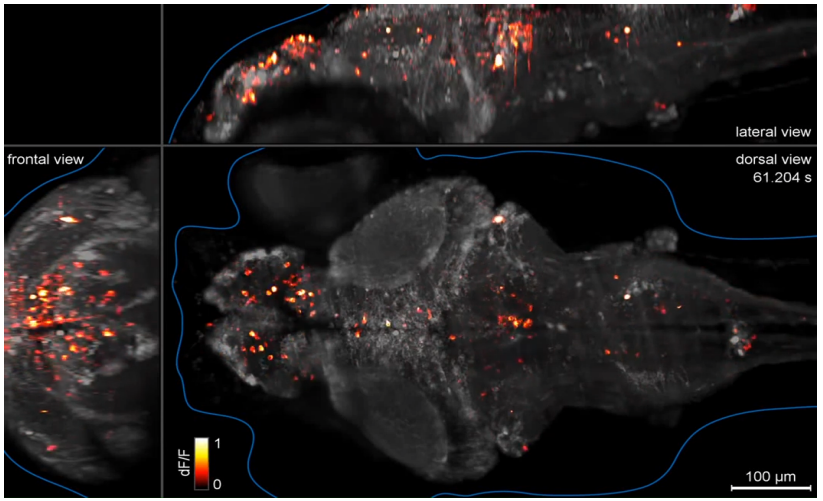
Event-based Simulation of Spiking Neural Networks in Julia

Rainer Engelken

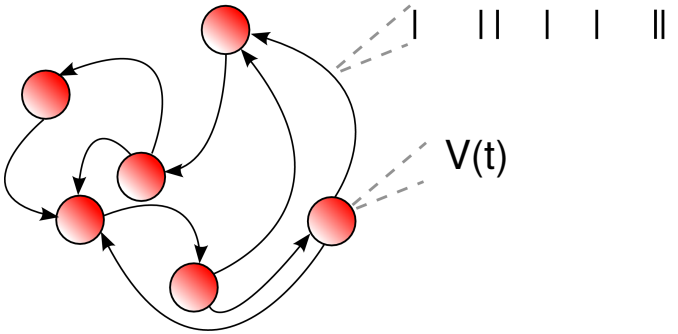
Max Planck Institute for Dynamics and Self-Organization, Göttingen
Bernstein Center for Computational Neuroscience, Göttingen
Georg-August-Universität Göttingen

Collective dynamics of neural circuits

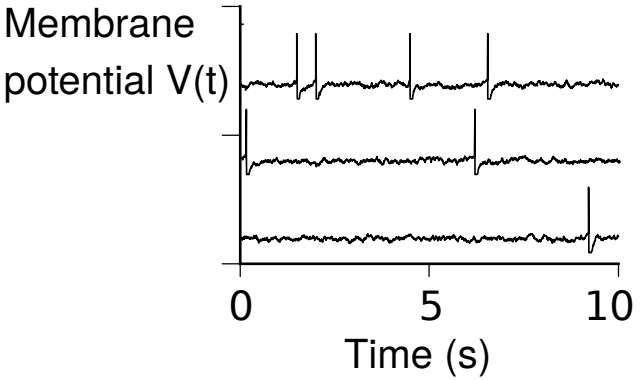
Population activity recording:



[Ahrens et al. 2013]



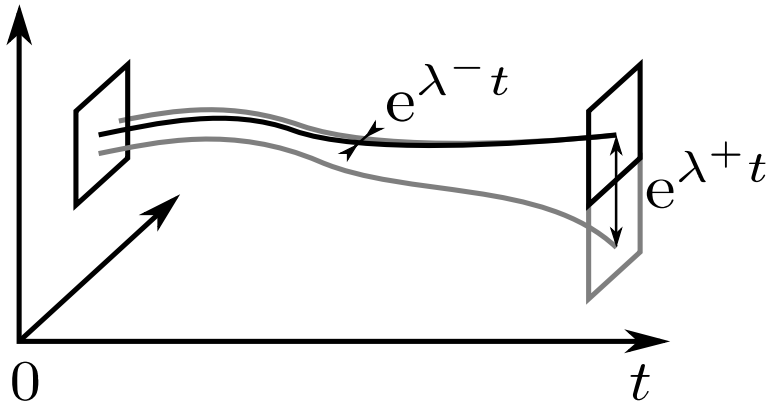
Spiking network



$$\tau_{\mathrm{m}}\dot{V}_i = F(V_i) + I_i(t)$$

Collective chaotic dynamics

Dynamical systems: stability and chaos



Stable dynamics:

- Small perturbations decay exponentially ($\lambda^- < 0$)

Chaotic dynamics:

- Small perturbations grow exponentially ($\lambda^+ > 0$)
- Entropy production rate H (Uncertainty amplification)
- $H = \sum_{\lambda_i > 0} \lambda_i$

Event-based simulation of spiking networks

$$\tau_{\mathrm{m}} \frac{\mathrm{d}V_i}{\mathrm{d}t} = F\left(V_i\right) + I^{\mathrm{ext}} + \sum_{j,s} J_{ij}\, \delta\left(t - t_j^{(s)}\right)$$

- Exact solution of dynamics between network spikes at t_s and t_{s+1}

$$f\left(\vec{V}(t_s)\right) = \vec{V}(t_{s+1})$$

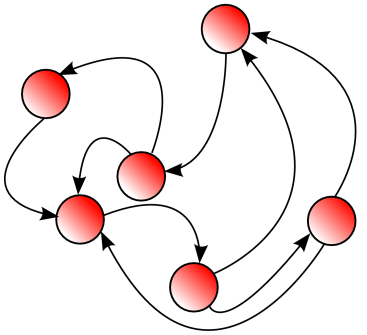
- Network simulation consists of four steps:

  1.) Finding next spiking neuron

  2.) Evolve all neurons to next network spike time

  3.) Update receiving neurons

  4.) Reset spiking neuron



Choosing a convenient representation

- Temporal evolution in voltage is nonlinear

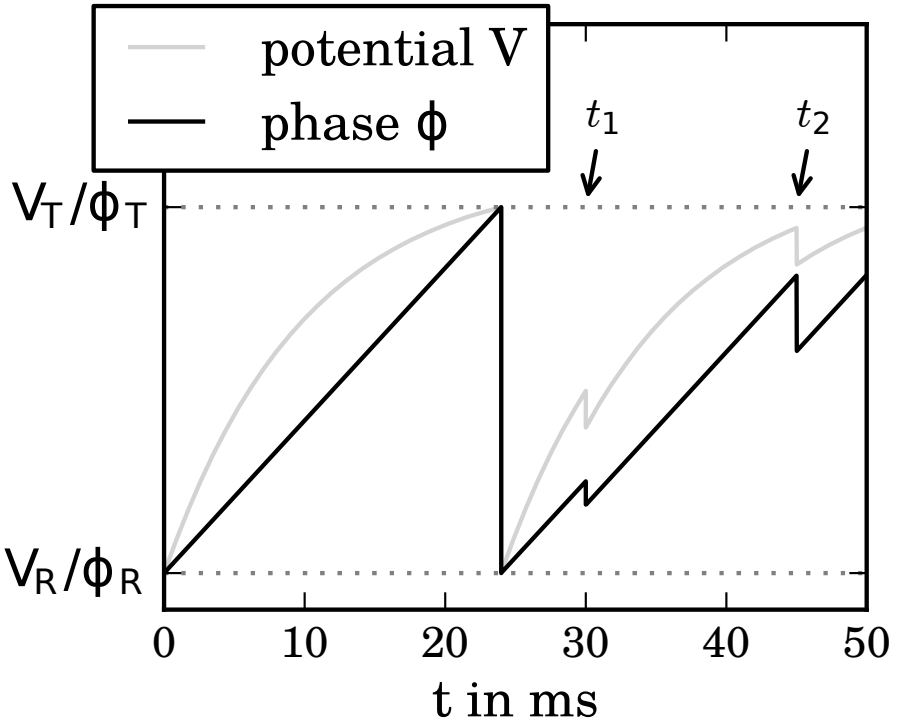


fig: Alexander Schmidt

- →change of variables to linear evolution
- →phase transition curve: phase changes by input spike

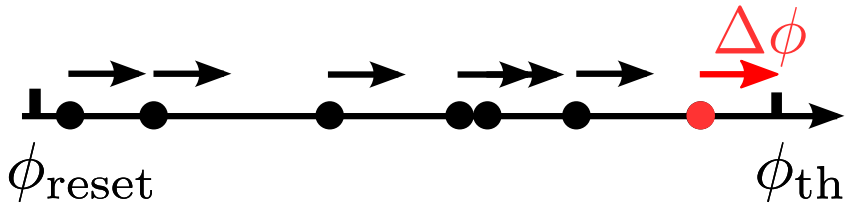
Toy example: Chaotic network of three neurons

See Jupyter notebook

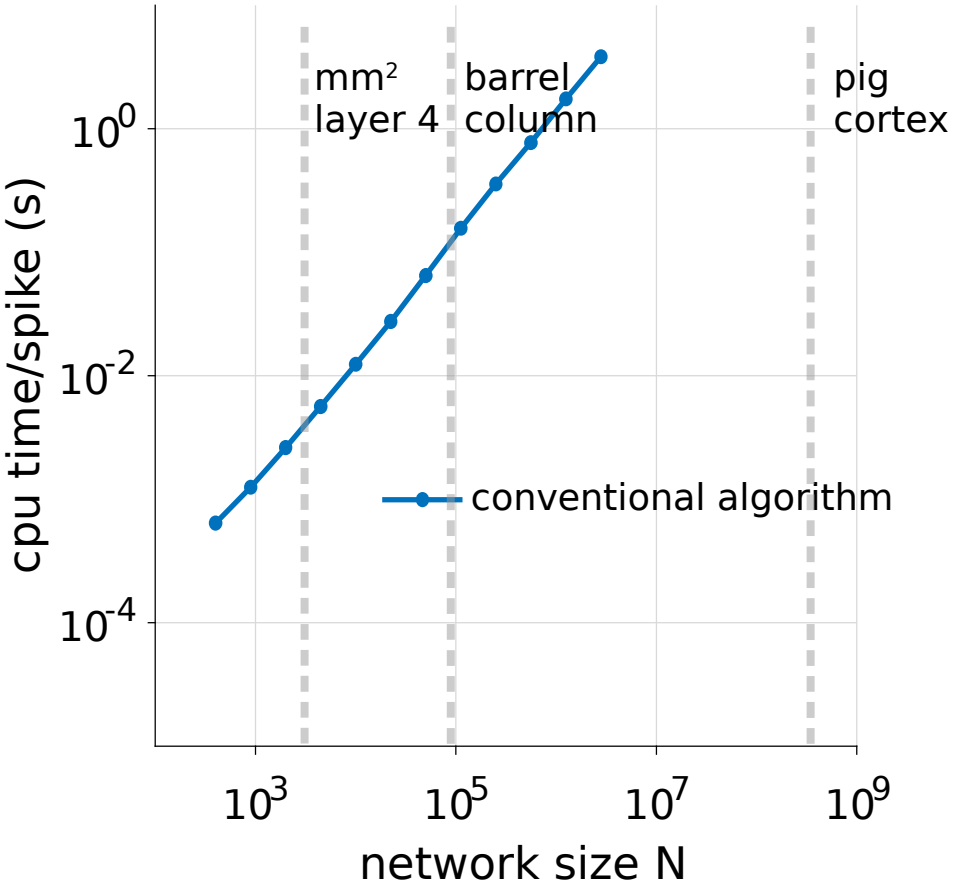
Novel algorithm for spiking network dynamics

