Dynamic Games and HTCondor

Mordecai Kurz, Kenneth Judd and Nathan Lazarus

CEF 2022

June 19, 2022

Overview

- How to solve dynamic games and dynamic programming problems in parallel on a computational grid
  - Computational structure and workflow
- Applications:
  - strategic competition in the world oil market
  - integrated assessment models of climate change
  - R&D investment in a market with many firms
  - Production with inventories

Introducing HTCondor

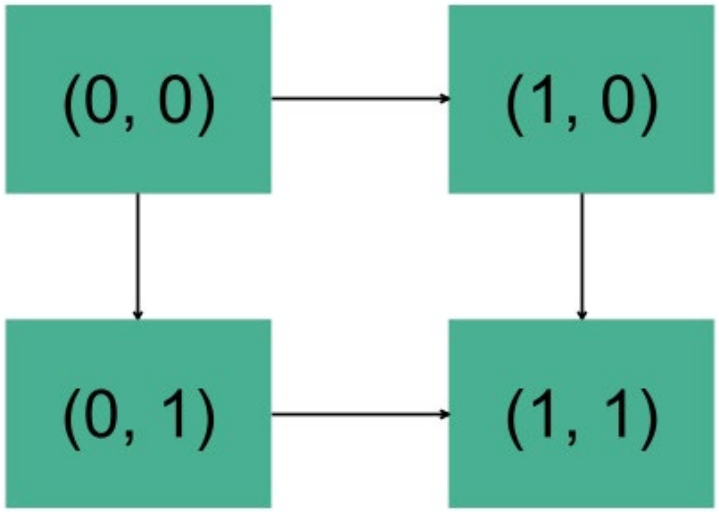
- HTCondor is a scheduling system for managing distributed computing nodes.
- Matches jobs and machines (nodes) based on “ClassAds” (requirements—disk space, cores, etc.)
- HTC (high-throughput computing)
- DAGMan: an extension to HTCondor that allows workflows to be described as DAGs

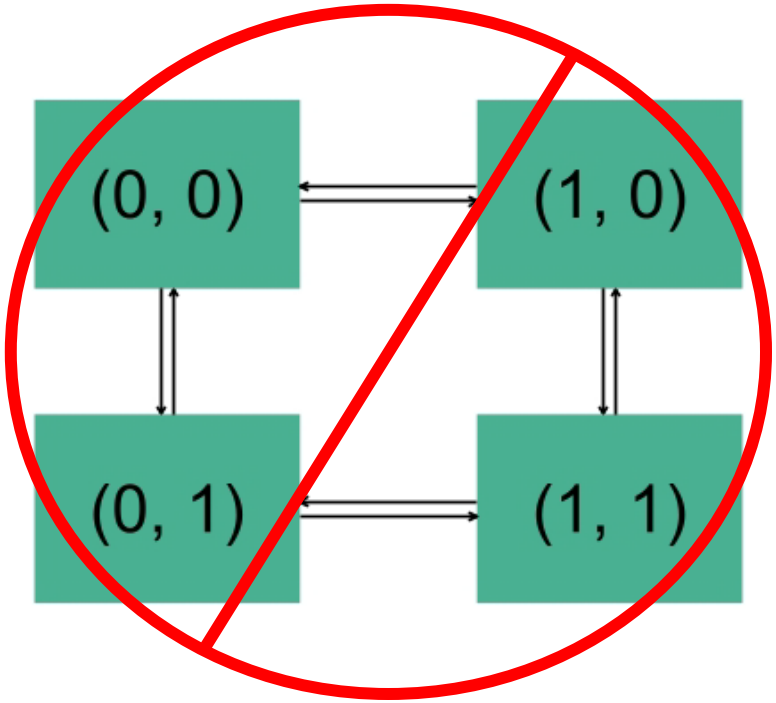
CHTC

- Center for High Throughput Computing (UW-Madison)
- A cluster with over 20,000 total cores
- Part of the Open Science Grid

Directed Acyclic Graph Structure

- What makes problems acyclic?
- Working on the extension to cyclic problems (will talk more about it at the end, time permitting)





Solution Method

- Solve backwards from the terminal states
- Works for anything with this unidirectional flow (a dynamic game, dynamic programming problems)

Applications

- Oil Extraction
  - Oil doesn’t go back into the ground.
- Time
  - A climate model where the terminal state is 600 years from now, and we solve for values and optimal policies back to the present.

Oil Game Model: Static Component

(Bold denotes vectors)

- Cost: $c(q_{jt}) = \kappa_j \frac{q_{jt}^2}{2}$
- Pricing: $p(\mathbf{q}_t) = A - \frac{\sum_{j=1}^{N} q_{jt}}{N}$
- Profit: $\Pi_j(\mathbf{q}_t) = p(\mathbf{q}_t) q_{jt} - c(q_{jt})$
- Reserves constraint: $R_{jt} = 0 \rightarrow q_{jt} = 0$

The only effect of reserves is that production must be 0 if reserves are 0. (A more realistic formulation is that production becomes more costly as reserves approach 0.)

Oil Game Model: Dynamics

- Reserves transition: $\mathbf{R}_t = \mathbf{R}_0 - \epsilon$
- Jumps: $P(\epsilon_j = n) = \frac{(\int_0^t \delta q_j(t)dt)^n \exp(-\int_0^t \delta q_j(t)dt)}{n!}$
  - (non-homogeneous Poisson process, equivalent to stretching or compressing the time dimension of a standard Poisson process)
- Transition hazard rate: $\phi_{jt} = \delta q_{jt}$

- Doraszelski-Judd (2012)

Oil Game Model: Solution

(ρ is the discount rate)

- Bellman equation:

$$\rho V_{jt}(\mathbf{R}) = \Pi_{jt} + (V_{jt}(\mathbf{R} + \mathbb{I}) - V_{jt}(\mathbf{R}))\phi_t$$

$$\rho V_{jt}(\mathbf{R}) = (A - \frac{\sum \mathbf{q_t}}{N})q_{jt} - \kappa_j \frac{q_{jt}^2}{2} + (V_{jt}(\mathbf{R} + \mathbb{I}) - V_{jt}(\mathbf{R}))\delta\mathbf{q_t}$$

- Investment FOC:

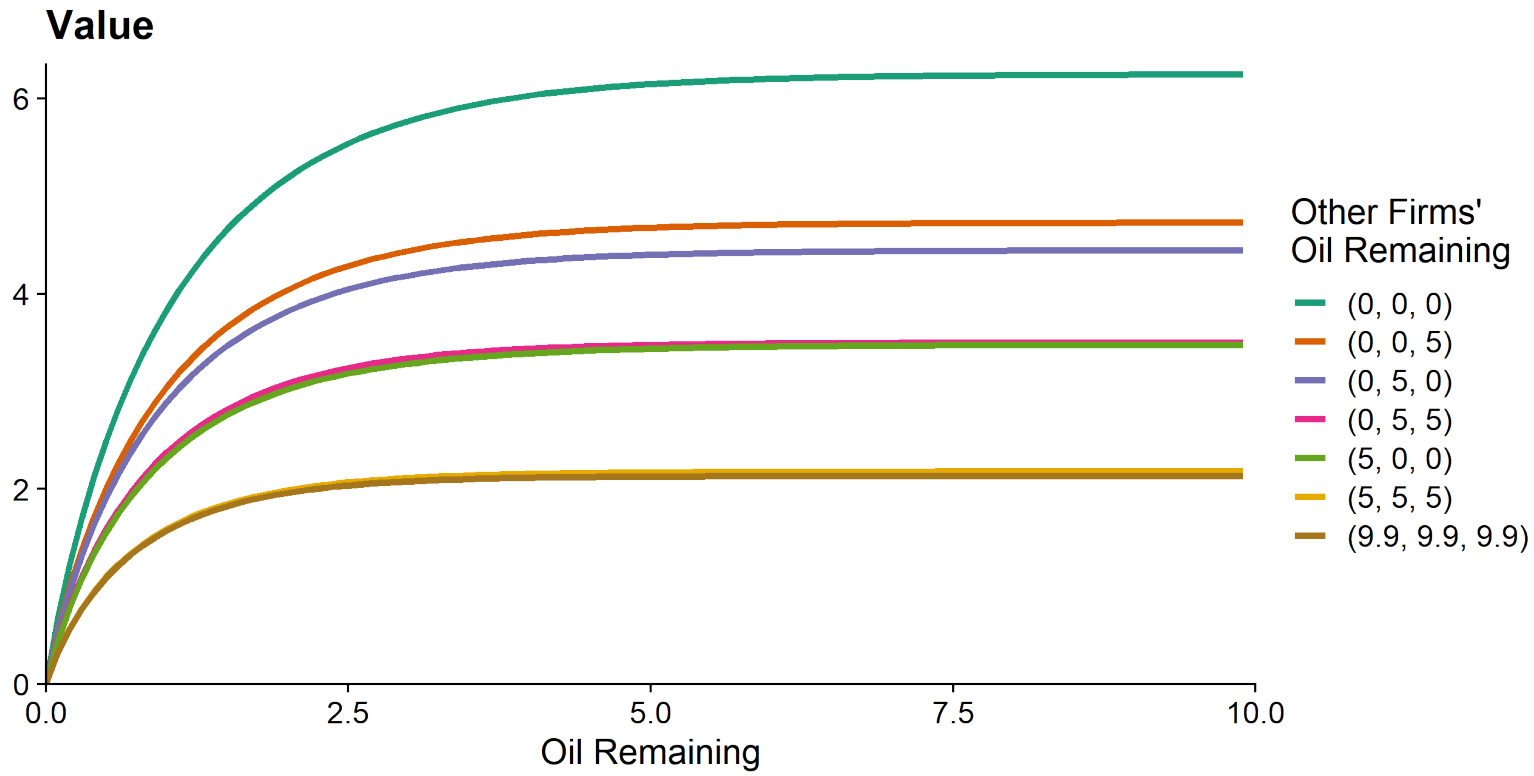
$$0 = \frac{q_{jt}}{N} + p(\mathbf{q_t}) - \kappa_j q_{jt} + (V_{jt}(\mathbf{R} + \mathbb{I}) - V_{jt}(\mathbf{R}))\delta\mathbf{q_t}$$

Solution Method

- N quadratic value functions plus N linear FOCs:
  - Bézout number 2^N
- The value function surface is bumpy, so optimizers will have trouble with it (and engage in implicit equilibrium selection)
- All-solution homotopy methods (Bertini, see Judd, Renner and Schmedders (2012))
  - Deliberate equilibrium selection
- Except Bertini can't solve for constrained equilibria, so implementing a barrier function approach

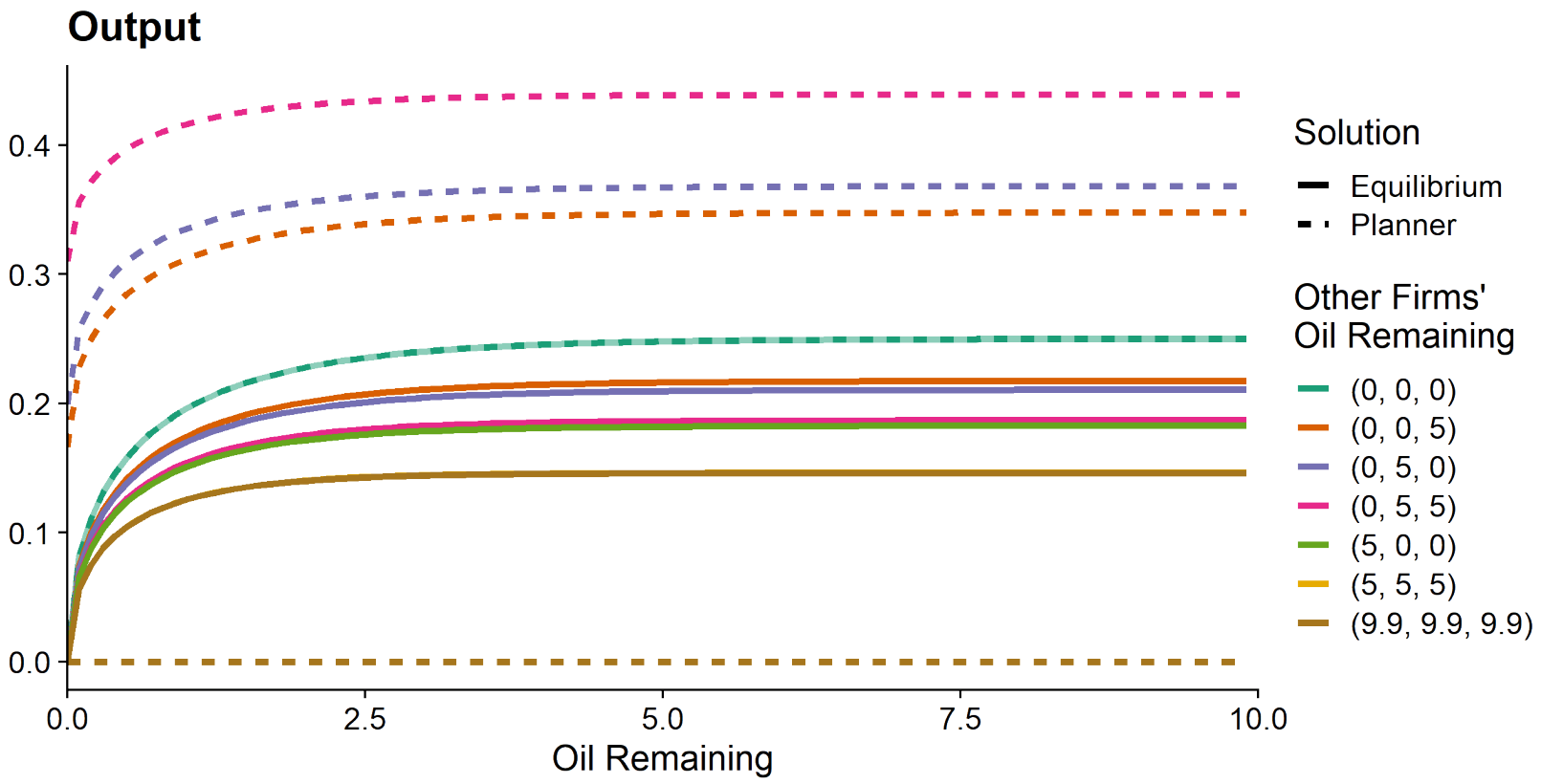
Oil Game Results

- 4 players, 100 states, value functions for the firm/country with the second highest productivity
- Parameter choices: $\kappa = [1, 2, 3, 4]; \rho = 0.02$



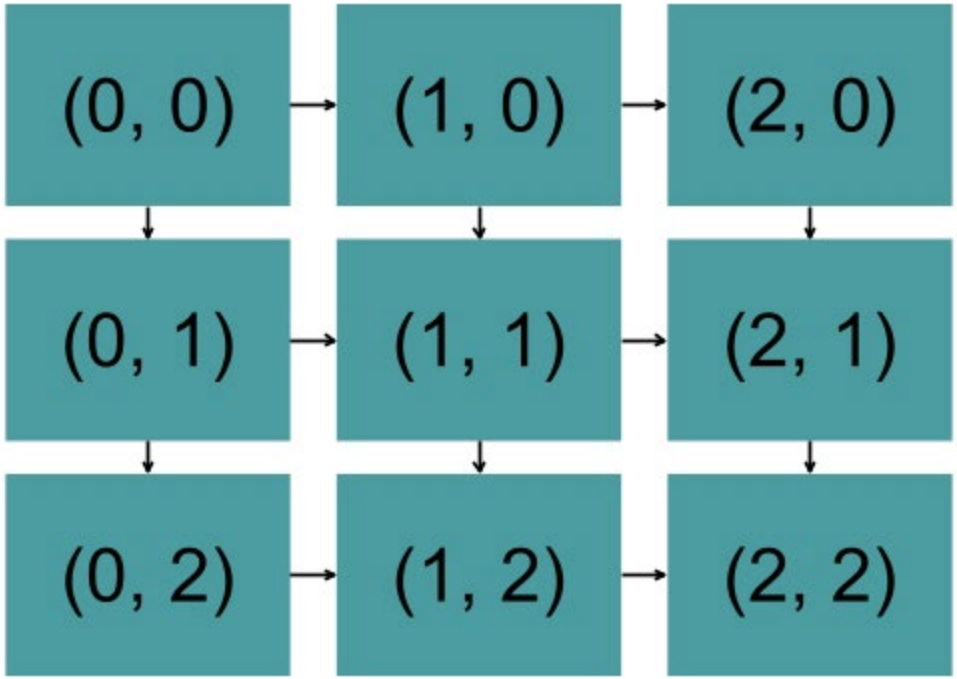
Oil Game Results

- Policy functions very different from the planners' (which is extract in the lowest-cost country first)



