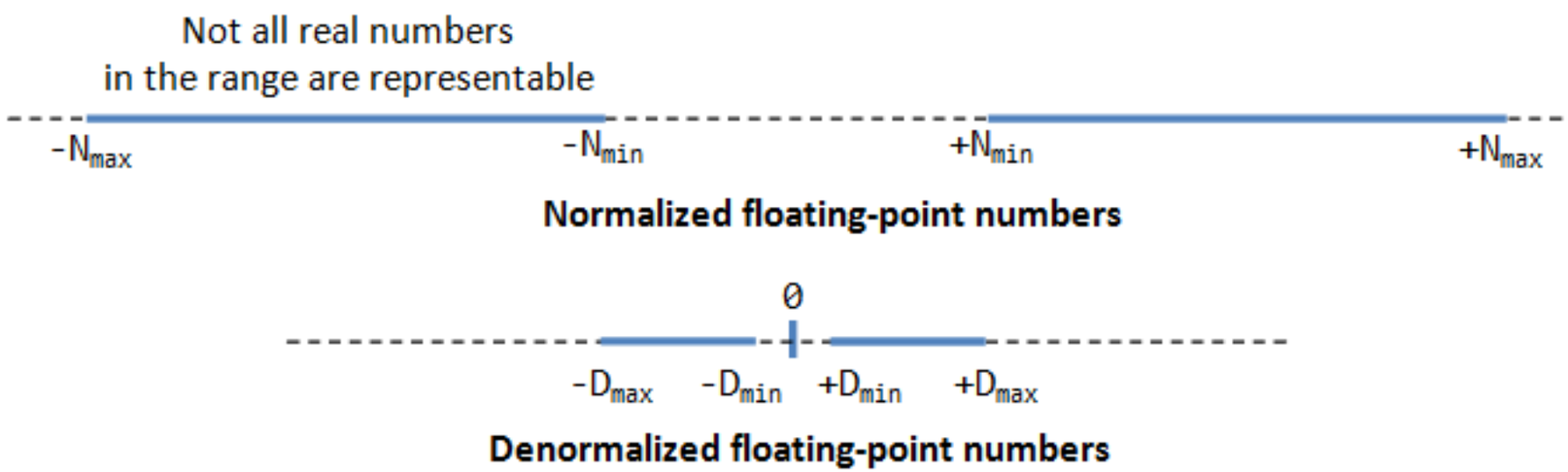
Strange Runtime-Measurements

- Phylogenetic placement methods: Inexplicable run time deviations of about 50 %
- . . . with the same number and type of operations just on different input data?

Strange Runtime-Measurements

- Phylogenetic placement methods: Inexplicable run time deviations of about 50 %
- . . . with the same number and type of operations just on different input data?

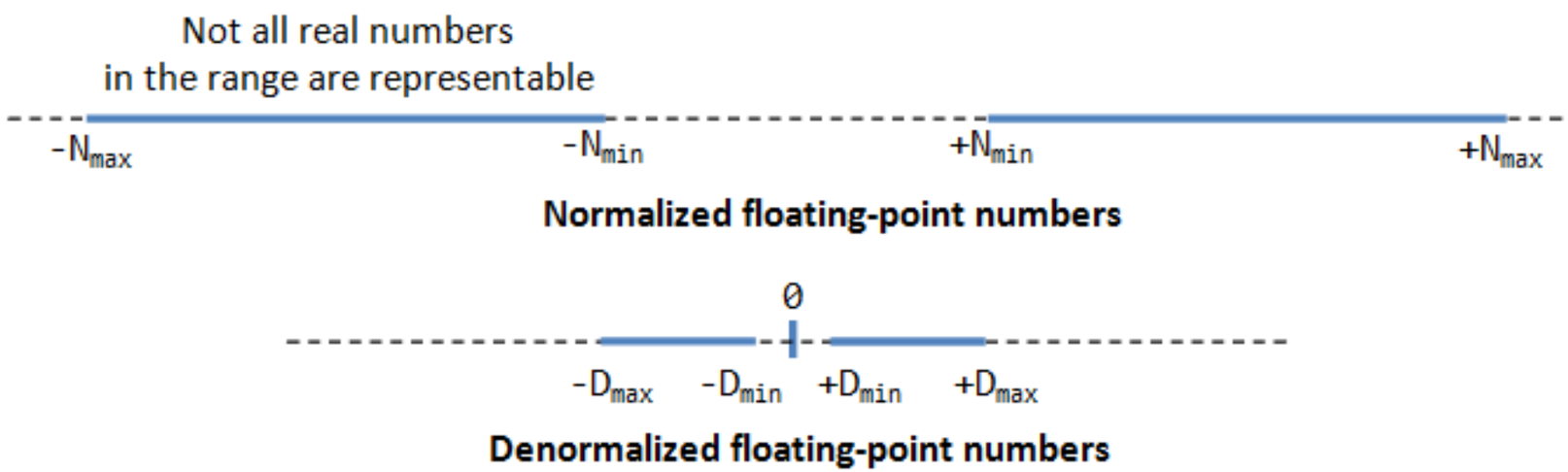
De-normalized floating-point number



Strange Runtime-Measurements

- Phylogenetic placement methods: Inexplicable run time deviations of about 50 %
- . . . with the same number and type of operations just on different input data?

De-normalized floating-point number



- Values too close to zero → slower multiplications and additions
- Benchmarks data dependent

Conclusion

- Node-failures in supercomputers will get more frequent
- We need to be able to handle node failures (FT-RAxML-NG, ReStore)
- Compiler optimization, SIMD version, and number of CPUs hinder reproducibility
- Version control is essential (RNG)