Exact firing map: $f\left(\vec{\phi}(t_s)\right) = \vec{\phi}(t_{s+1})$

- **Conventional algorithm:**



→ time-complexity $\mathcal{O}\left(N\right)$

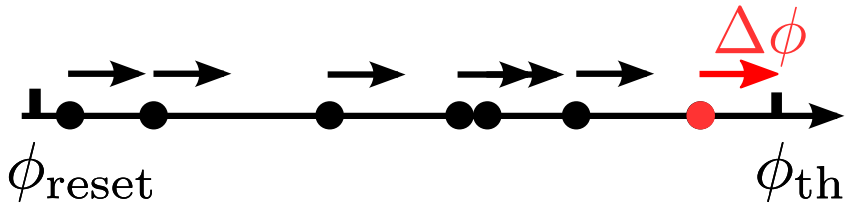


Intel(R) Xeon(R) CPU E5-4620 v2
@ 2.60GHz and 512 GB RAM

Novel algorithm for spiking network dynamics

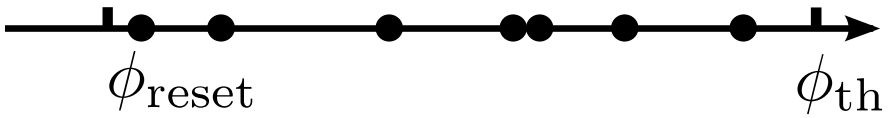
Exact firing map: $f\left(\vec{\phi}(t_s)\right) = \vec{\phi}(t_{s+1})$

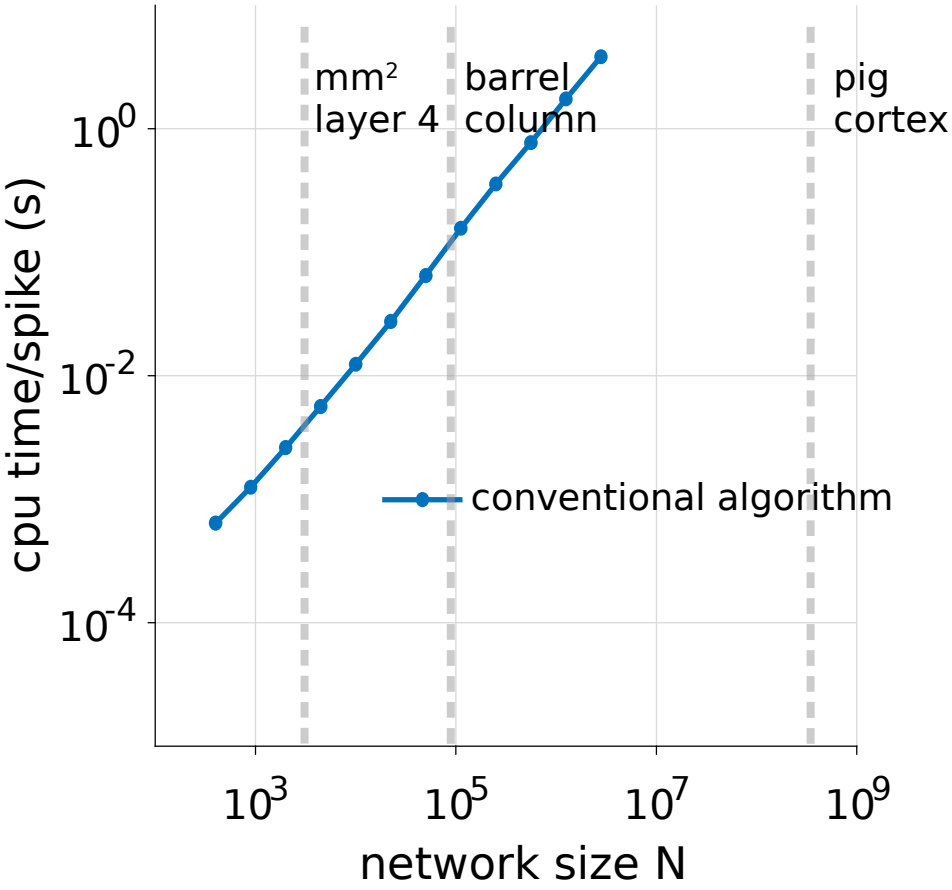
- **Conventional algorithm:**



→ time-complexity $\mathcal{O}\left(N\right)$

- **Novel algorithm:**



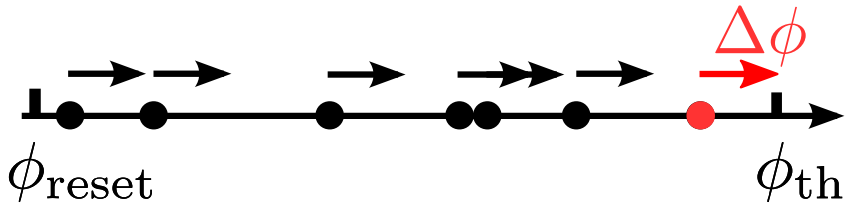


Intel(R) Xeon(R) CPU E5-4620 v2
@ 2.60GHz and 512 GB RAM

Novel algorithm for spiking network dynamics

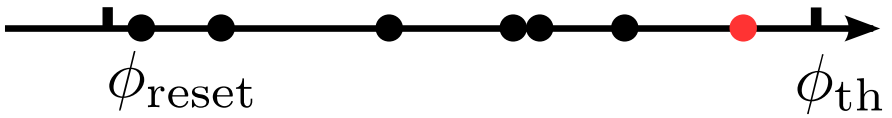
Exact firing map: $f\left(\vec{\phi}(t_s)\right) = \vec{\phi}(t_{s+1})$

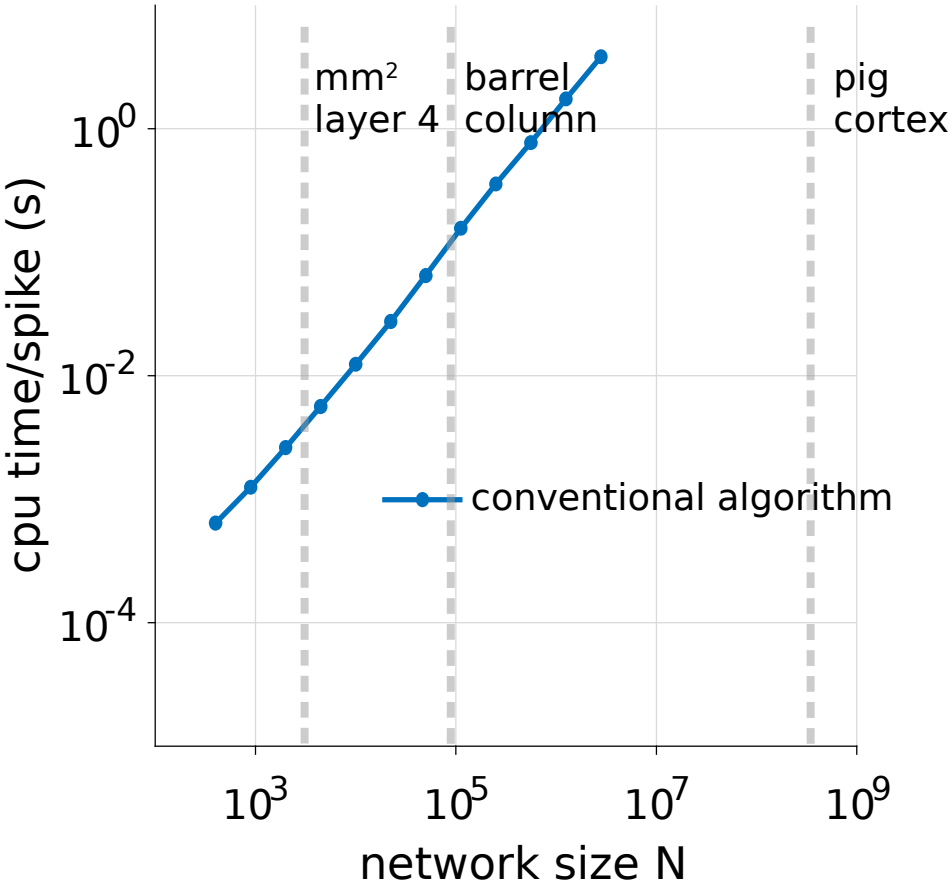
- **Conventional algorithm:**



→ time-complexity $\mathcal{O}\left(N\right)$

- **Novel algorithm:**



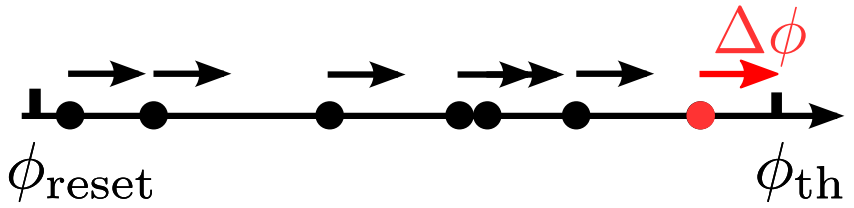


Intel(R) Xeon(R) CPU E5-4620 v2
@ 2.60GHz and 512 GB RAM

Novel algorithm for spiking network dynamics

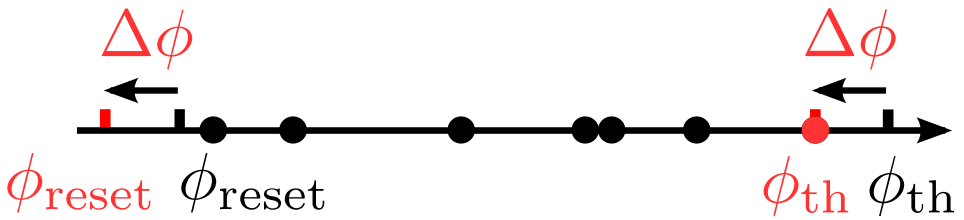
Exact firing map: $f\left(\vec{\phi}(t_s)\right) = \vec{\phi}(t_{s+1})$