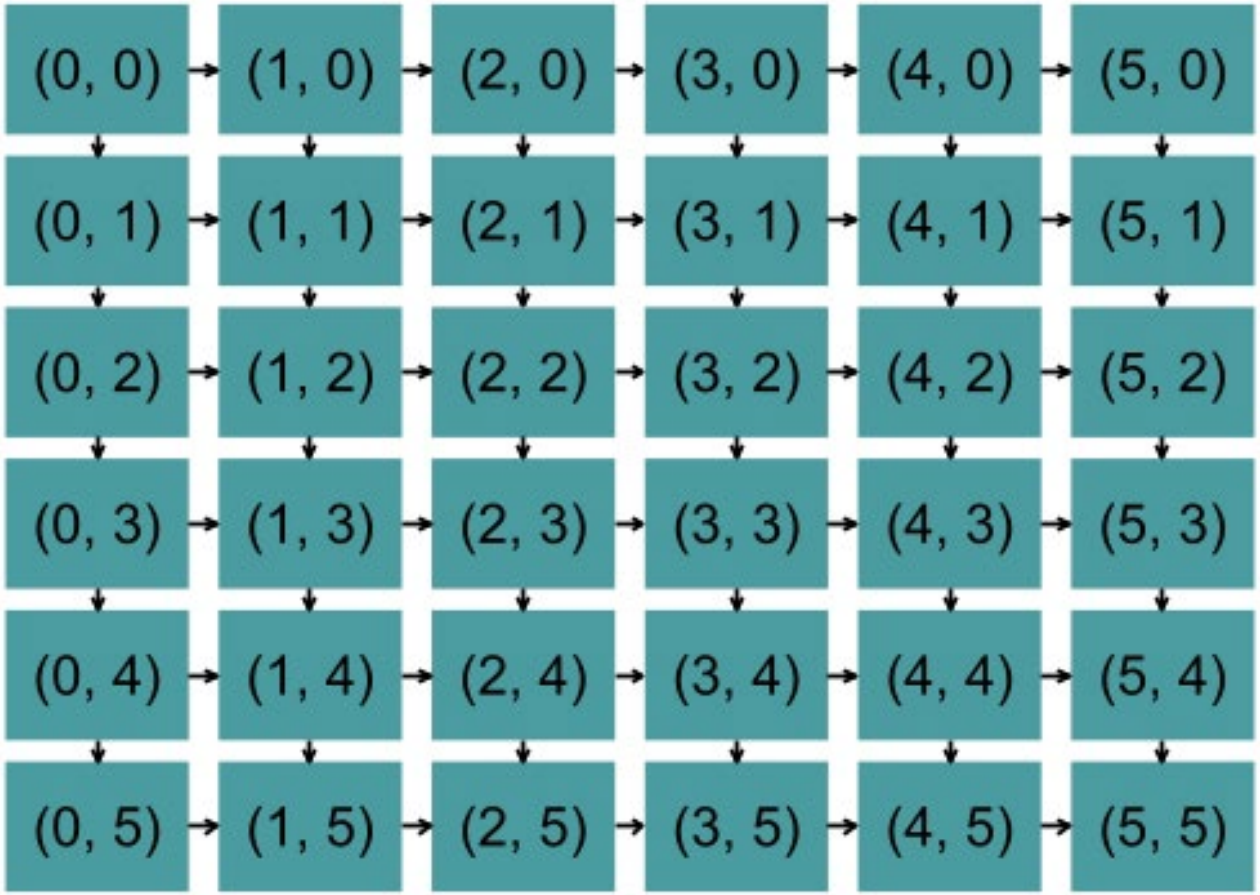
The Computational Structure

A small two-player game



The Computational Structure

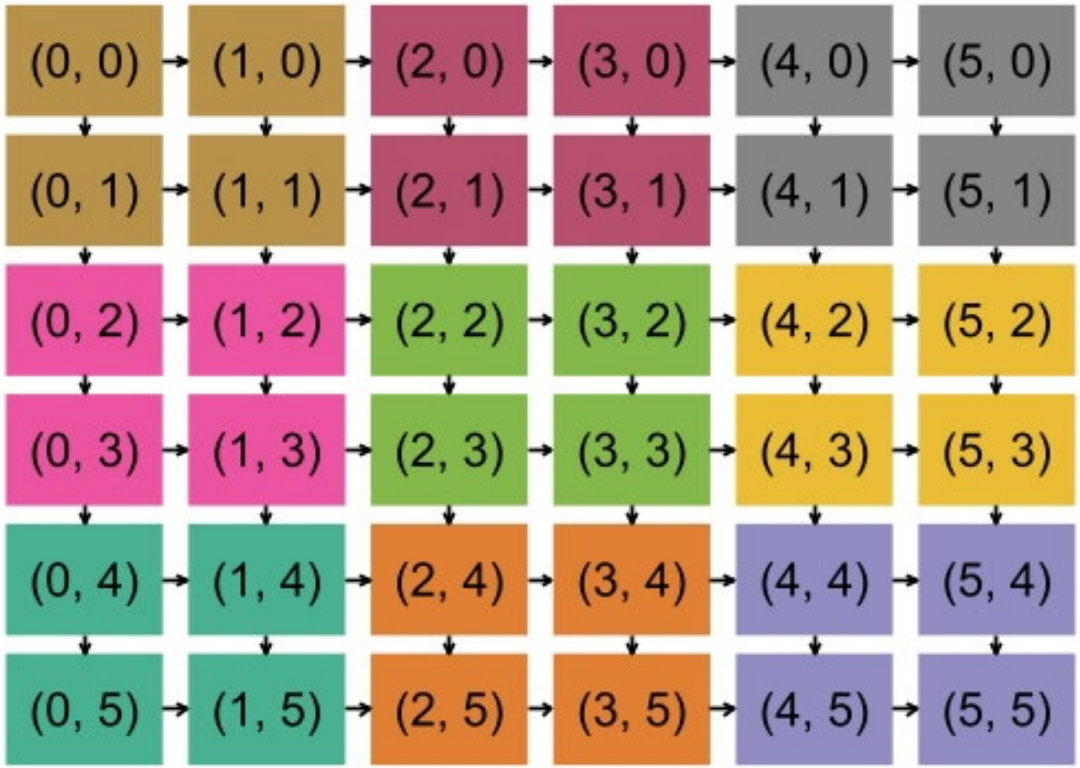
A large two-player game



(I won't try a visual of higher-dimensional cubes for the case with more than 2 players.)

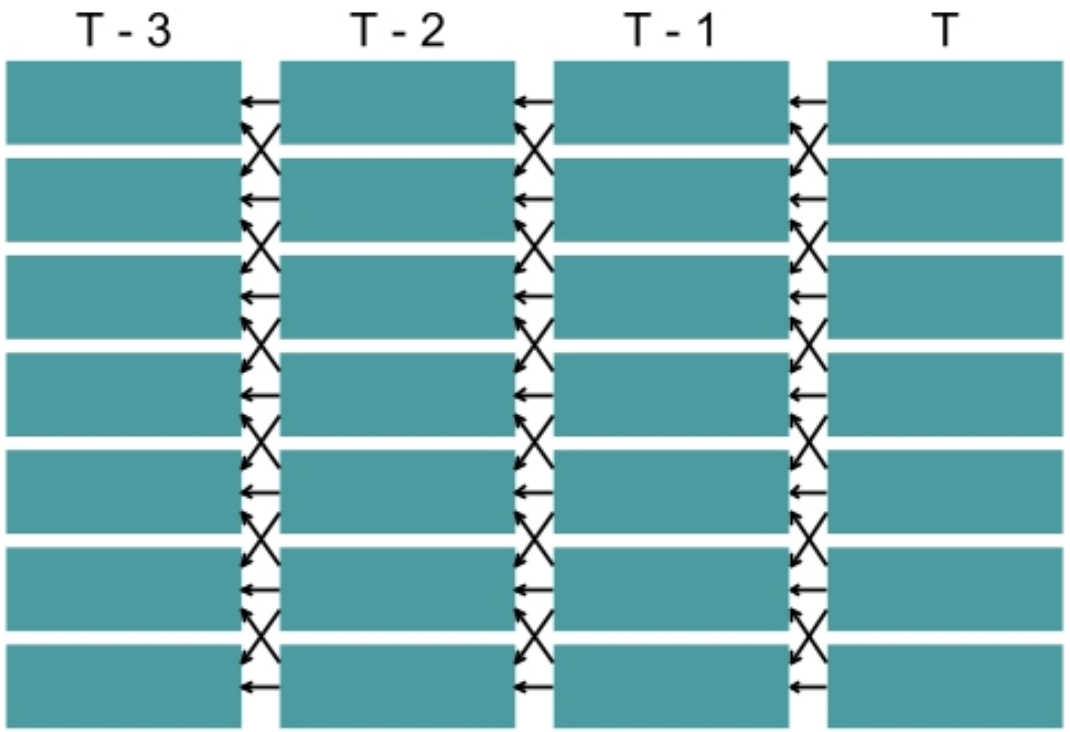
The Computational Structure

- 4 players and 100 states means 100^4=100 million jobs.
- Each one runs in a tenth of a second.
- There's overhead to every job submission.
- Solution: group them into batches



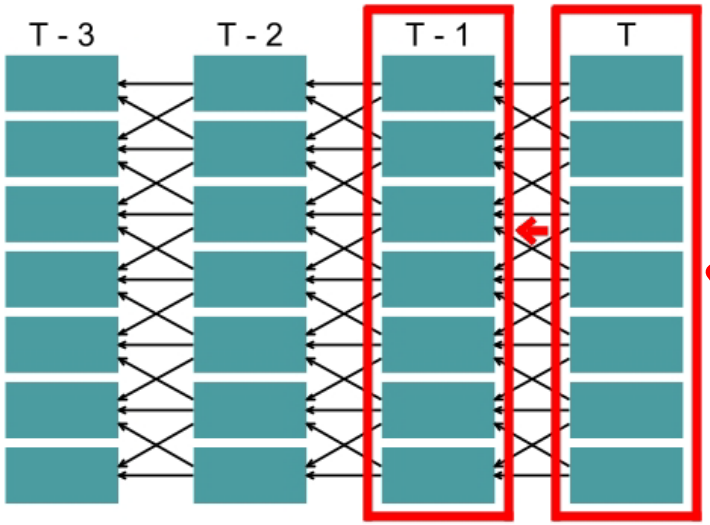
The Computational Structure

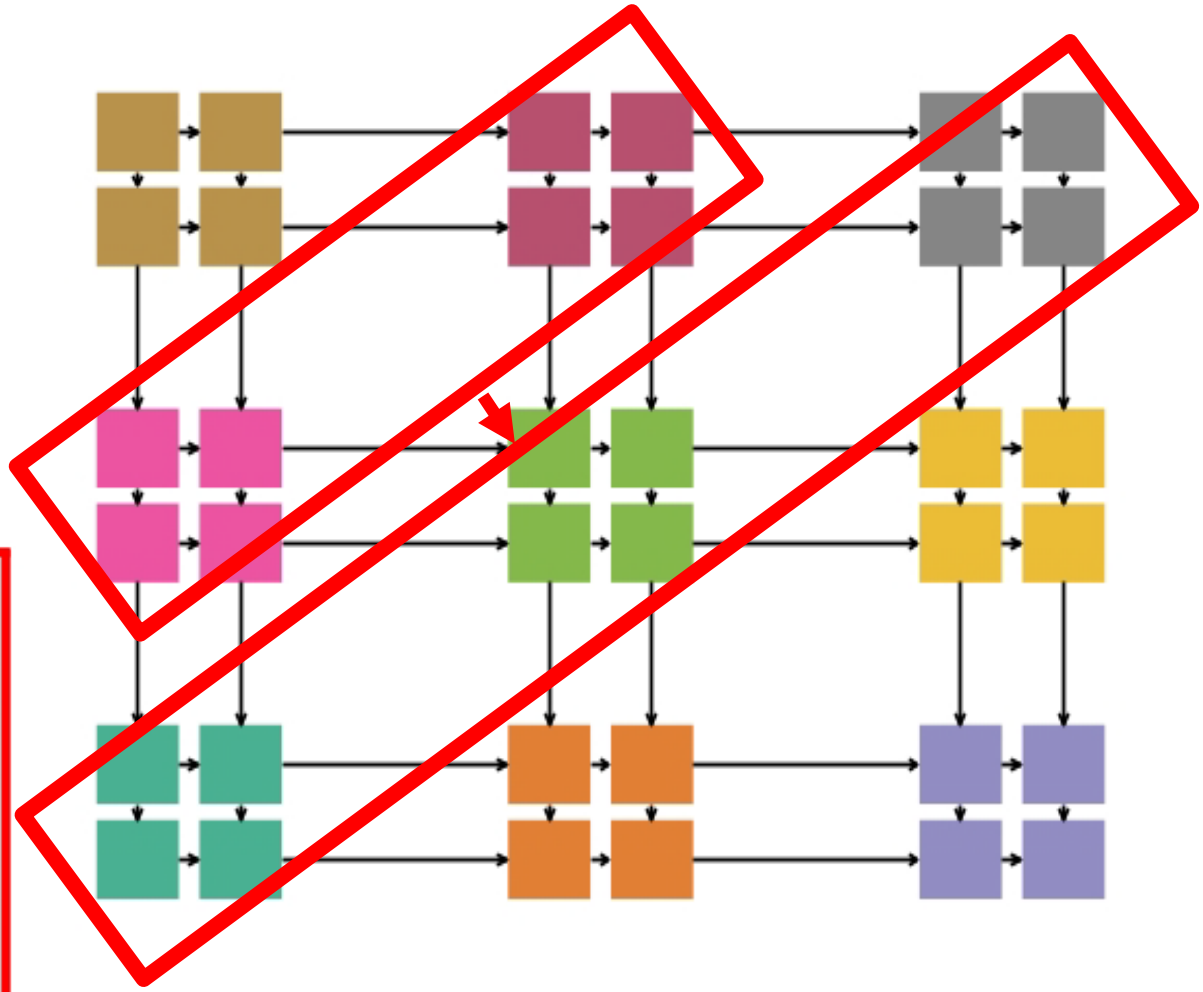
- For this dynamic programming problem, the direction of flow is rotated 135 degrees; instead of going down and to the right, it flows left.
- Again, I want to focus on the sparsity of the state transitions.
- In the full model, we have 1729 state discrete states per period, but each state only has 35 possible transitions, and 35 dependencies.



Block-wise Parallelization

- This type of parallelization waits for the last job from one block before starting on any jobs in the next block.