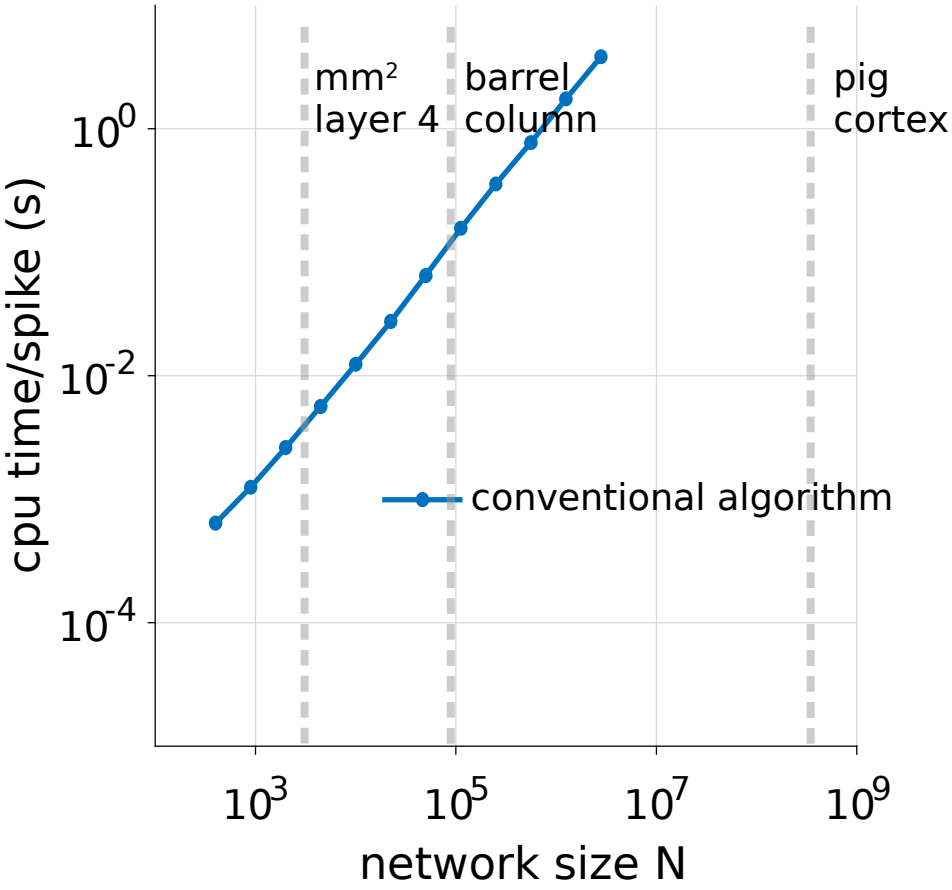
- **Conventional algorithm:**



→ time-complexity $\mathcal{O}\left(N\right)$

- **Novel algorithm:**





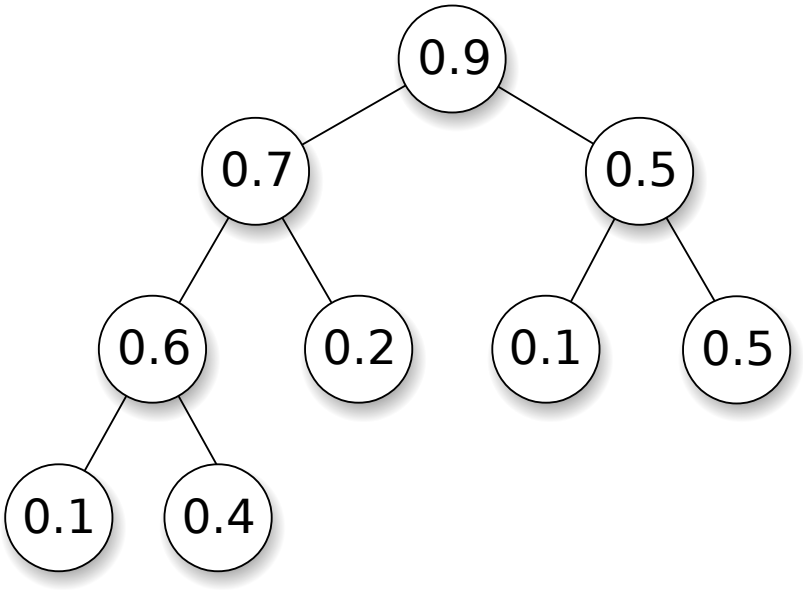
Intel(R) Xeon(R) CPU E5-4620 v2
@ 2.60GHz and 512 GB RAM

Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$

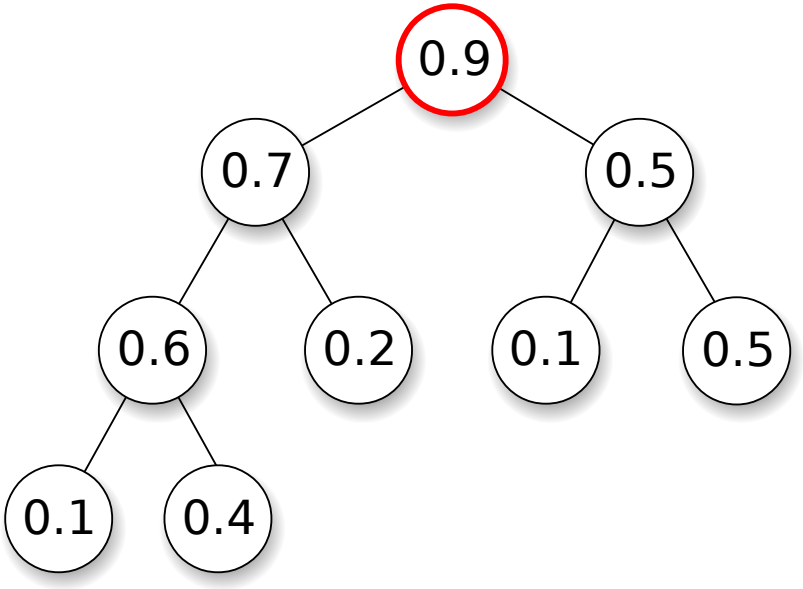
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



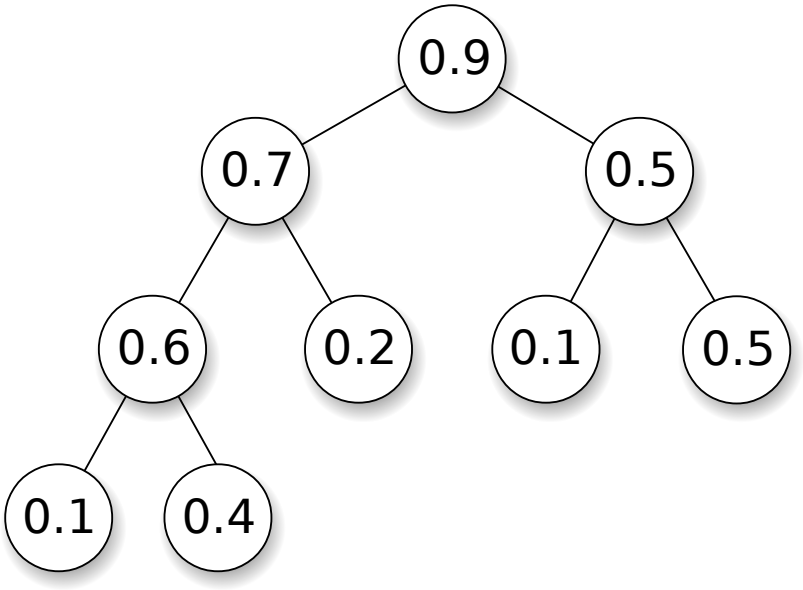
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



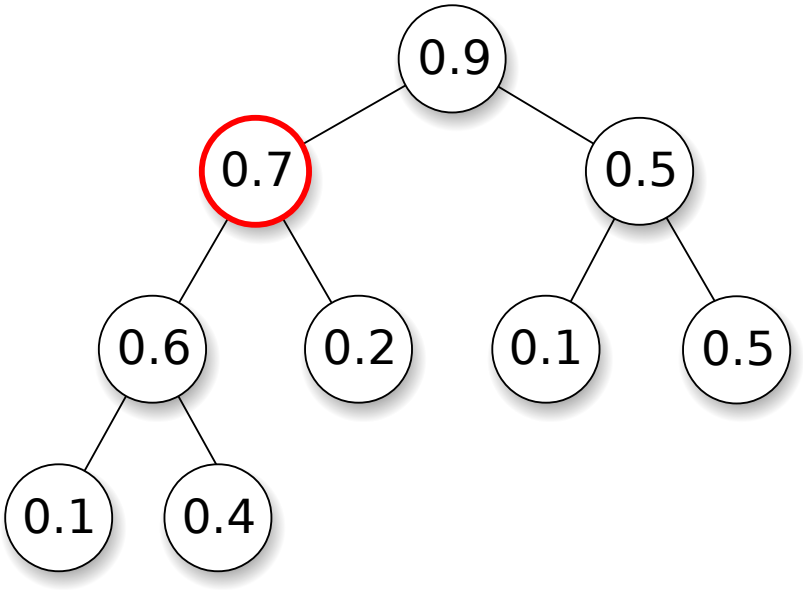
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



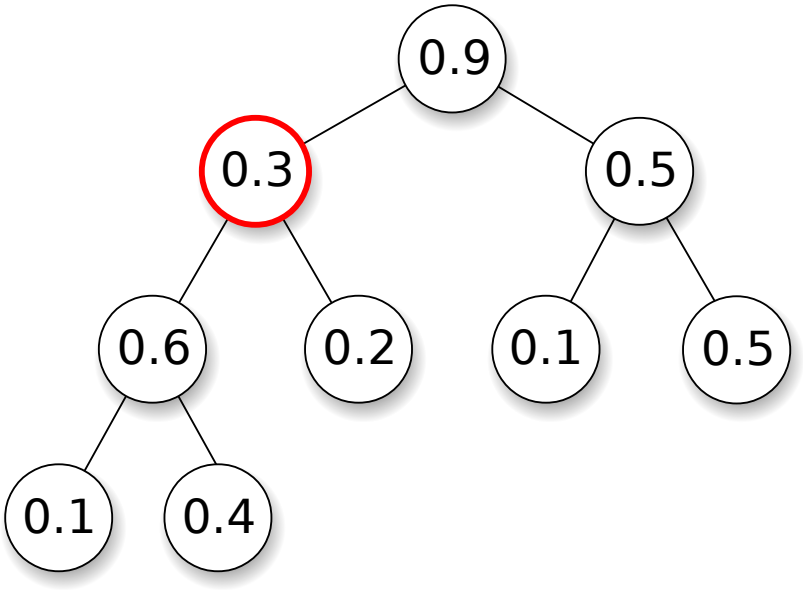
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



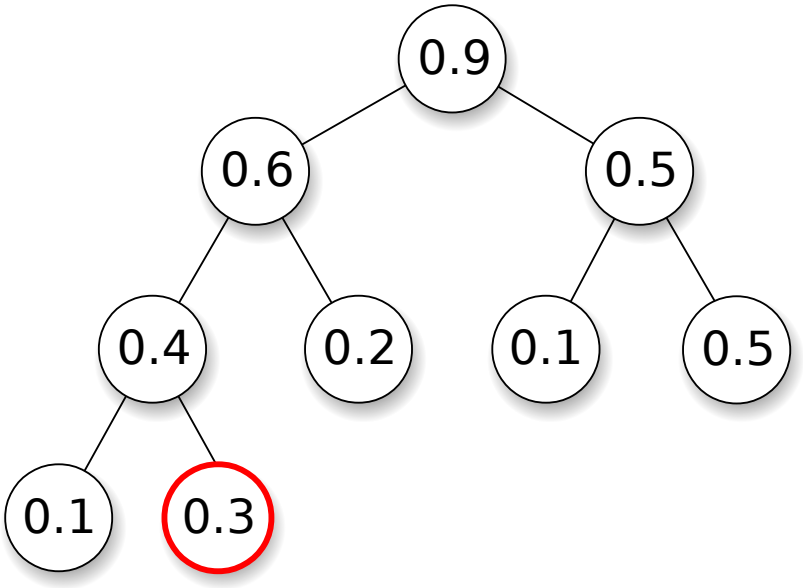
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



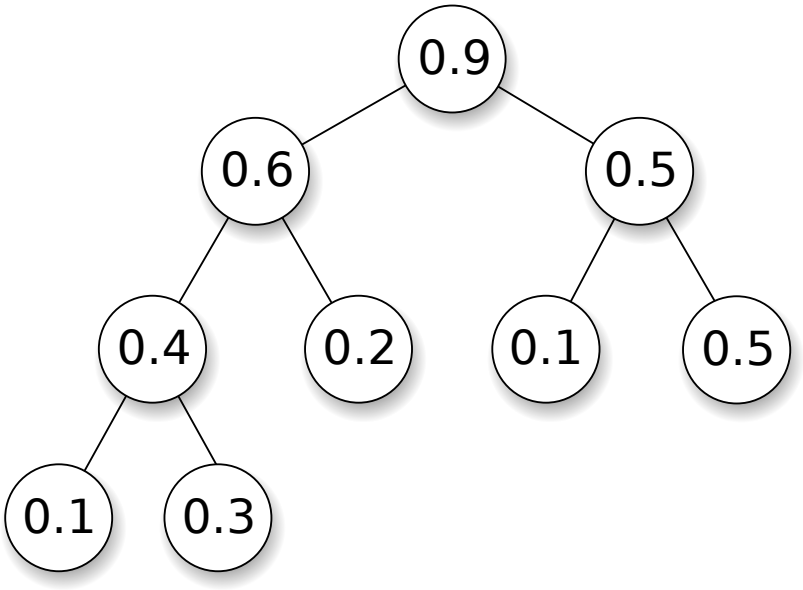
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



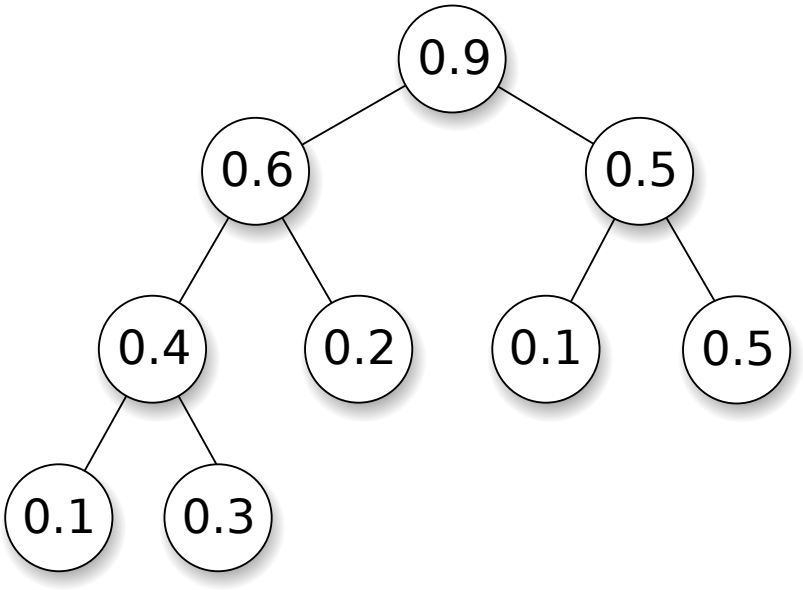
Find next spiking neuron and state update

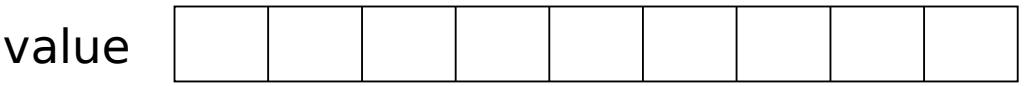
- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



Find next spiking neuron and state update

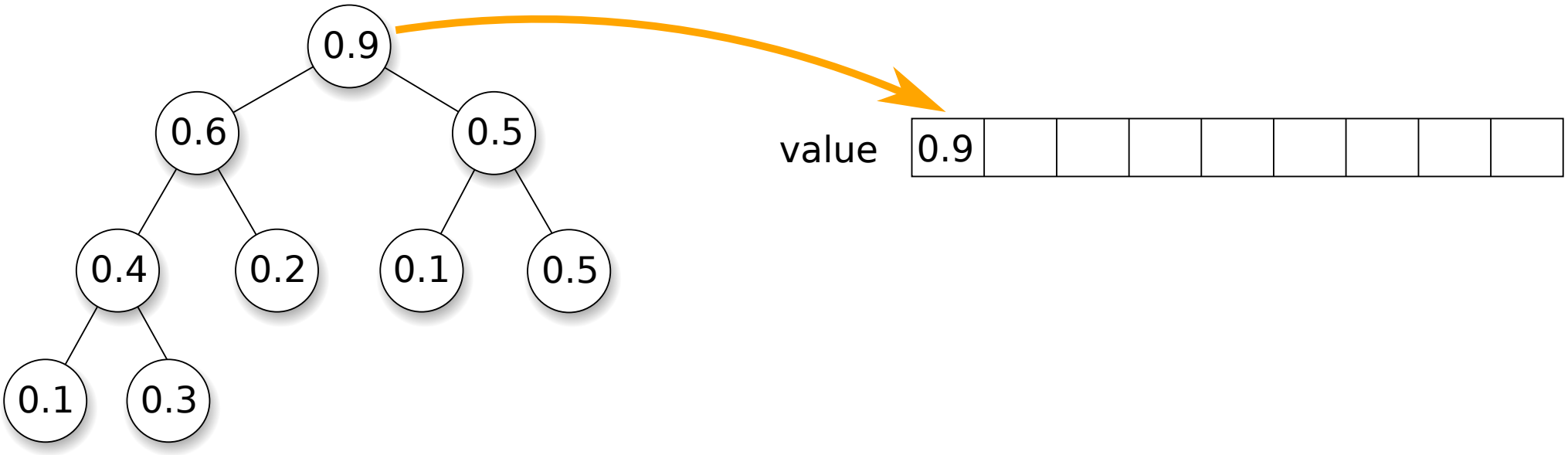
- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update





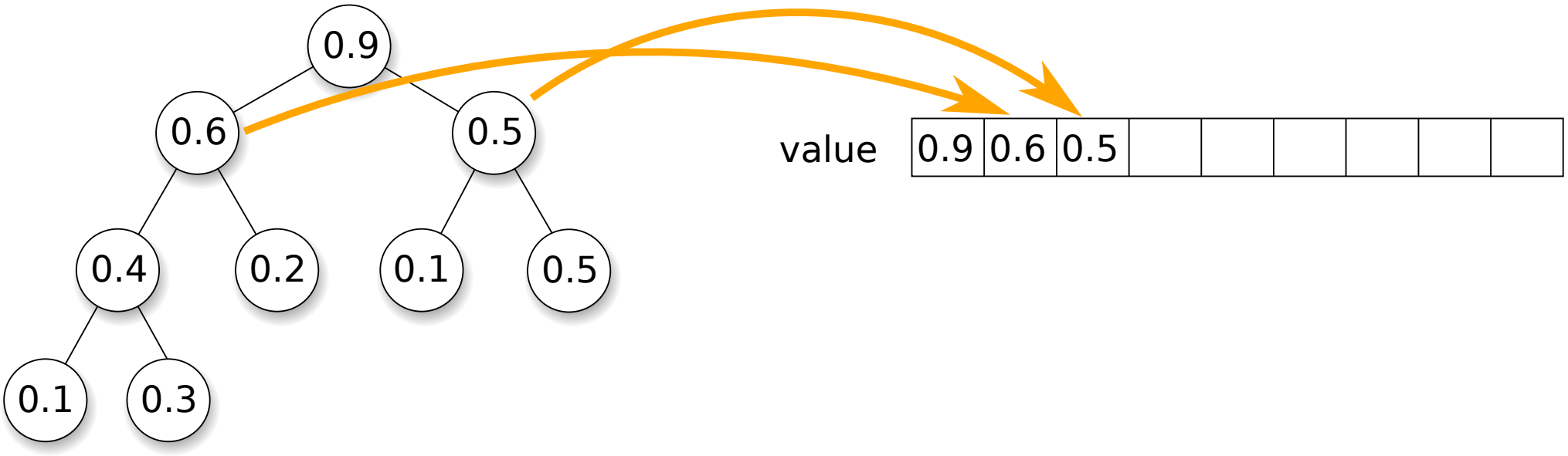
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