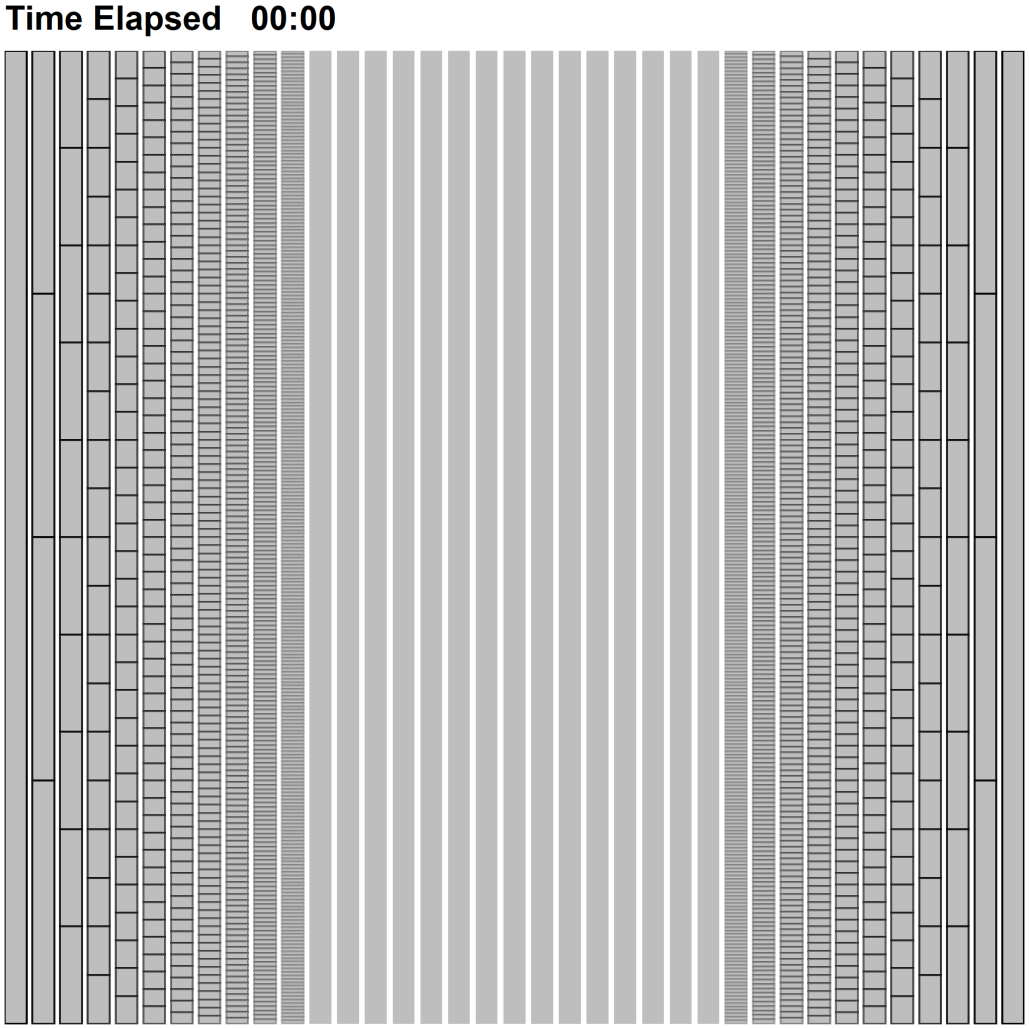


Wavefront

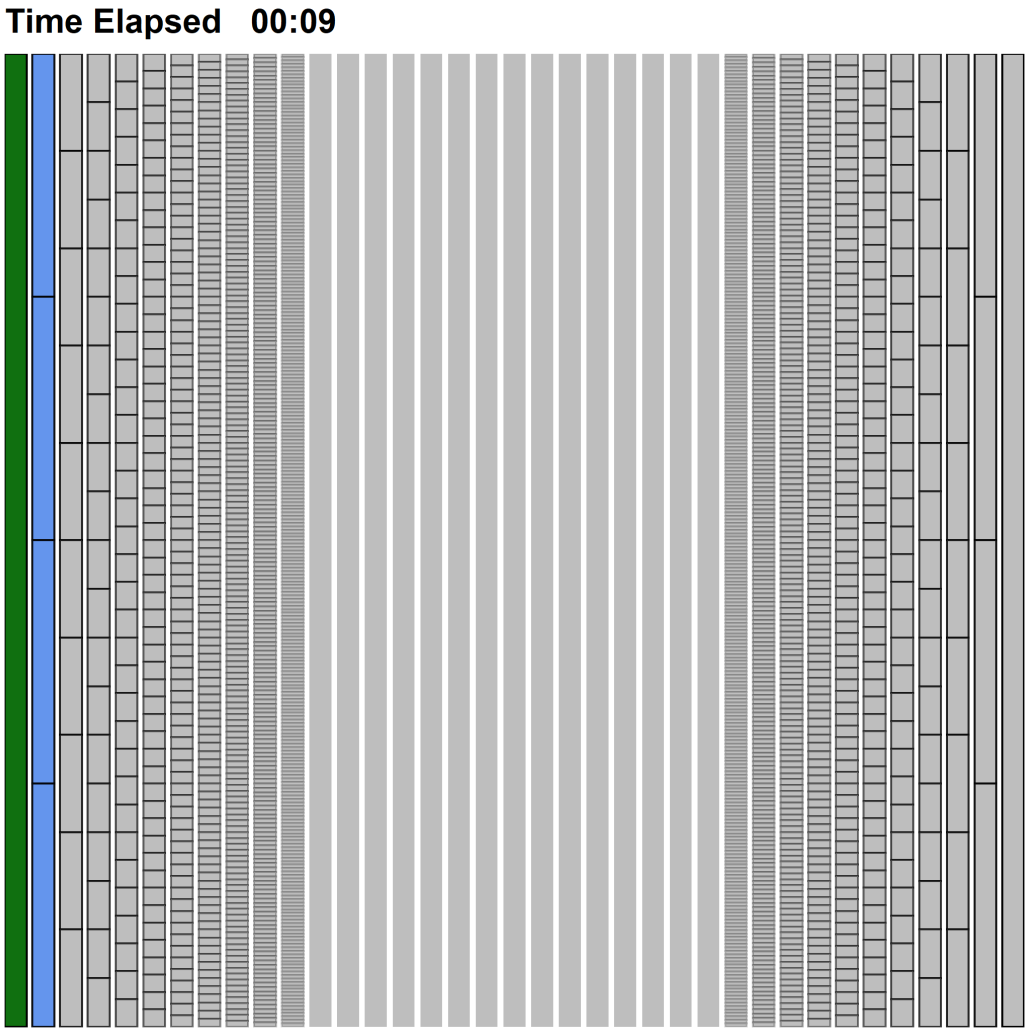
- Starts jobs when their dependencies are ready.
- Much more efficient on a computational grid: 6x speedup
- Particularly beneficial on larger problems

Wavefront

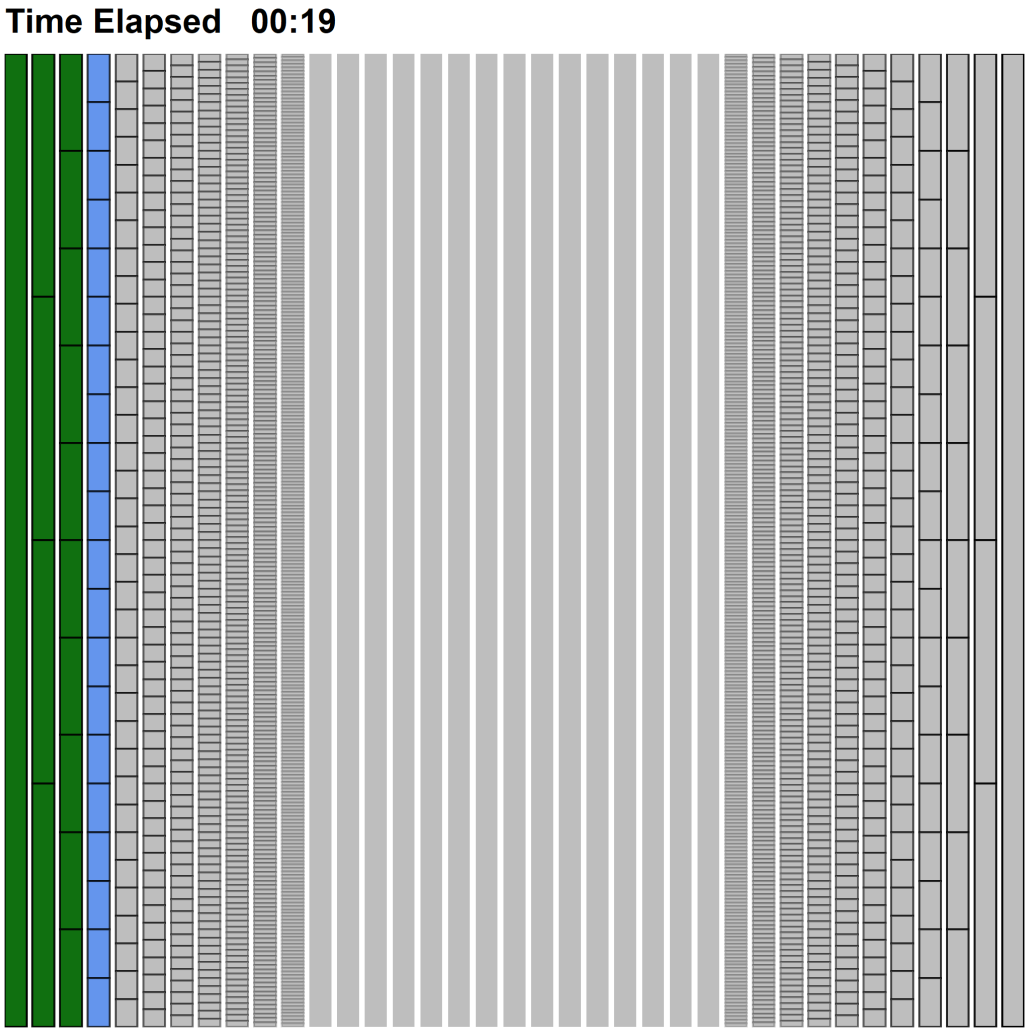
Here, I sort jobs into blocks, and then within blocks by the time at which they're submitted.