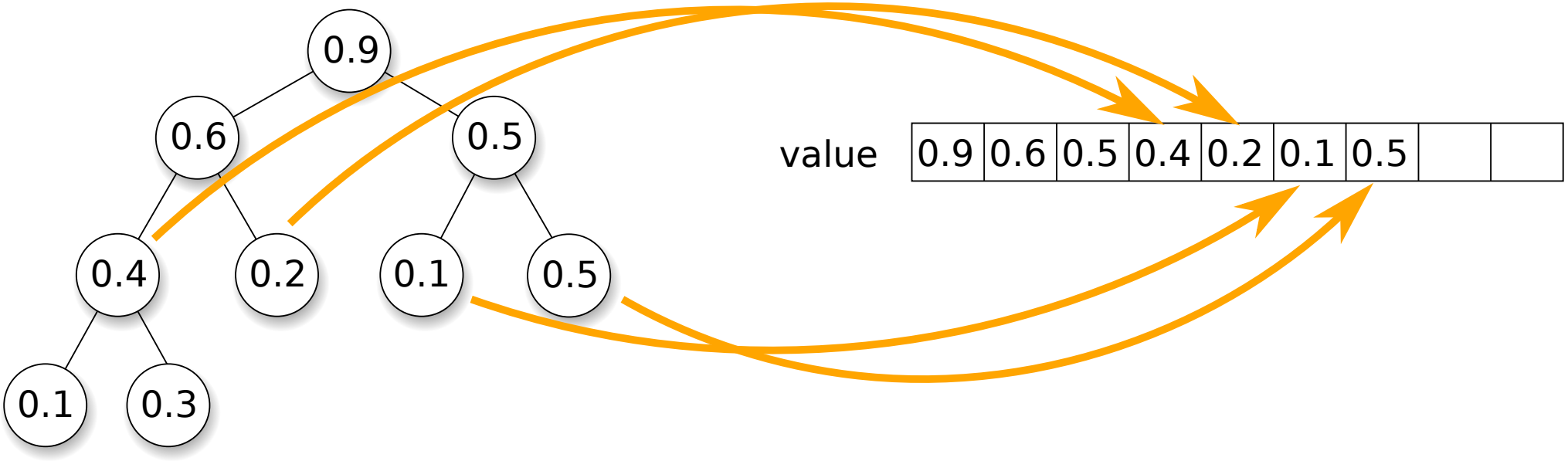
Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



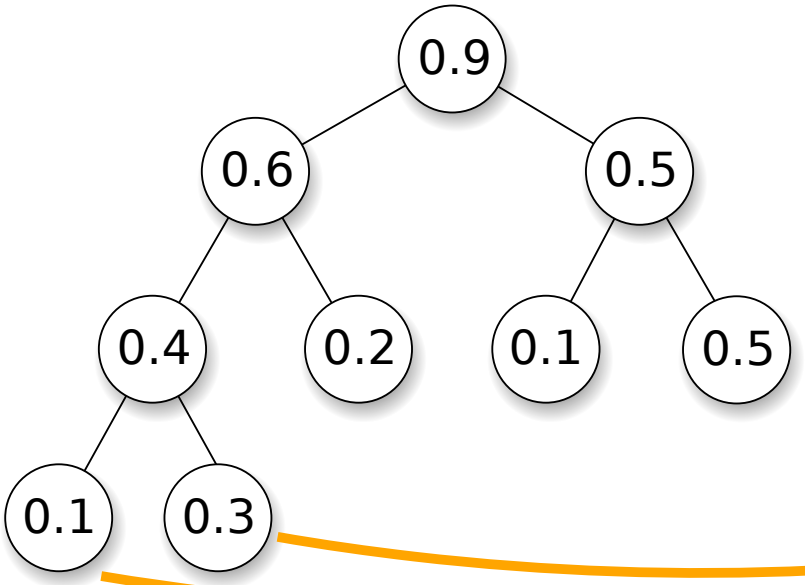
Find next spiking neuron and state update

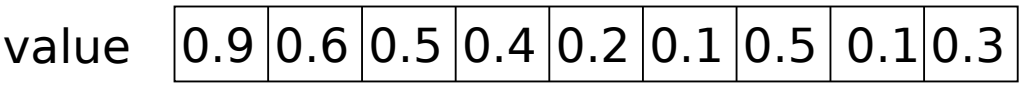
- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update



Find next spiking neuron and state update

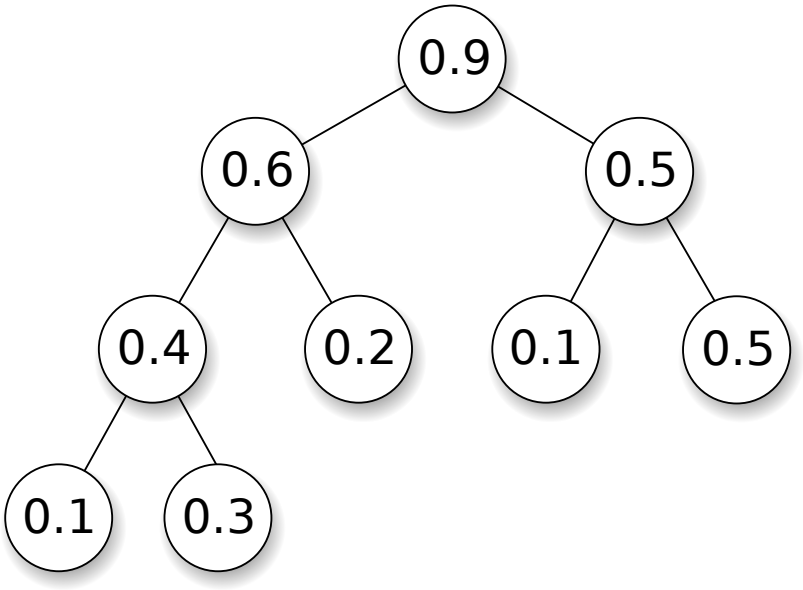
- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update





Find next spiking neuron and state update

- Problem: Finding next spiking neuron by iterating through vector of all states $\vec{\phi}$ slow for large networks $\mathcal{O}(N)$
- Solution: Put neuron states in mutable binary heap (DataStructures.jl)
  - efficiently finding next spiking neuron
  - Efficient state update

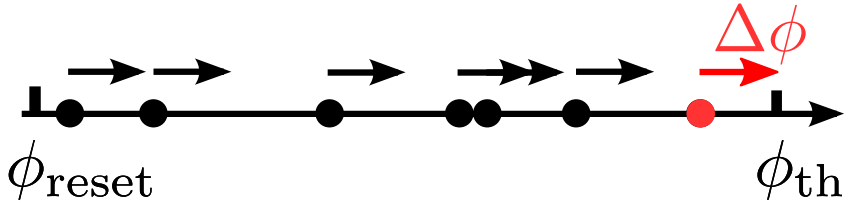


value	0.9	0.6	0.5	0.4	0.2	0.1	0.5	0.1	0.3
node	1	2	3	4	5	6	7	8	9

Novel algorithm for spiking network dynamics

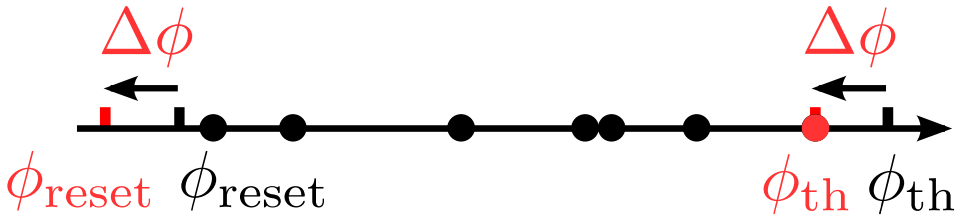
Exact firing map: $f\left(\vec{\phi}(t_s)\right) = \vec{\phi}(t_{s+1})$

- **Conventional algorithm:**

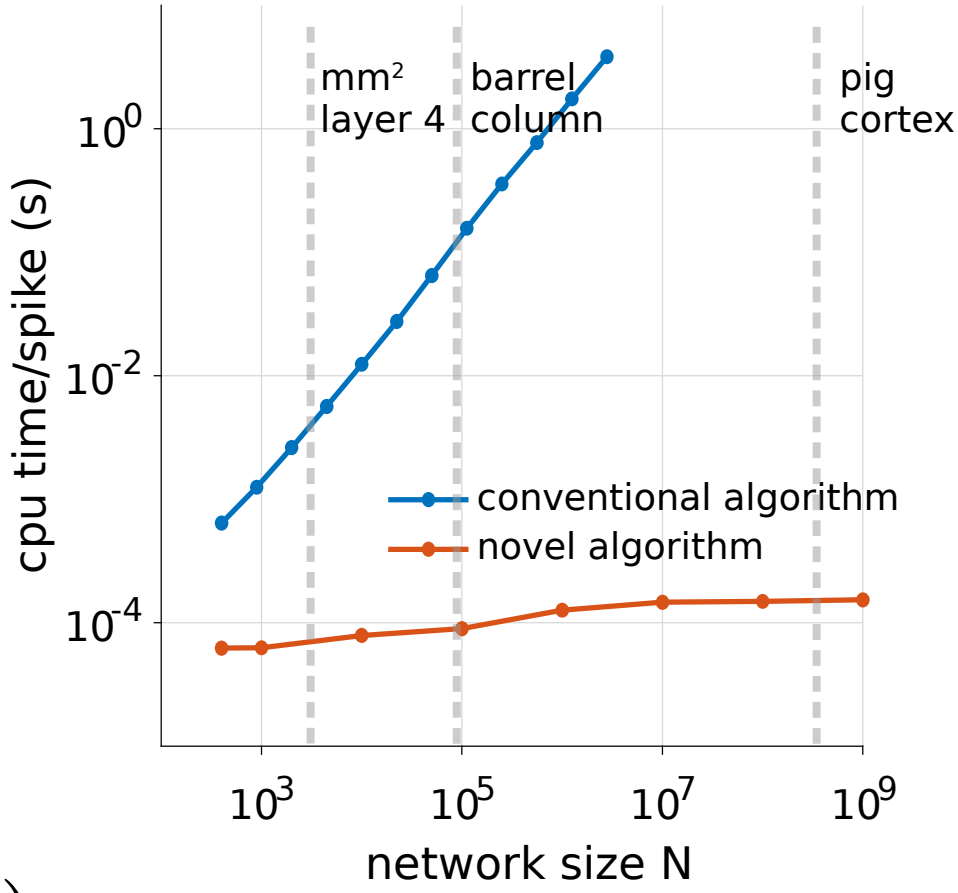


→ time-complexity $\mathcal{O}\left(N\right)$

- **Novel algorithm:**



→ time-complexity $\mathcal{O}\left(\log(N)\right)$



Intel(R) Xeon(R) CPU E5-4620 v2
@ 2.60GHz and 512 GB RAM

Demo: generating a large network topology

See Jupyter notebook

How to deal with large random networks?