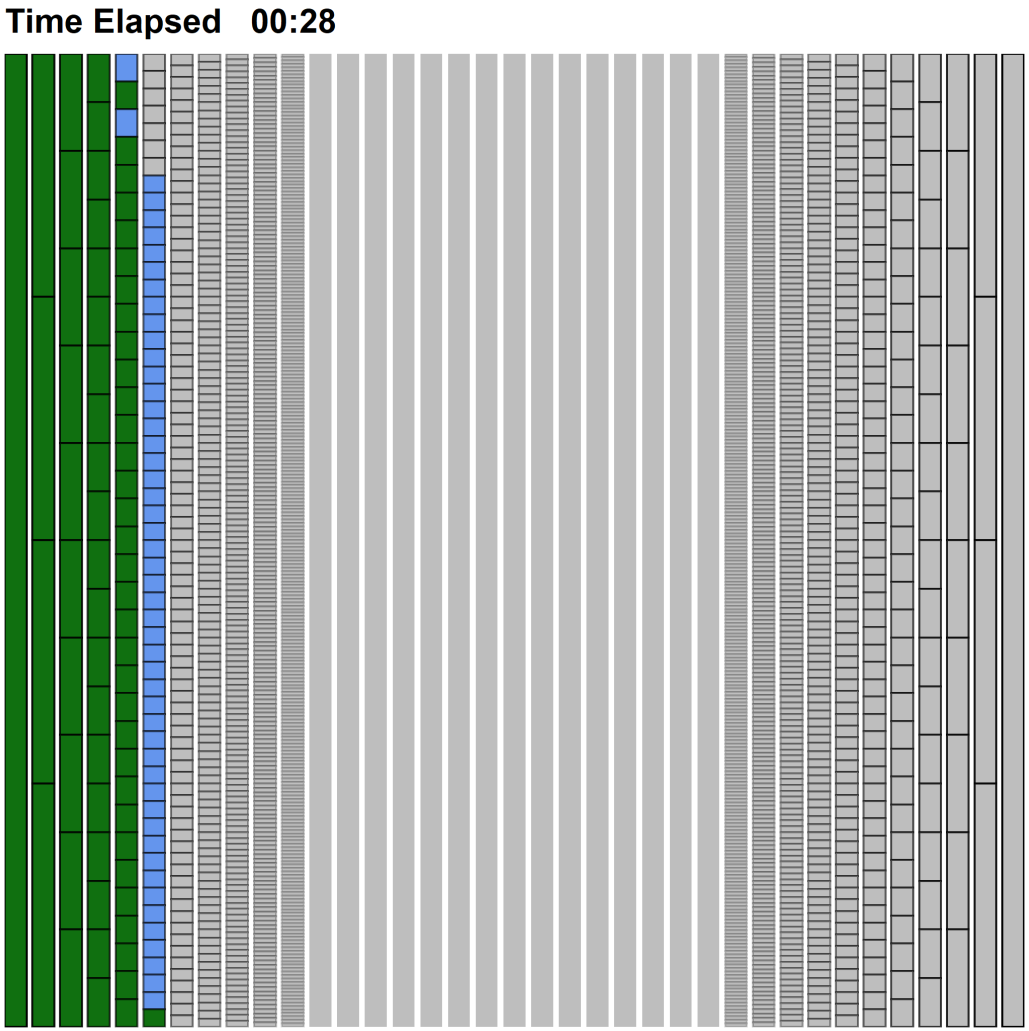
Wavefront



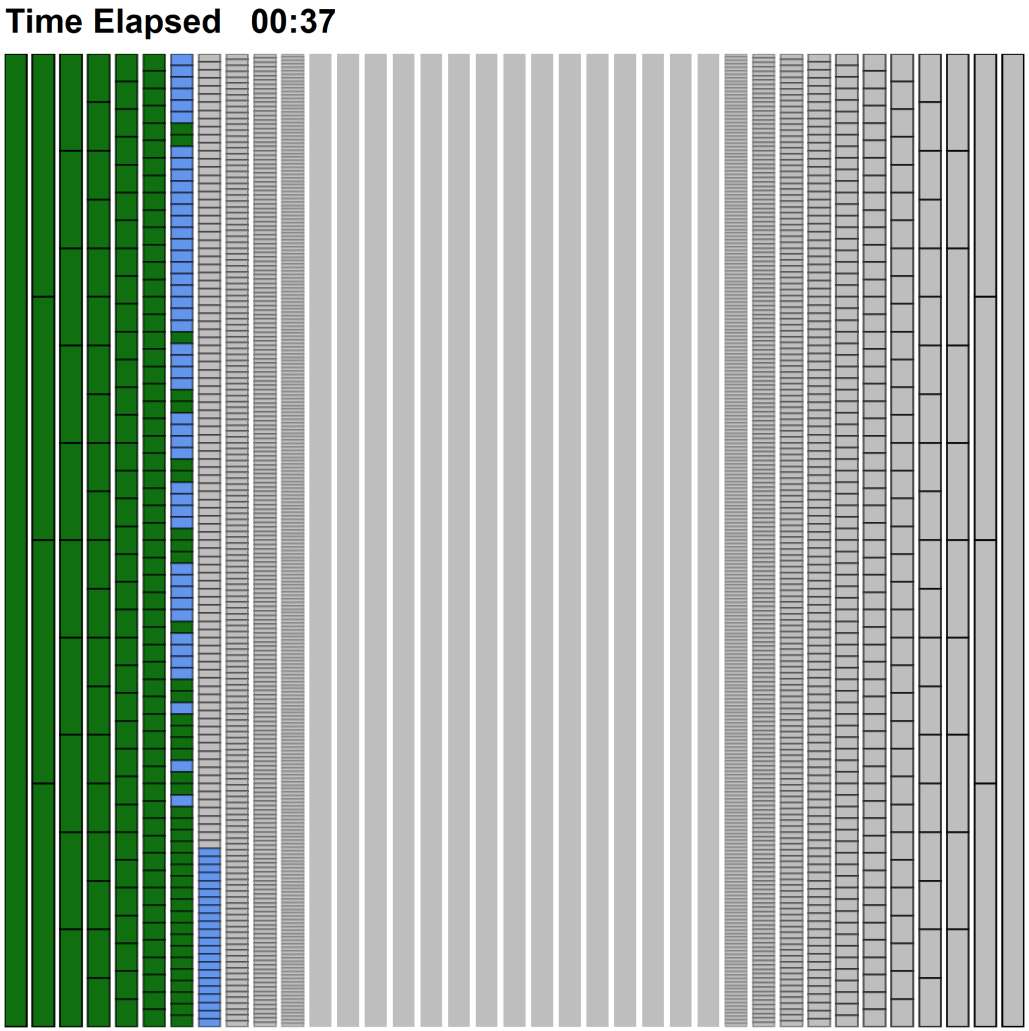
Wavefront



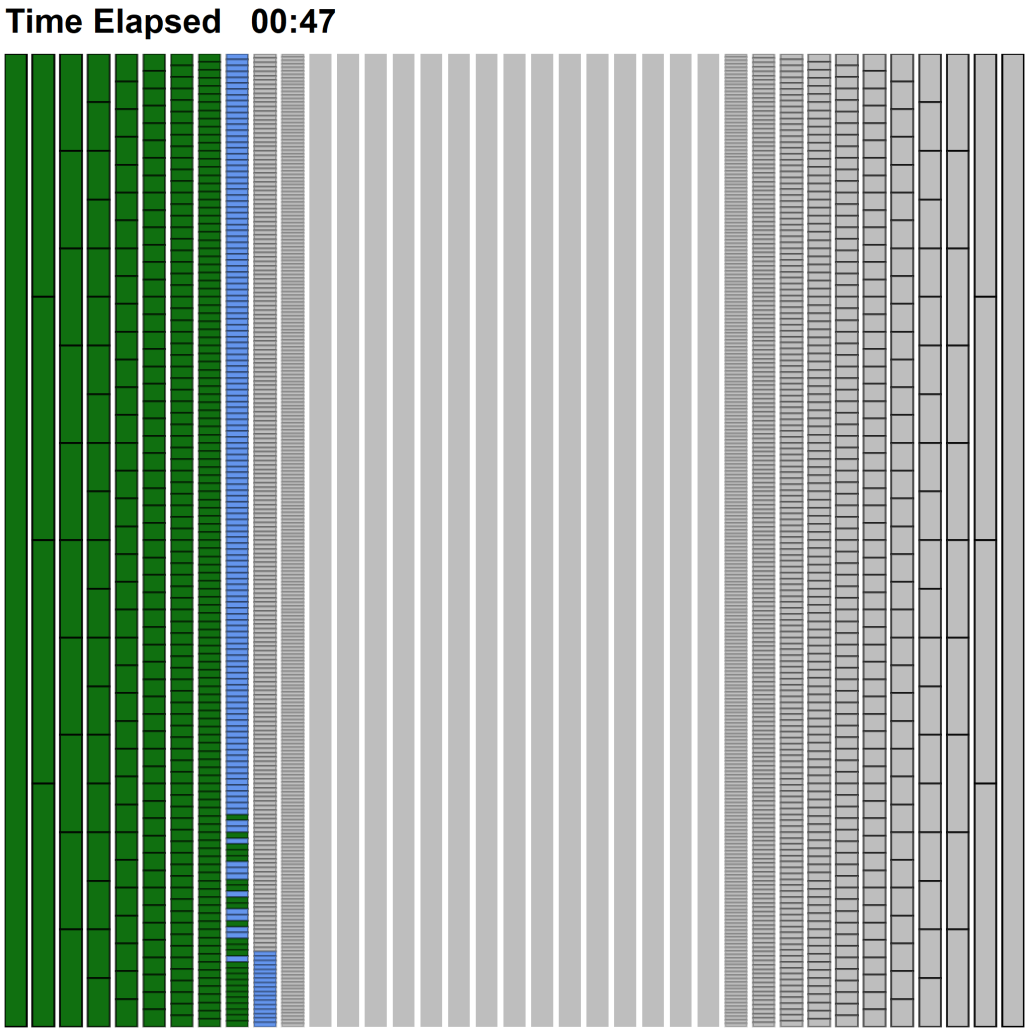
Wavefront



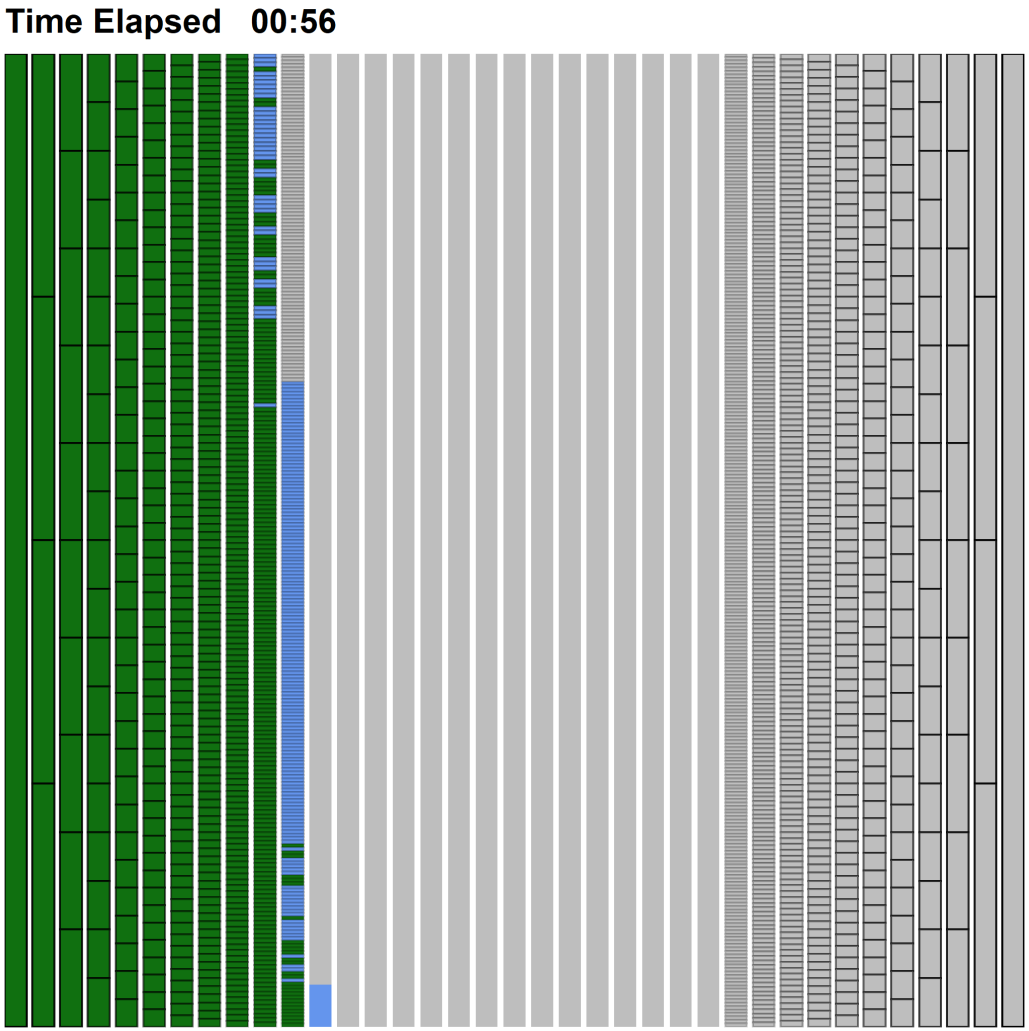
Wavefront



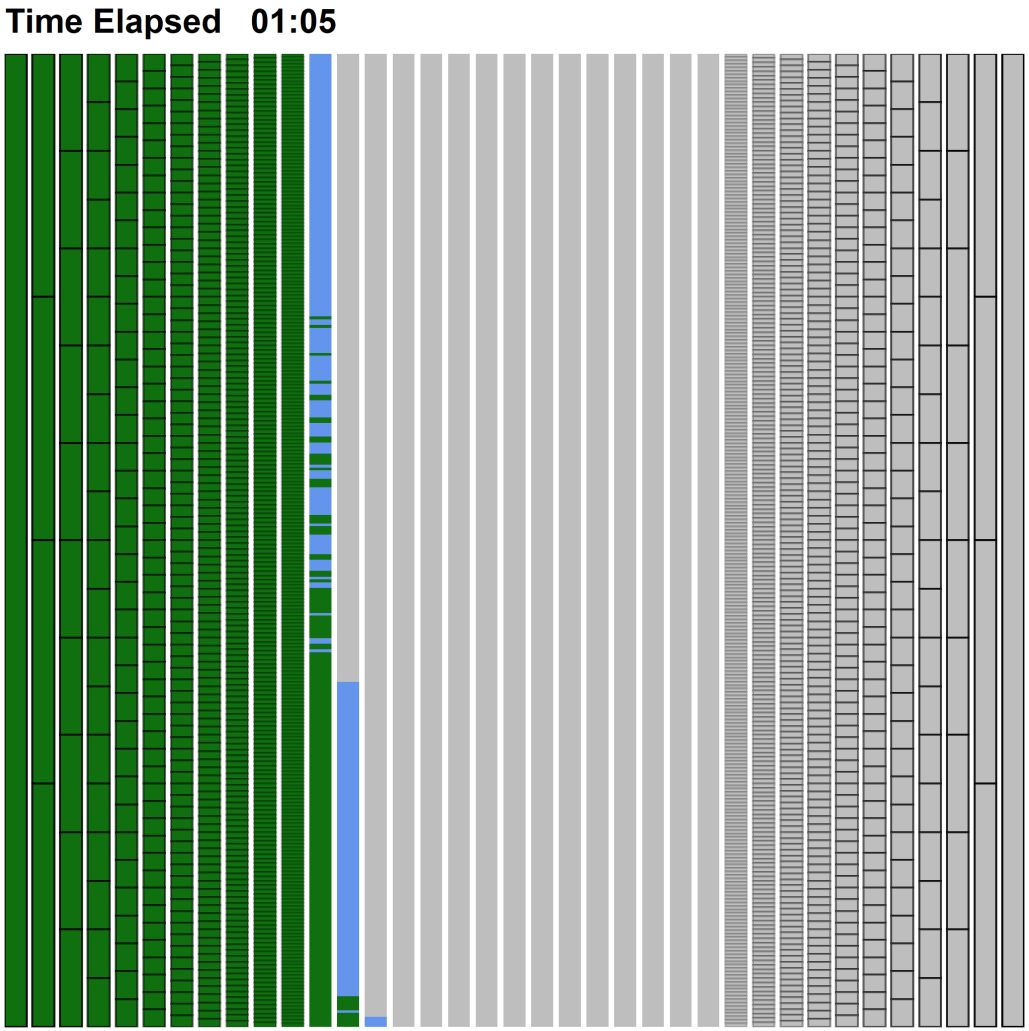
Wavefront



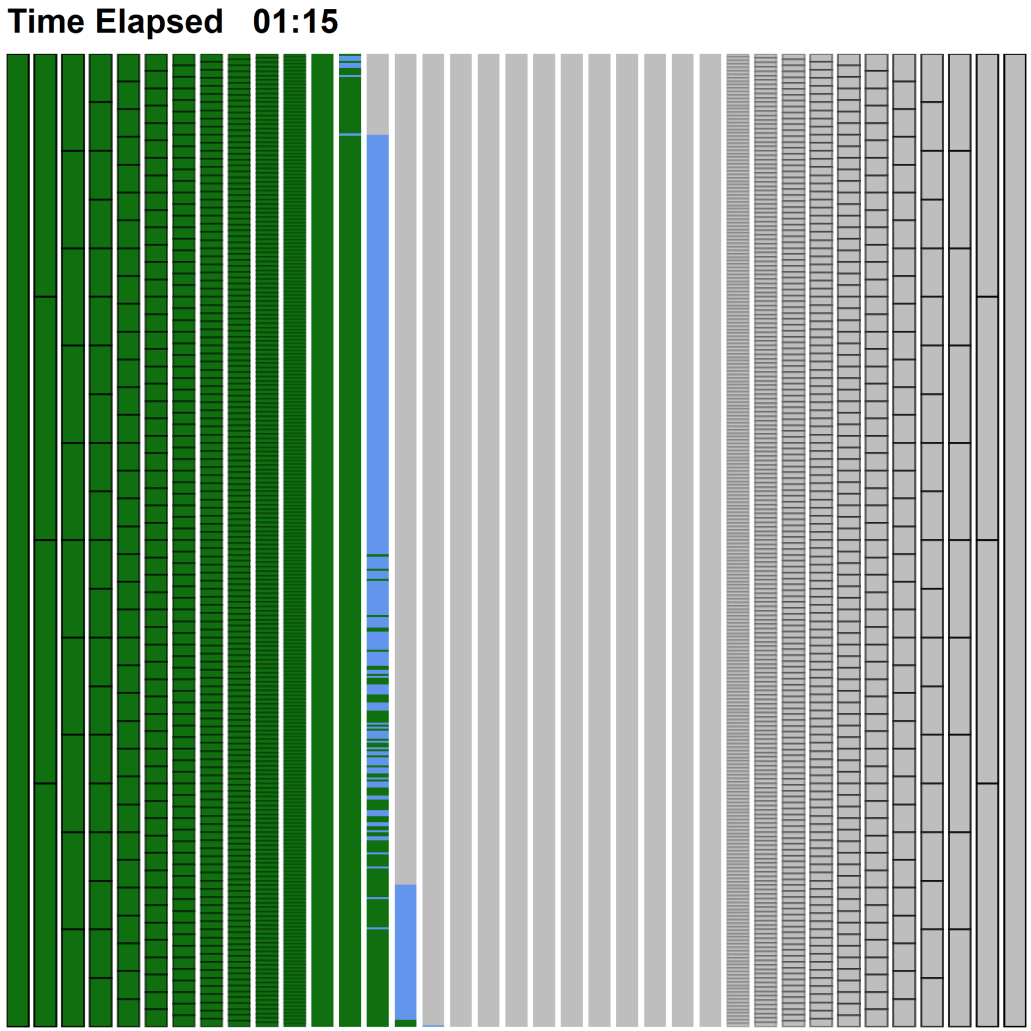
Wavefront



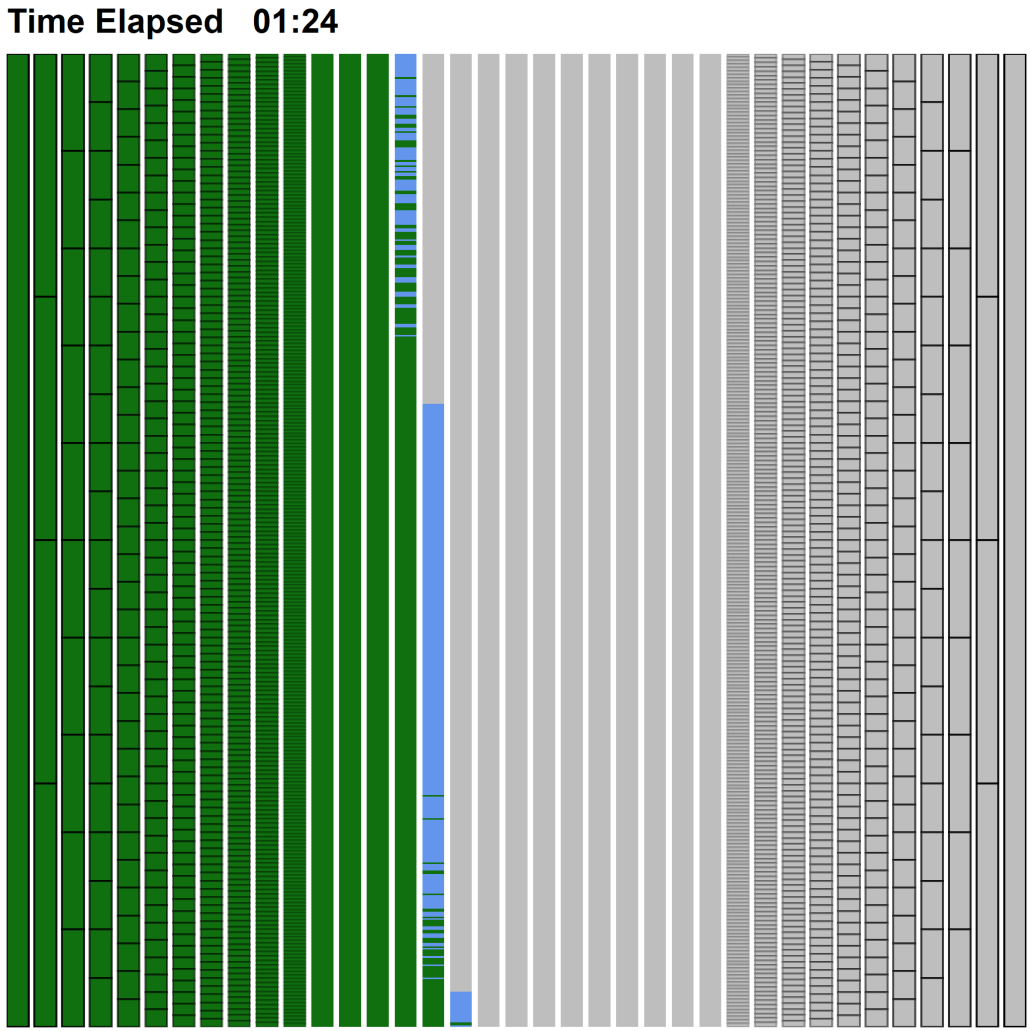
Wavefront



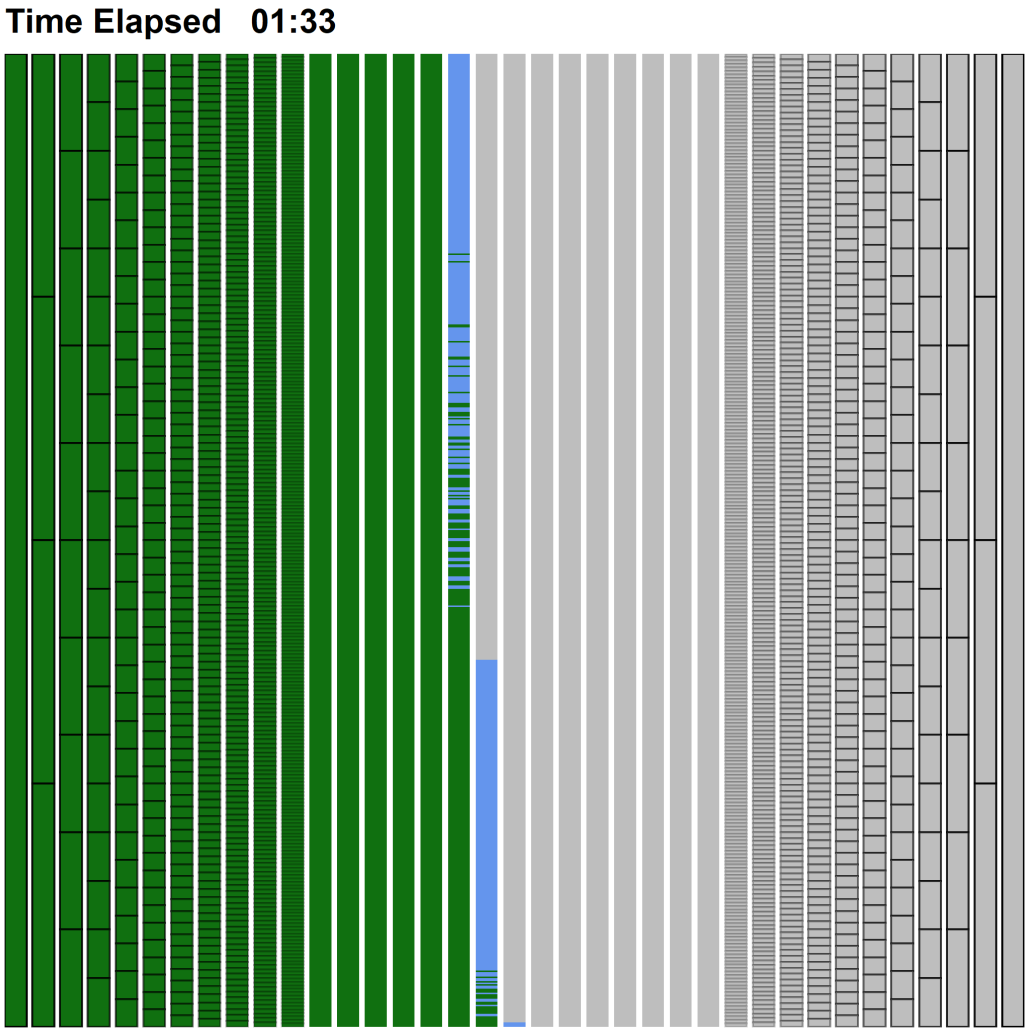
Wavefront



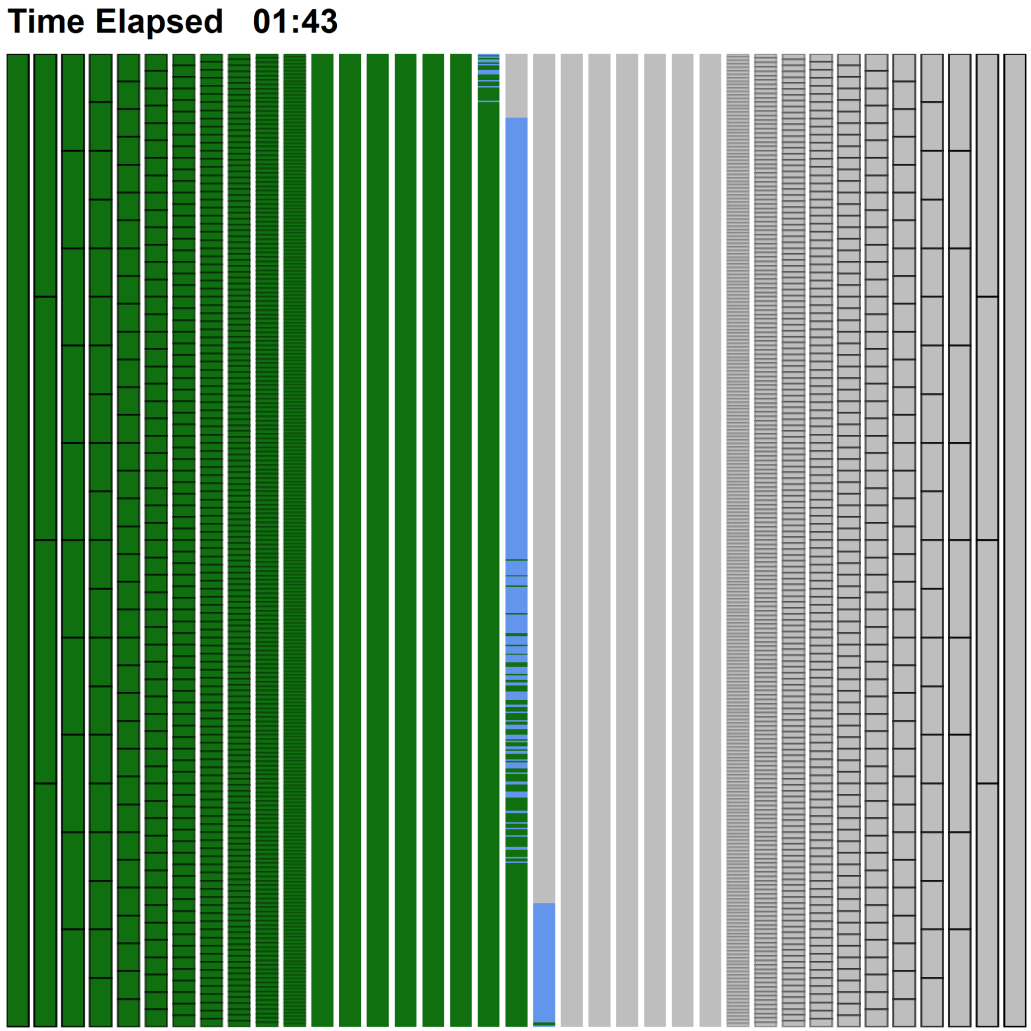
Wavefront



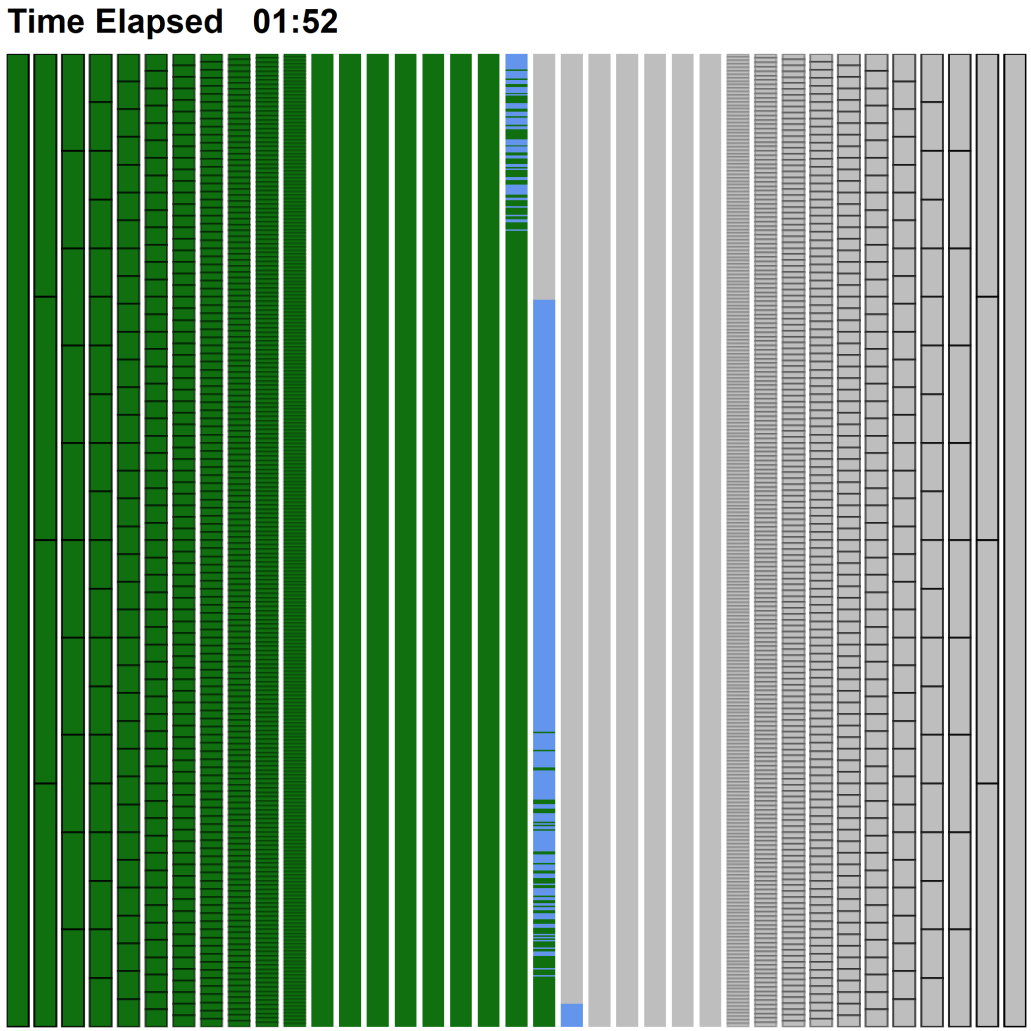
Wavefront



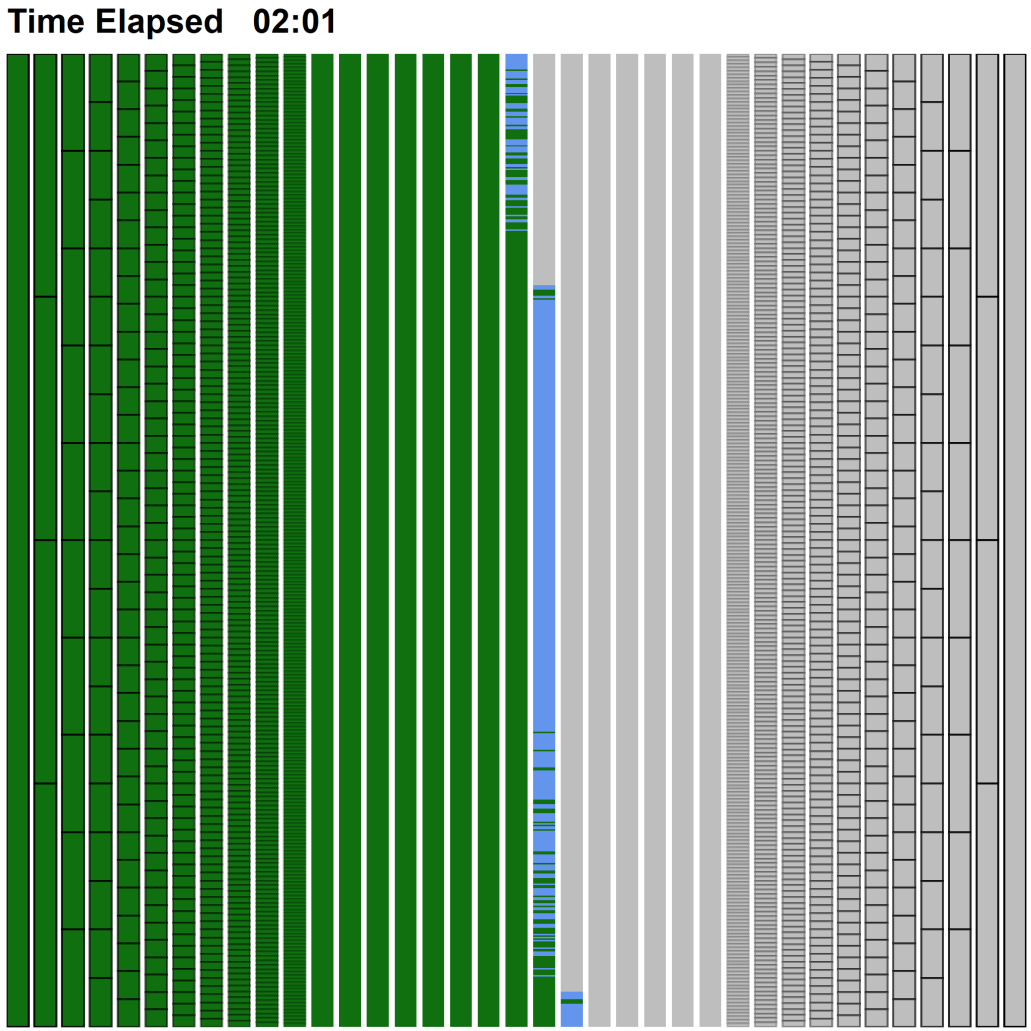