- Problem: huge adjacency matrices $\mathcal{O}\left(N^2\right)$
- Solution:
  - Sparse matrices $\mathcal{O}\left(K \cdot N\right)$ K: synapses per neuron
- Better solution:
  - Generate network on the fly using fast random number generator
  - i.e. use Xoroshiro128Star (RandomNumbers.jl)
  - Memory scaling for network simulation $\mathcal{O}\left(N\right)$

Idea: Robert Rosenbaum, Eugene Izhikevich

Demo: minimal spiking network

See Jupyter notebook

Calculate Lyapunov spectrum:

- Lyapunov spectrum obtained from Jacobian along trajectory
- Problem: huge matrix-matrix multiplication at every network spike:

  $\rightarrow \mathcal{O}\left(N^2 \cdot m\right)$

  m : number of Lyapunov exponents to calculate

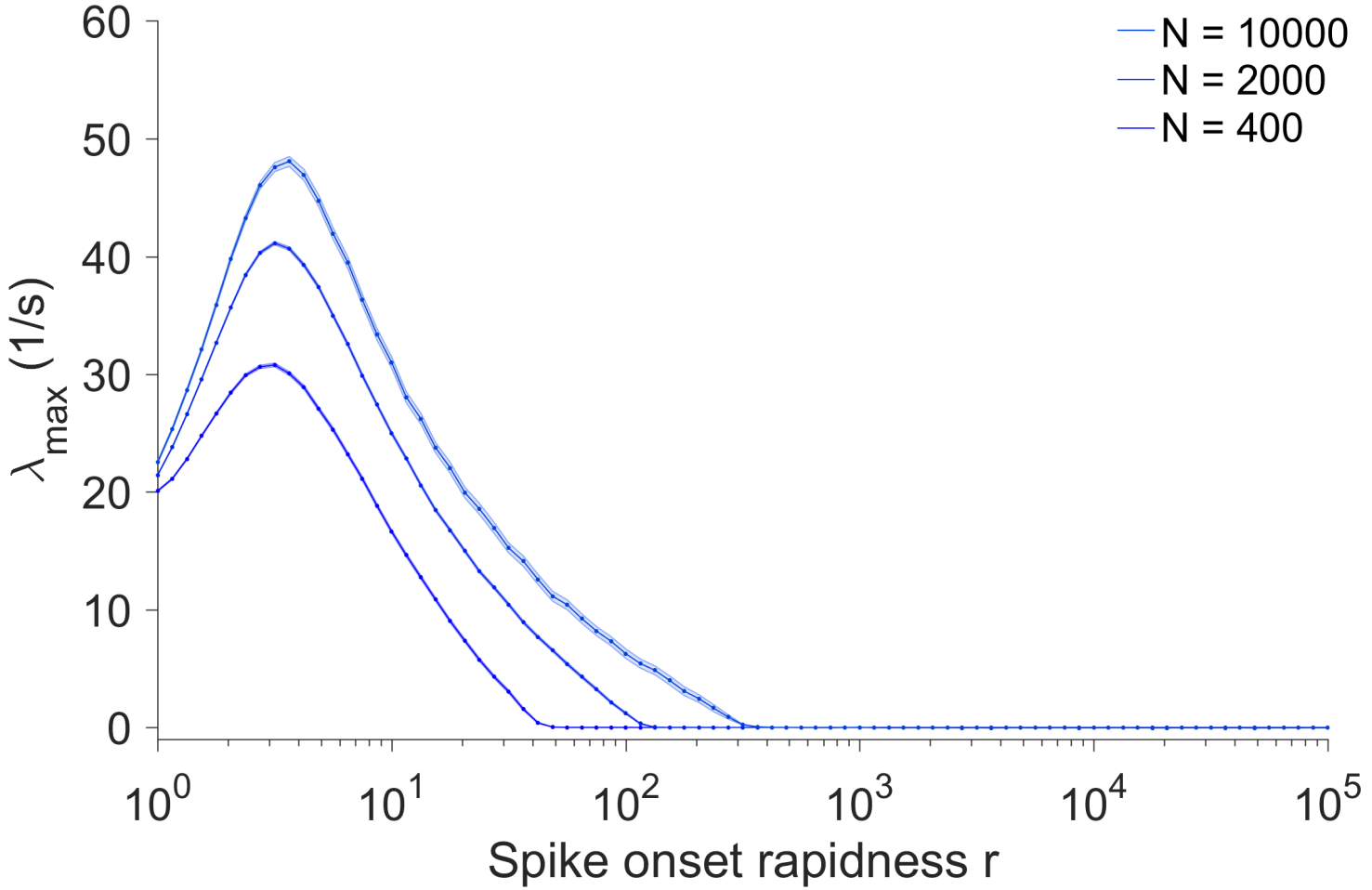
- Solution:
  - Use sparsity of Jacobian matrix $\mathcal{O}\left(2K \cdot m\right)$
  - Custom in-place matrix sparse multiplication
    - Cache-friendly code: respect column-major order of arrays
    - Make use of specific structure of Jacobian
    - Avoid any memory allocation

Applications

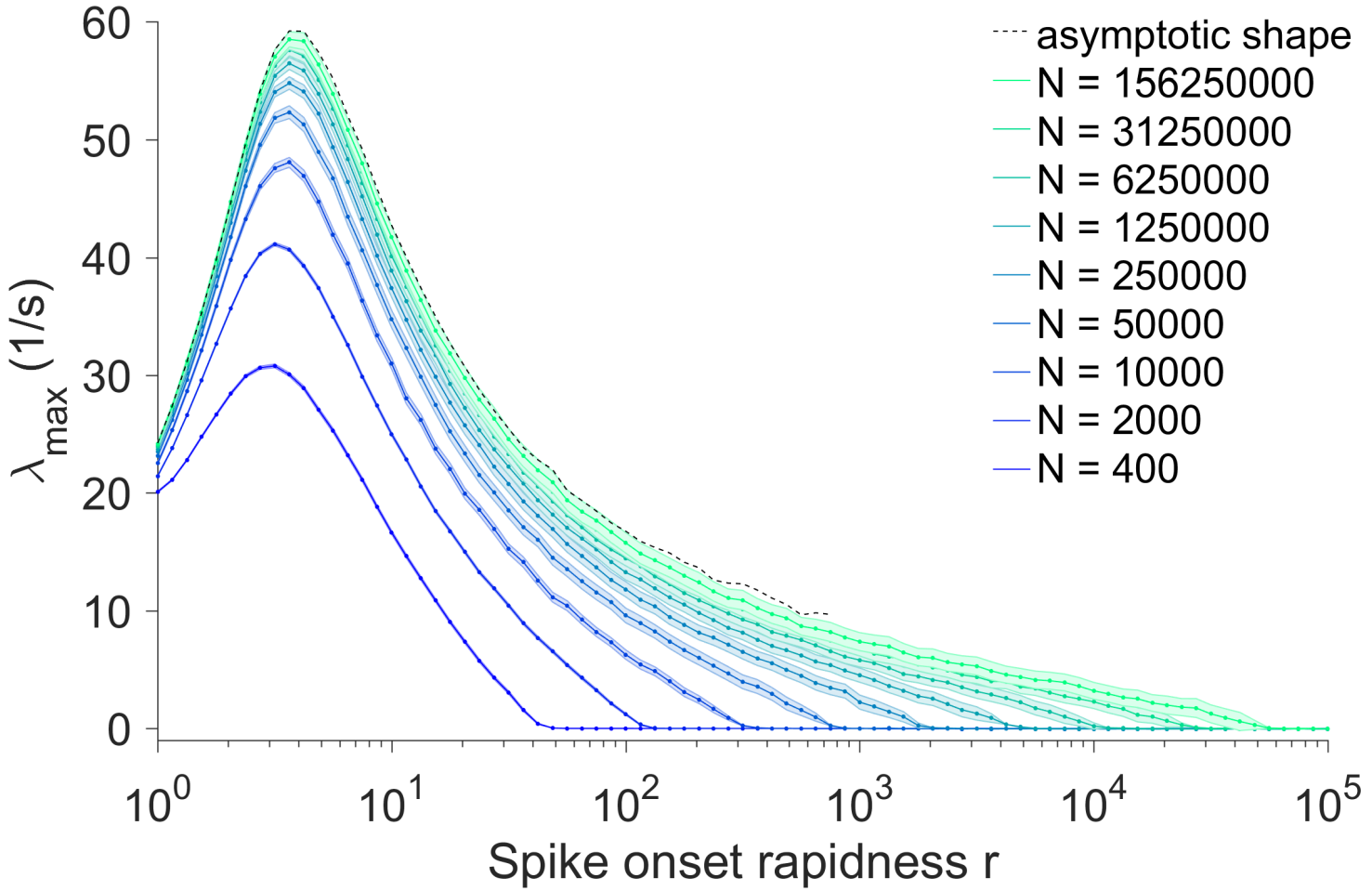
- Design of neurons that generate less chaotic circuit dynamics
  - Reduced entropy rate
  - Lower attractor dimensionality
  - Easier control from outside

- Role of input spike trains
  - Suppression of chaos by streams of input spike trains
  - Transition to complete network state control
  - Multistability for spiking input

I) Asymptotic form of largest Lyapunov exponent

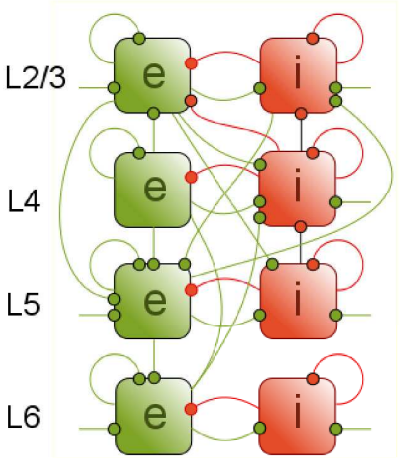


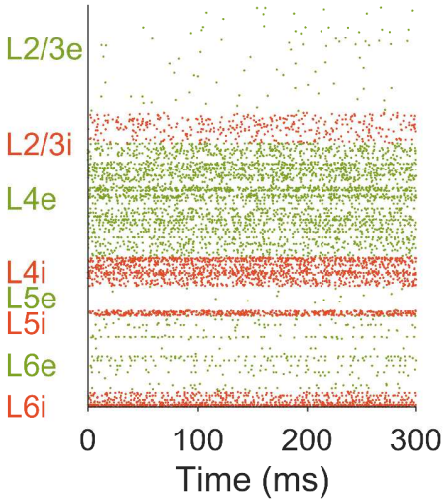
I) Asymptotic form of largest Lyapunov exponent



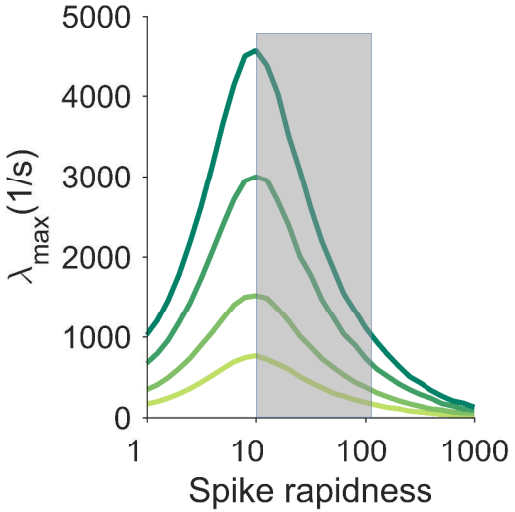
- $\lambda_{\max}$ saturates for large N
- $\mathrm{r}_{\mathrm{peak}} \propto \sqrt{K\nu_0\tau_m/J_0}$