Wavefront



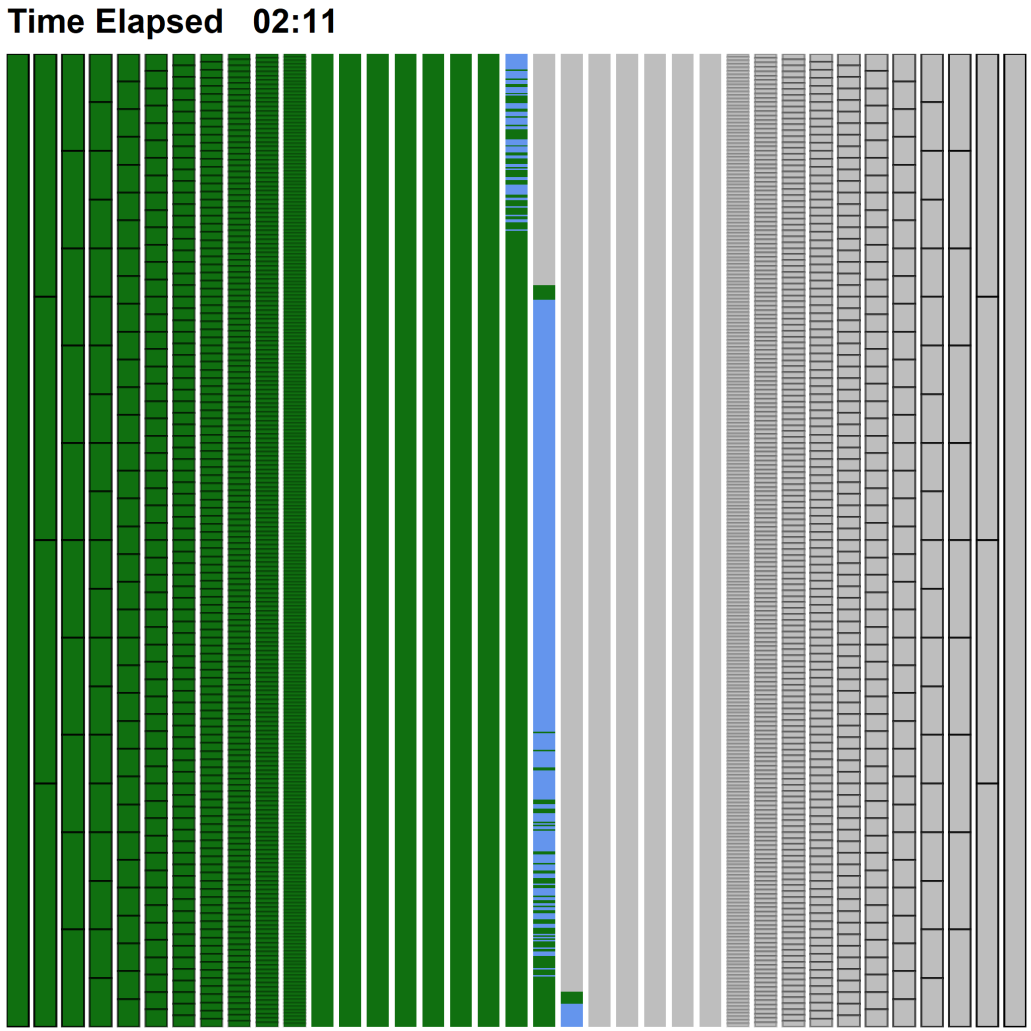
Wavefront



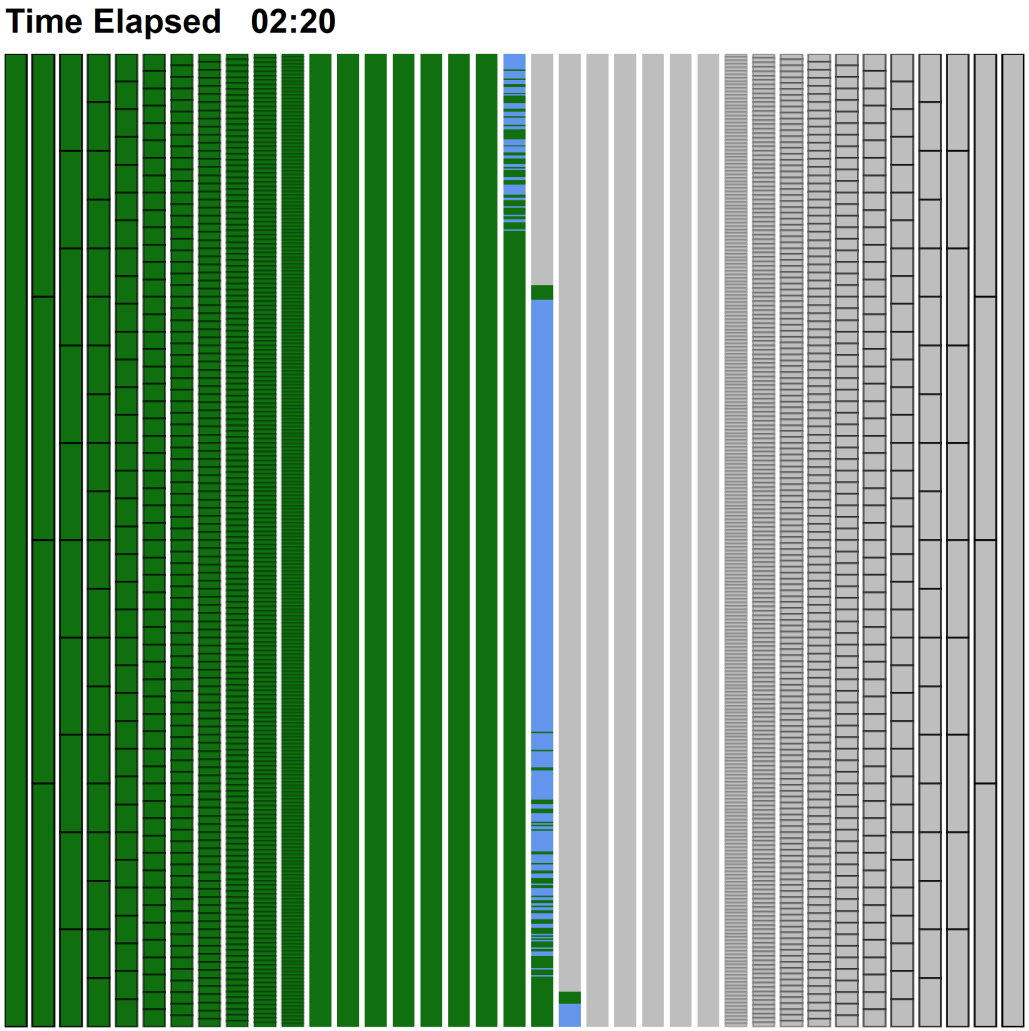
Wavefront



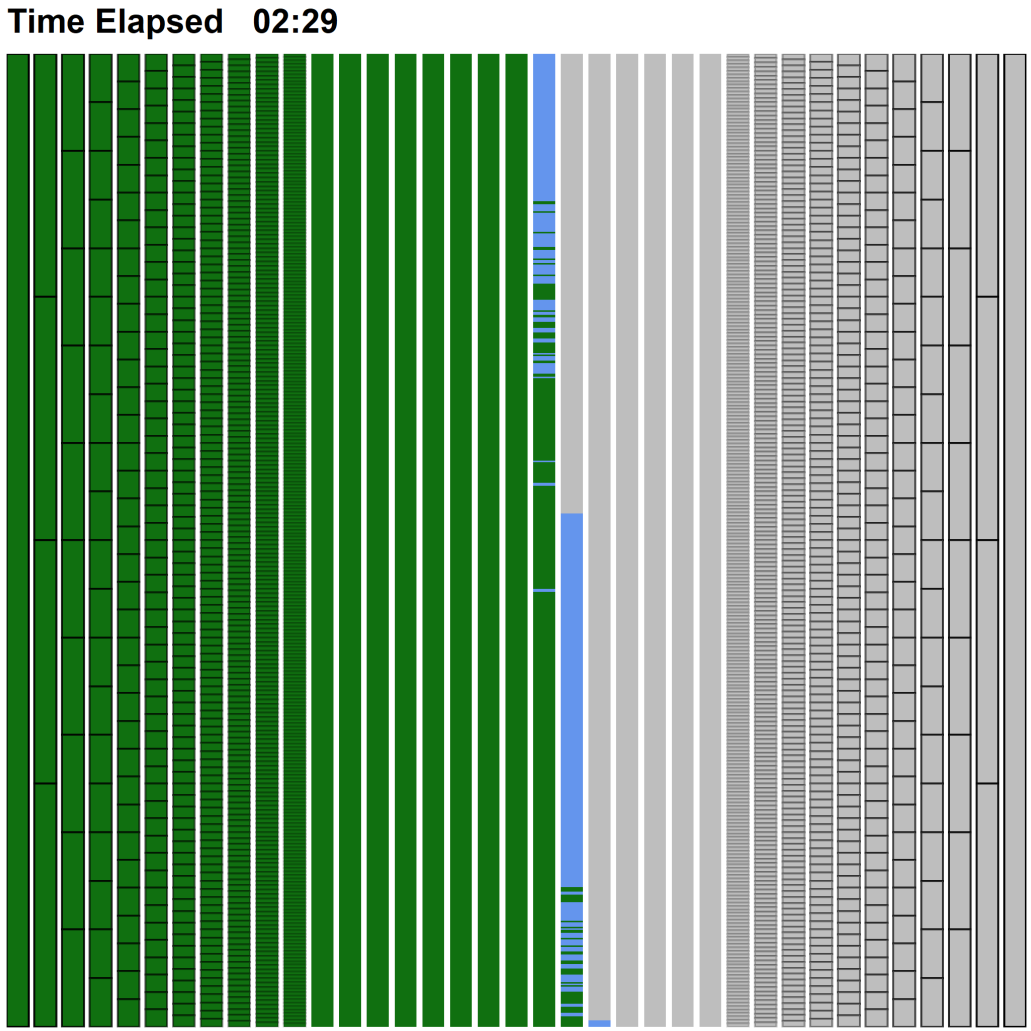
Wavefront



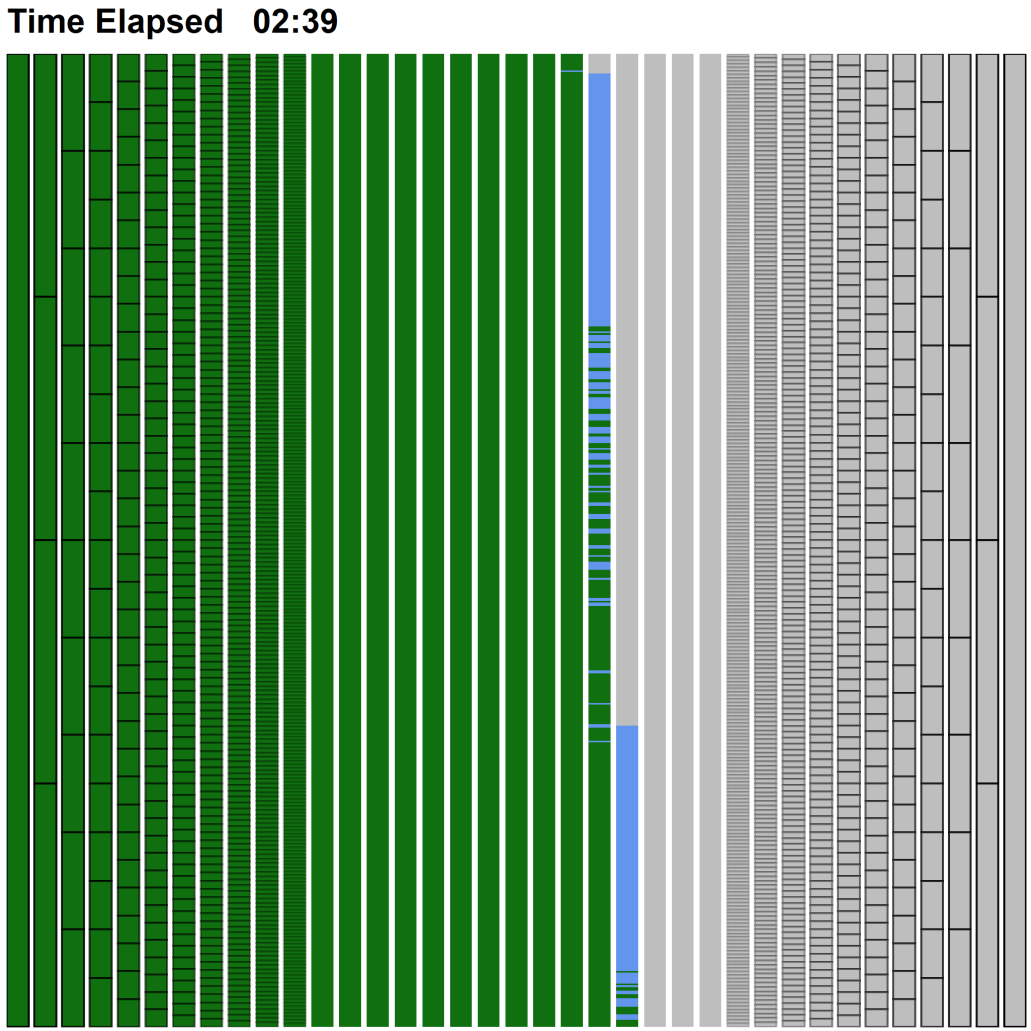
Wavefront



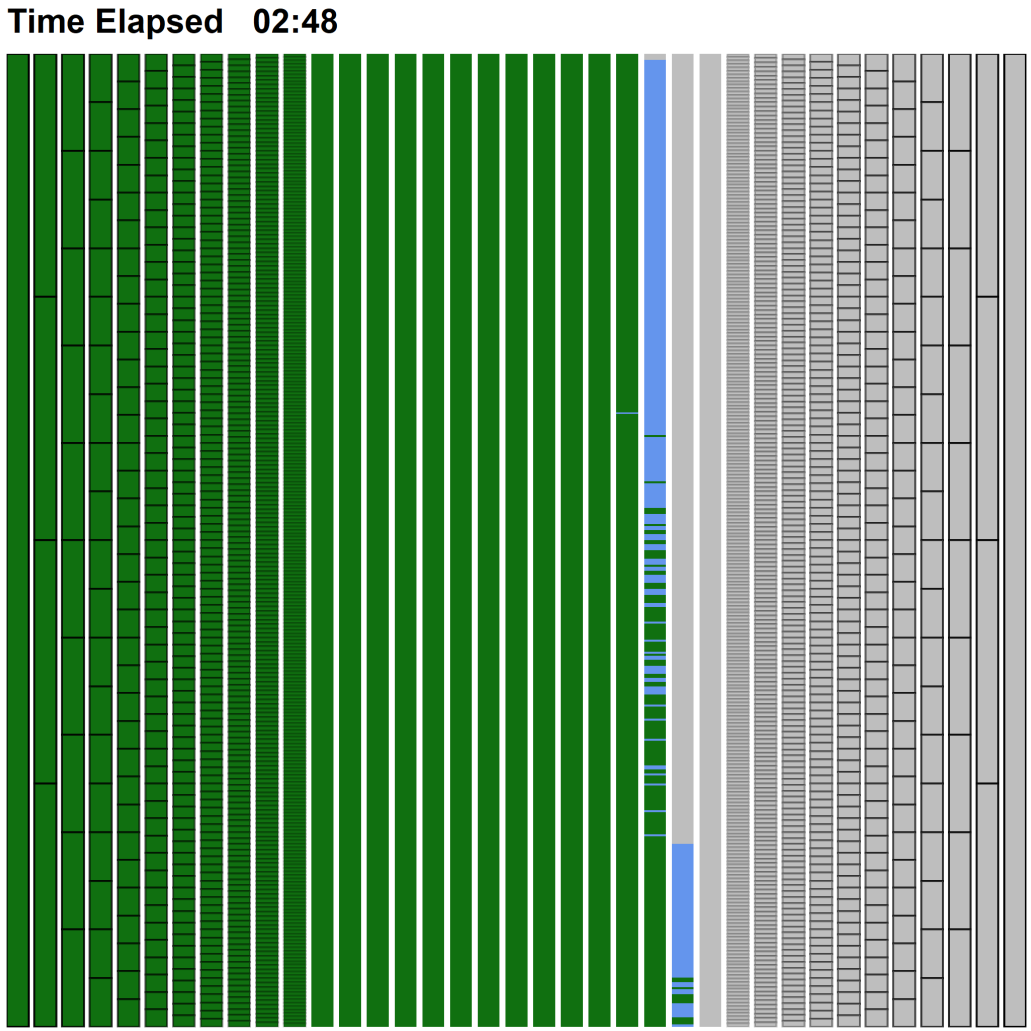
Wavefront



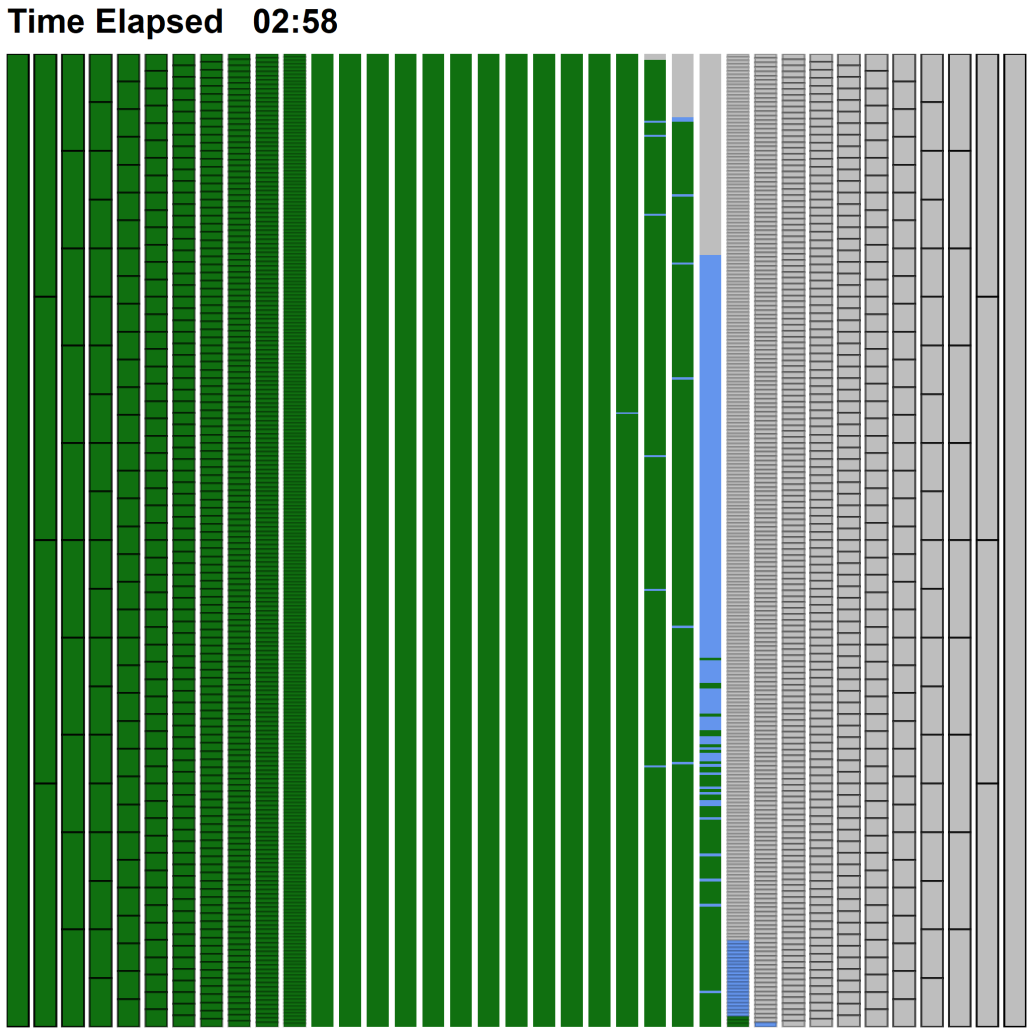
Wavefront



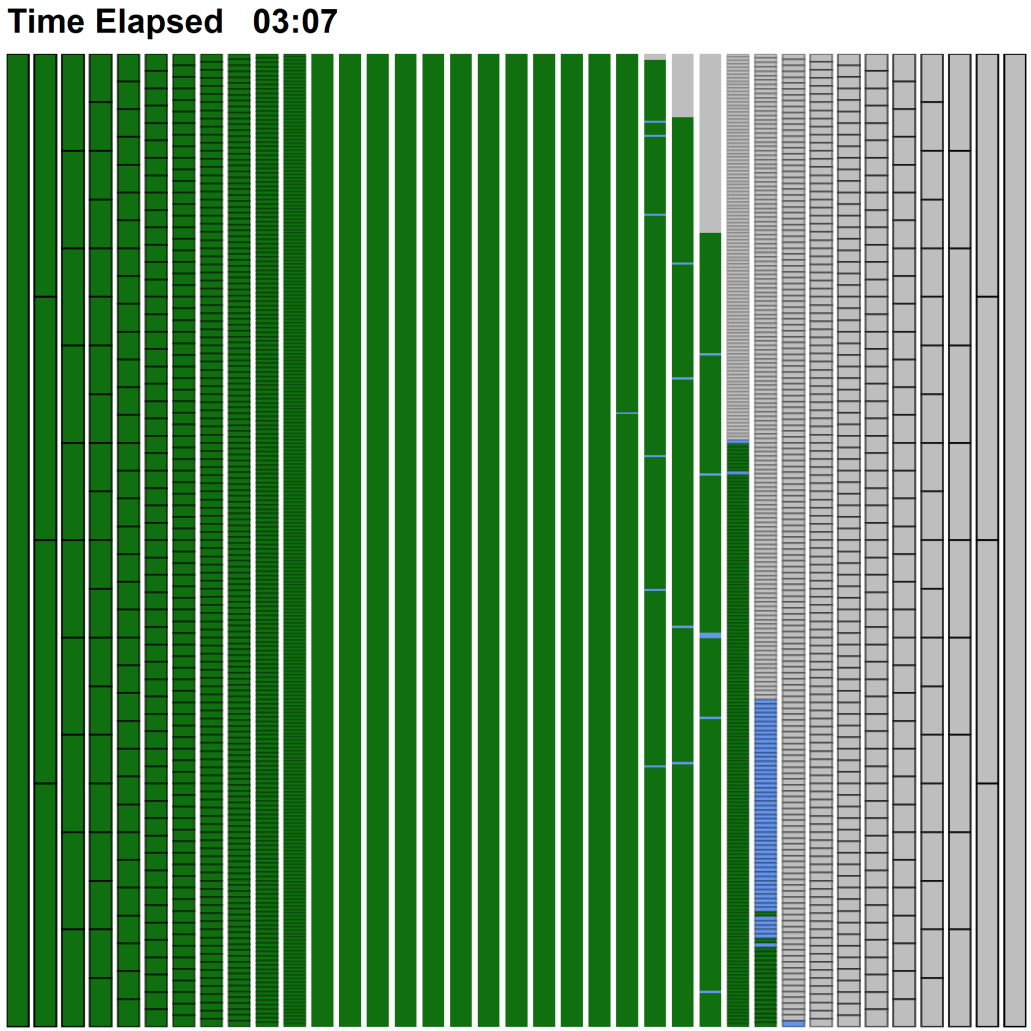
Wavefront



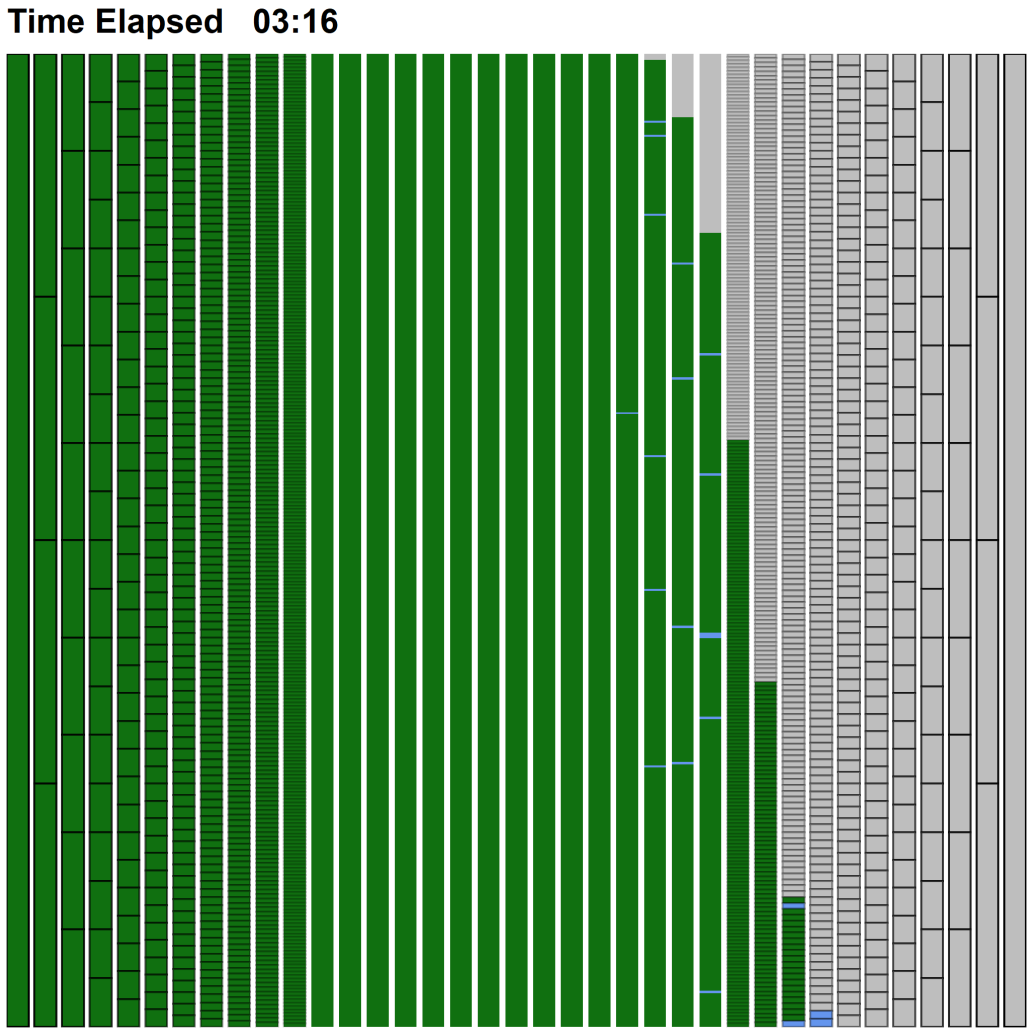
Wavefront



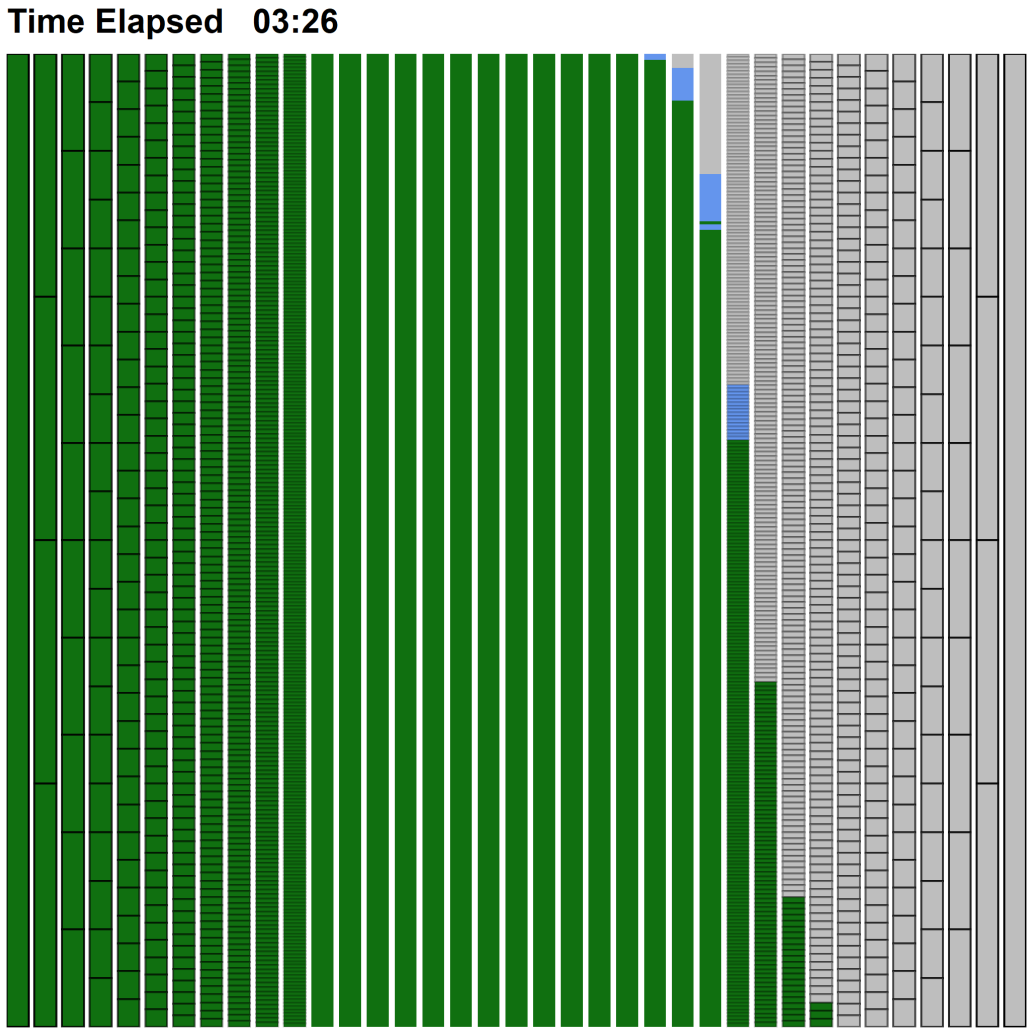
Wavefront



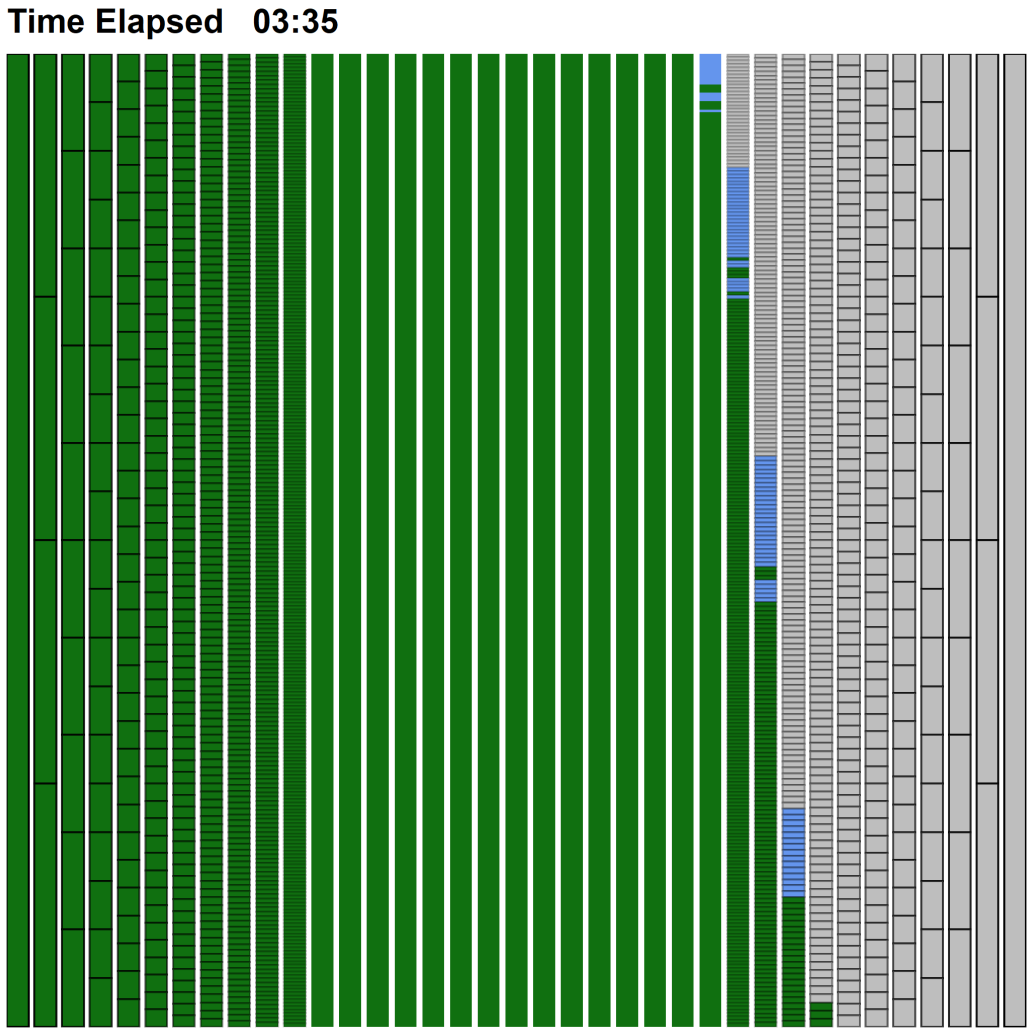
Wavefront



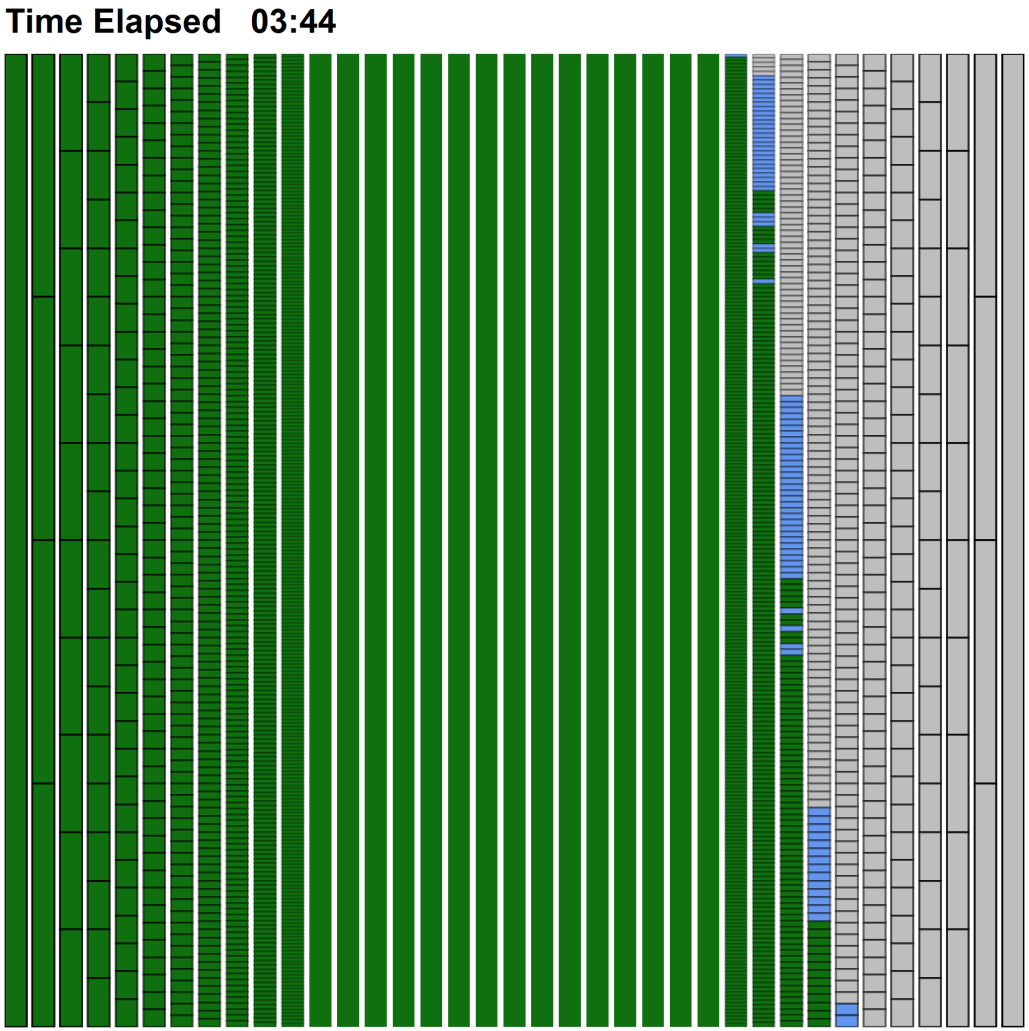