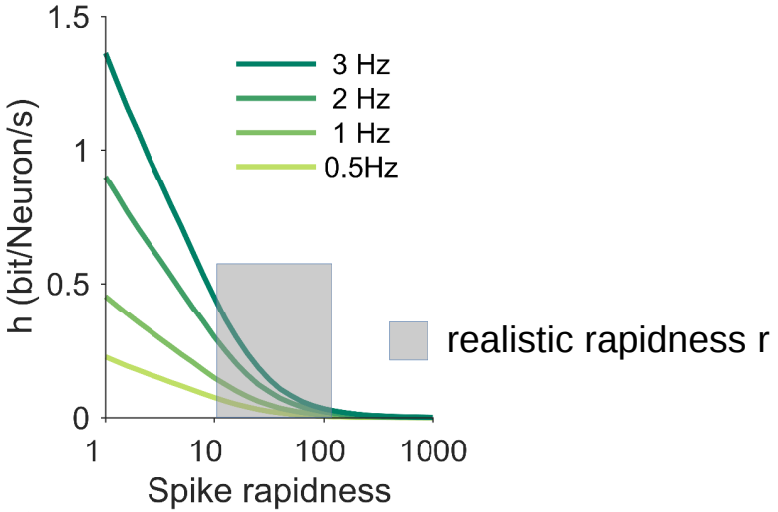
II) Taming chaos in a cortical column





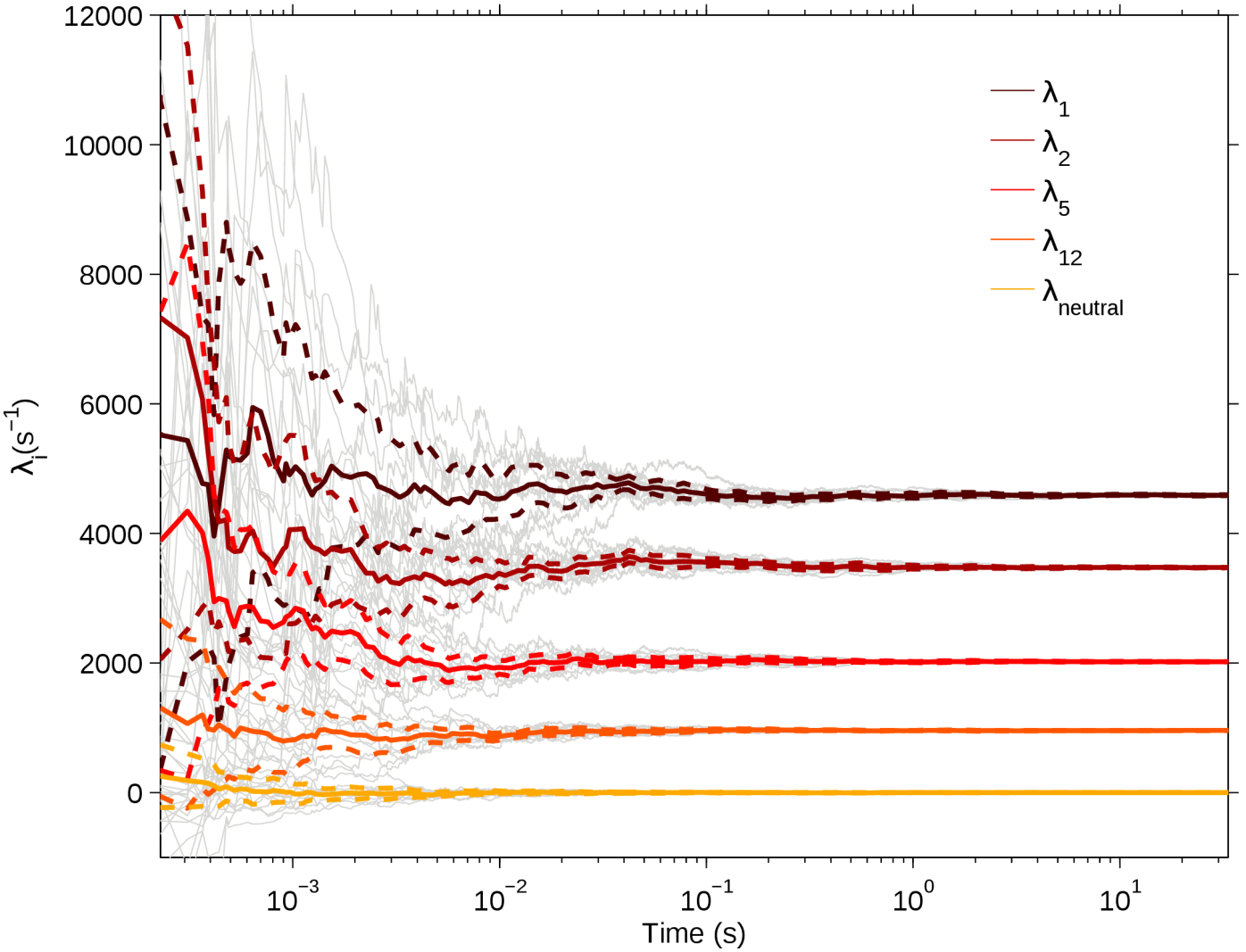
Cortical circuit model:
~ 80 000 neurons
Layered structure
'Realistic' wiring probabilities [Potjans 2014]



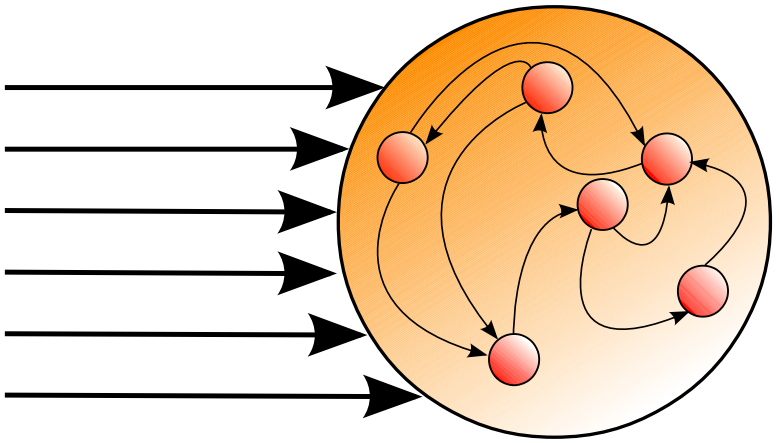


Rapid spike onset reduces entropy in cortical column model

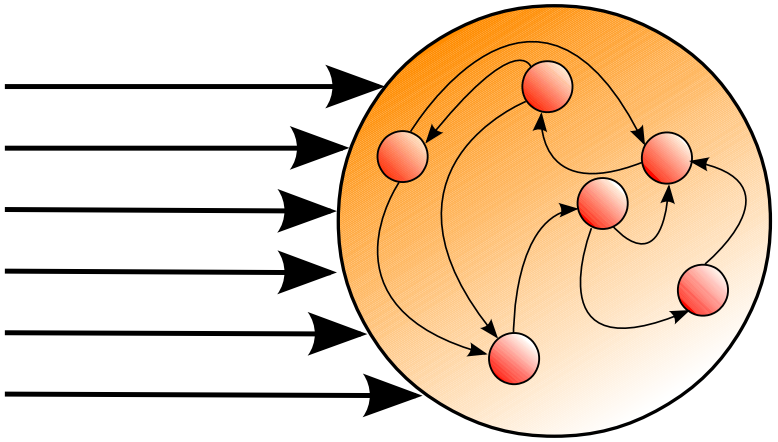
Convergence of Lyapunov spectra



III) Spike-driven network dynamics



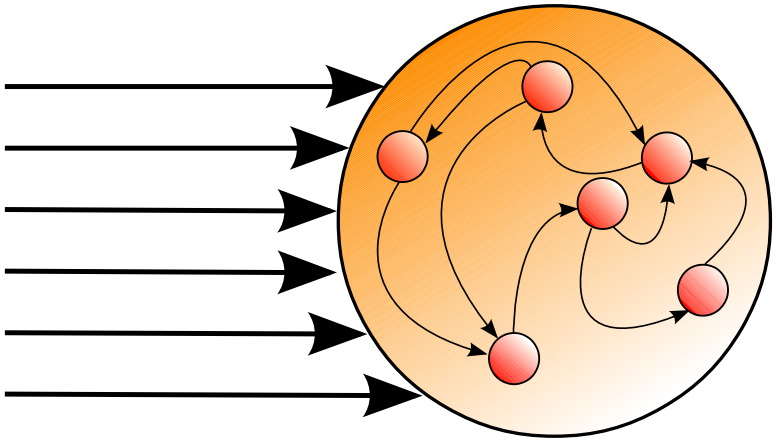
III) Spike-driven network dynamics



constant input [Monteforte, Wolf 2010, 2012]

white noise [Lajoie et al., 2013, 2014, 2016]

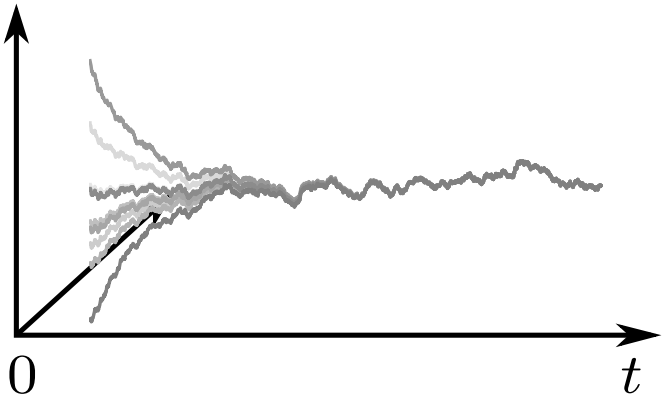
III) Spike-driven network dynamics



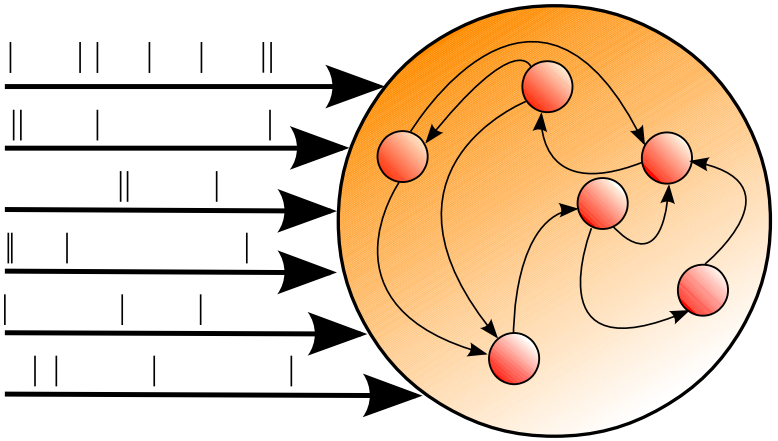
constant input [Monteforte, Wolf 2010, 2012]

white noise [Lajoie et al., 2013, 2014, 2016]

→ random sink for $\lambda_{max} < 0$



III) Spike-driven network dynamics

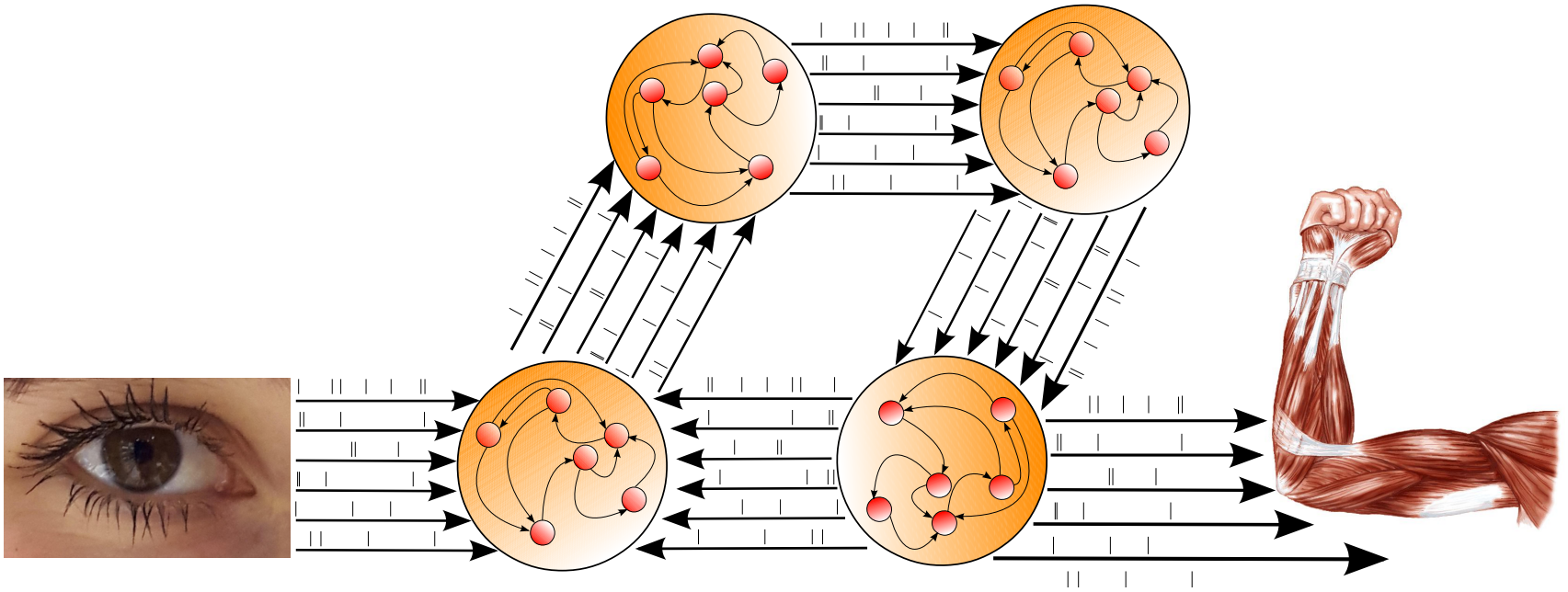


constant input [Monteforte, Wolf 2010, 2012]

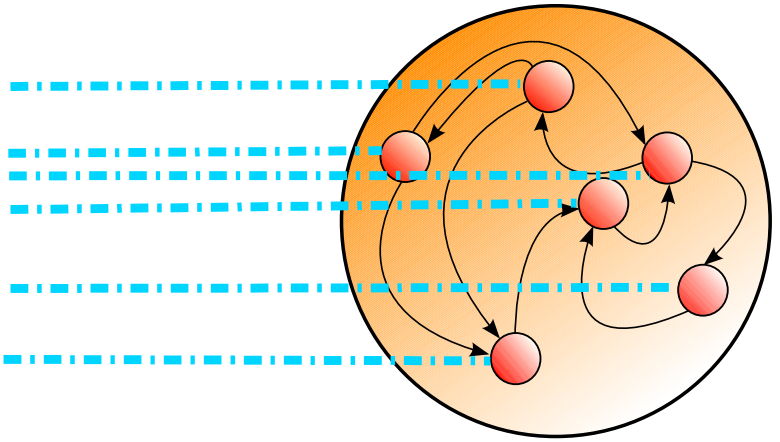
white noise [Lajoie et al., 2013, 2014, 2016]

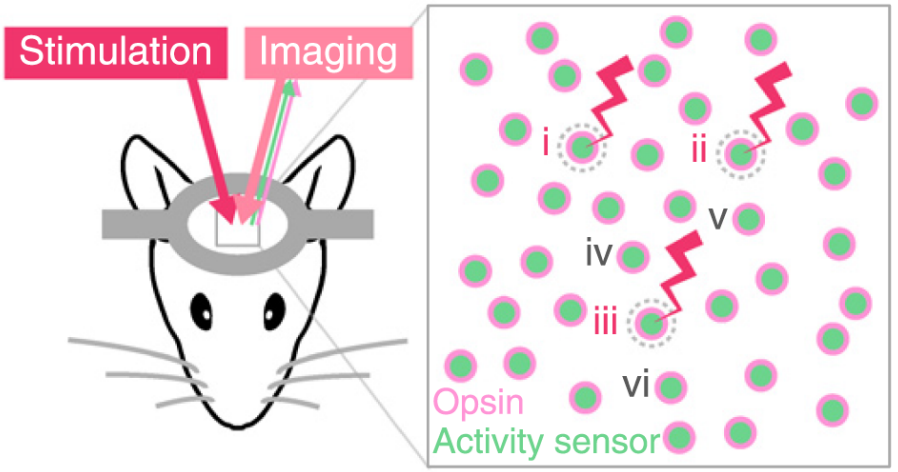
→ random sink for $\lambda_{max} < 0$

III) Spike-driven network dynamics



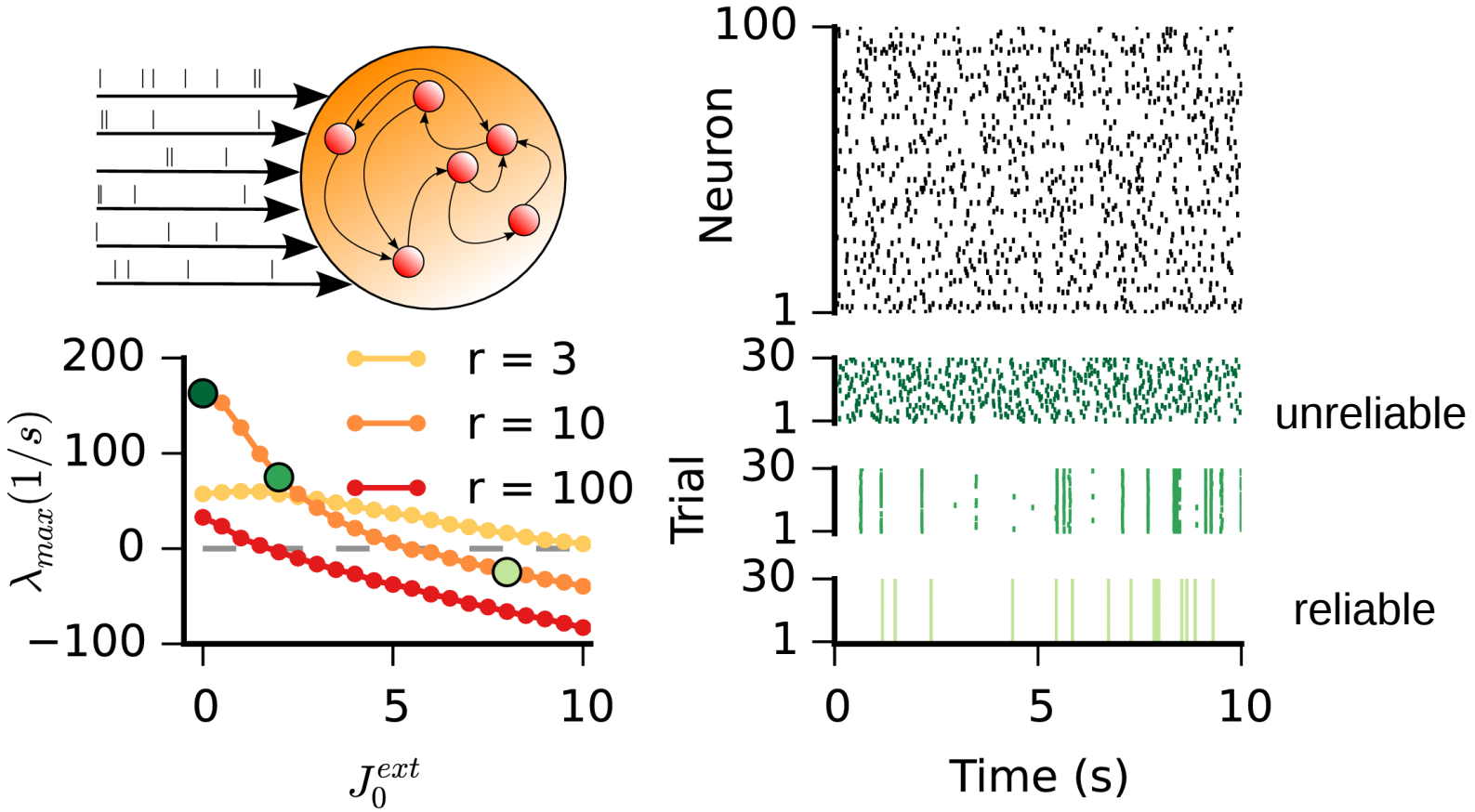
III) Spike-driven network dynamics





[Packer et al., 2015]

Input spike trains suppress network chaos



Suppression of chaos by streams of input spike trains

→ facilitated by rapid spike onset

→ transition to complete network state control (reliable)

Julia summary

- Easily find bottlenecks of earlier implementation
- Write new efficient implementation in a few lines of code
- Optimize it within Julia
- Helpful features:
  - Fast loops
  - Easy profiling and visualization: Profile, ProfileView.jl
  - Easy tracking of memory allocation: --track-allocation
  - Easy track down type instabilities: @code_warntype
  - Fast random number generator: RandomNumbers.jl
  - Check precision using BigFloats
  - Data types (e.g. binary heap) ‘off the shelf’: DataStructures.jl
  - Transparent Julia core code

Julia helps to make scientific computation transparent & reproducibility

Acknowledgments

MPI DS:

Fred Wolf

Michael Monteforte

Max Puelma Touzel

Joscha Liedtke

Manuel Schottdorf

Guillaume Lajoie

Agostina Palmigiano

Alexander Schmidt

Raoul-Martin Memmesheimer

Julia Community:

RandomNumbers.jl

DataStructures.jl

gitter folks

Tim Holy

Chris Rackauckas

Simon Danisch

Evangelisches Studienwerk e.V. Villigst

Wir bewegen Wissen.