Wavefront



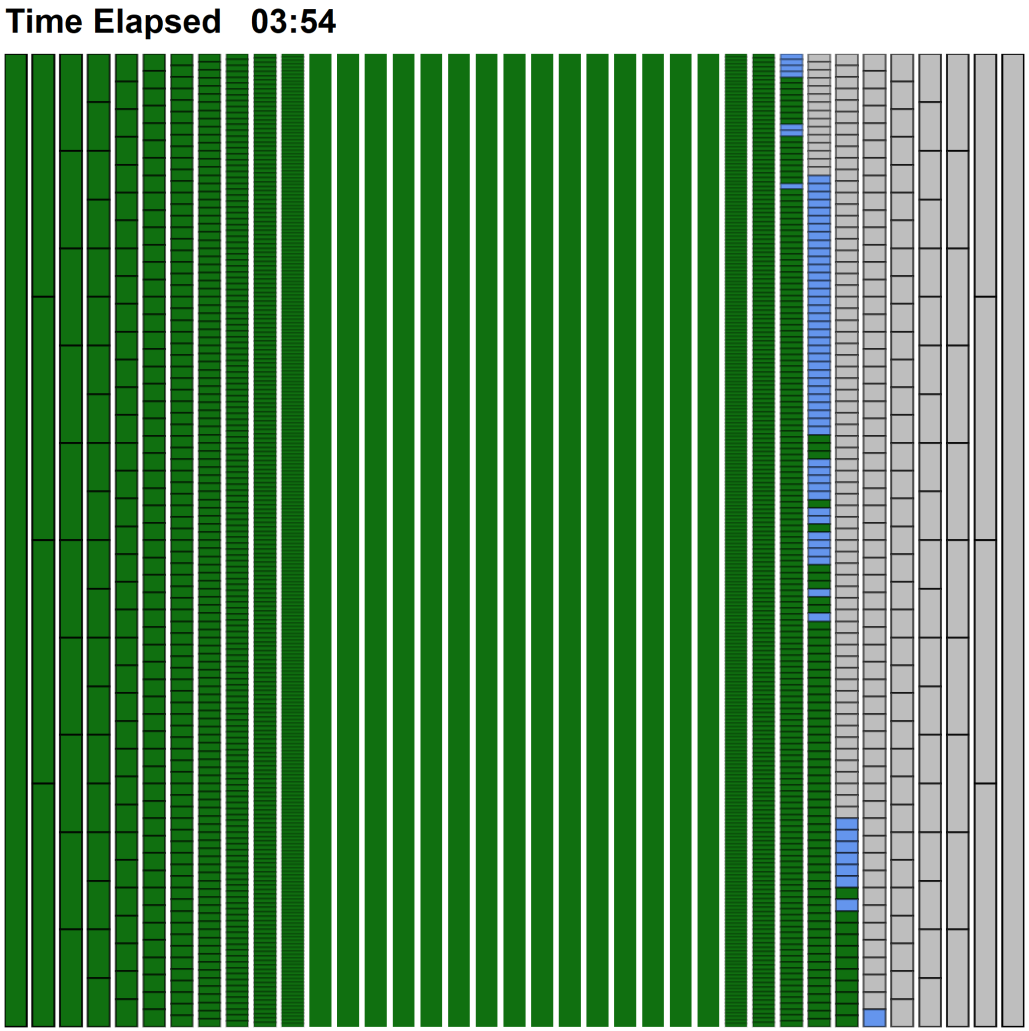
Wavefront



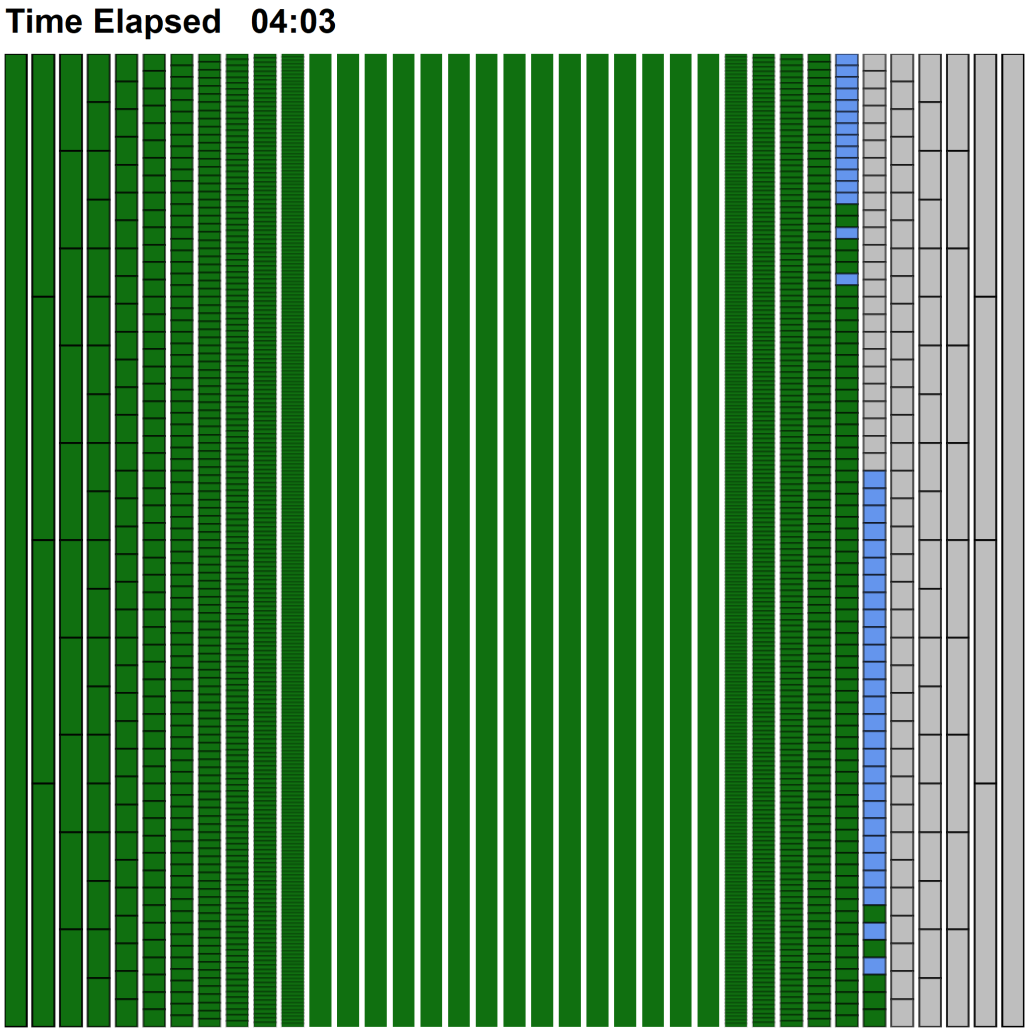
Wavefront



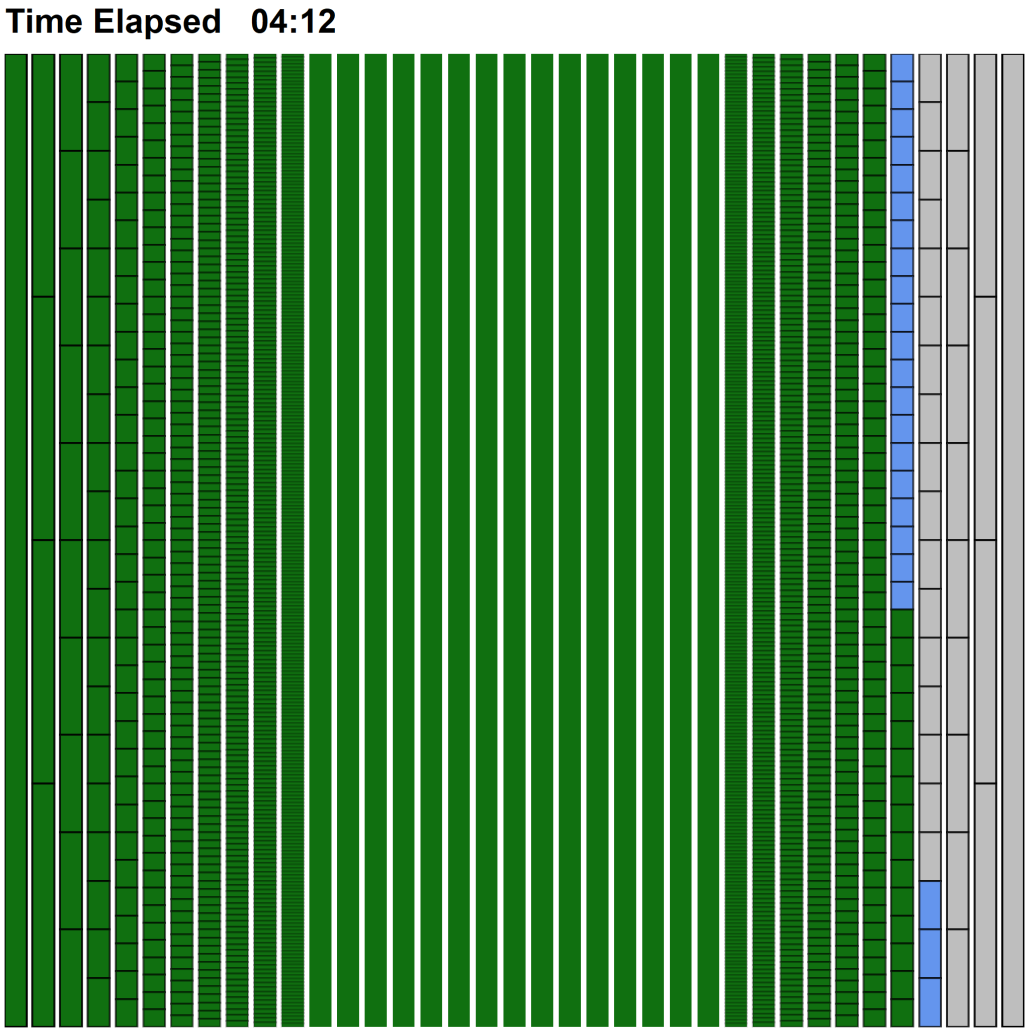
Wavefront



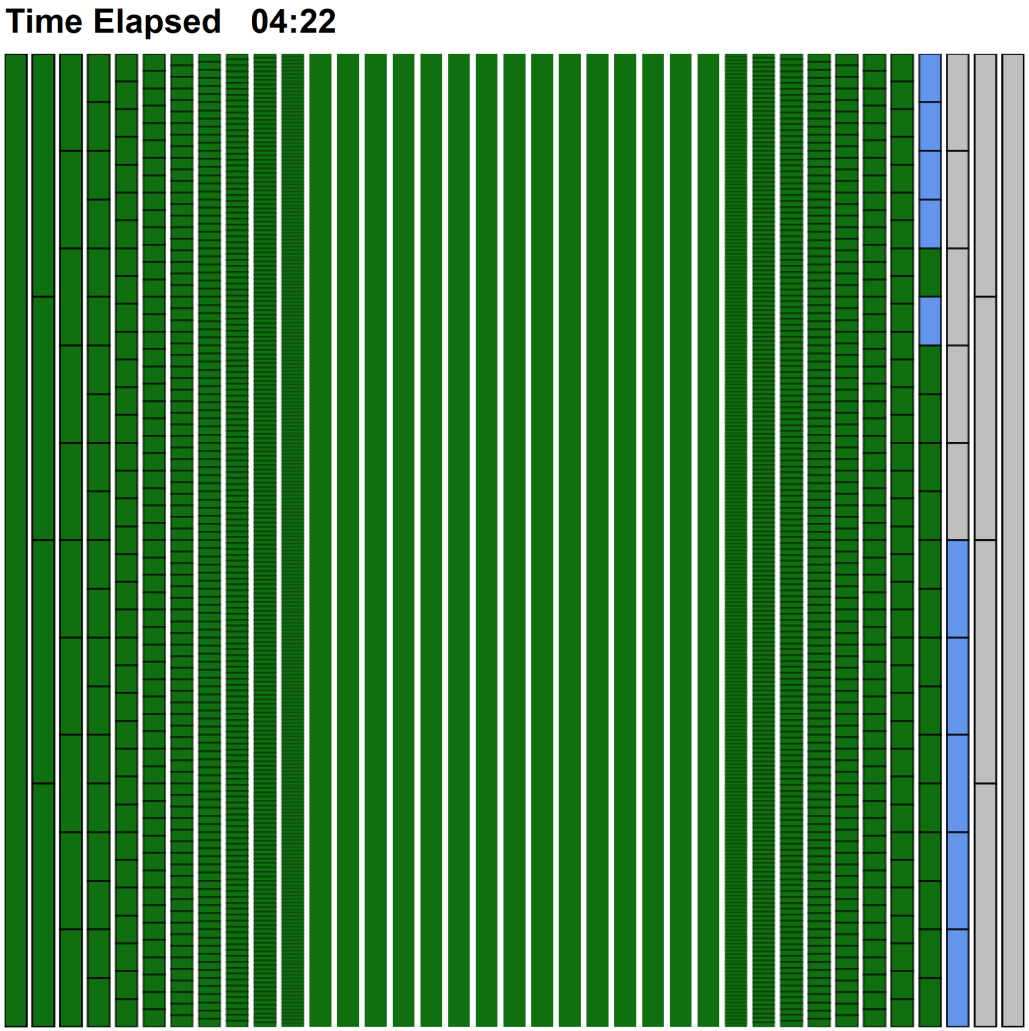
Wavefront



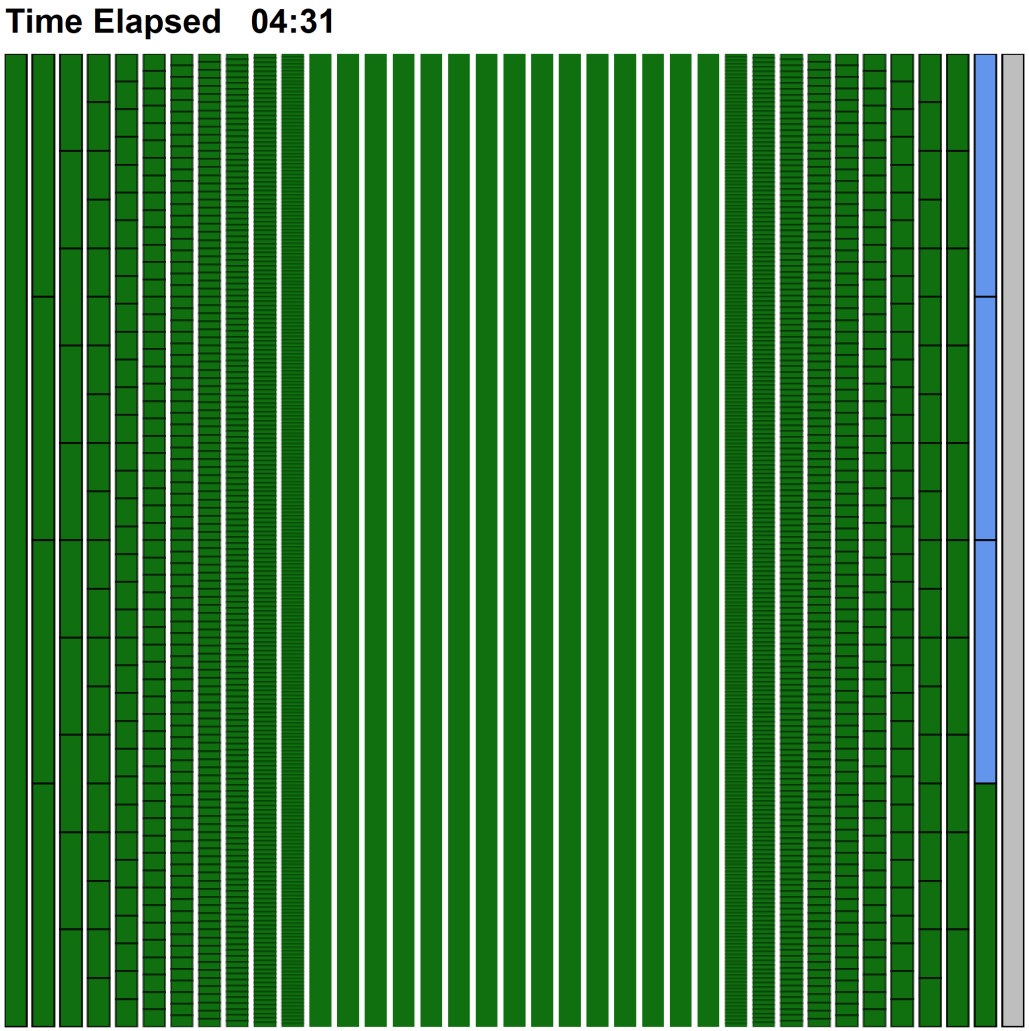
Wavefront



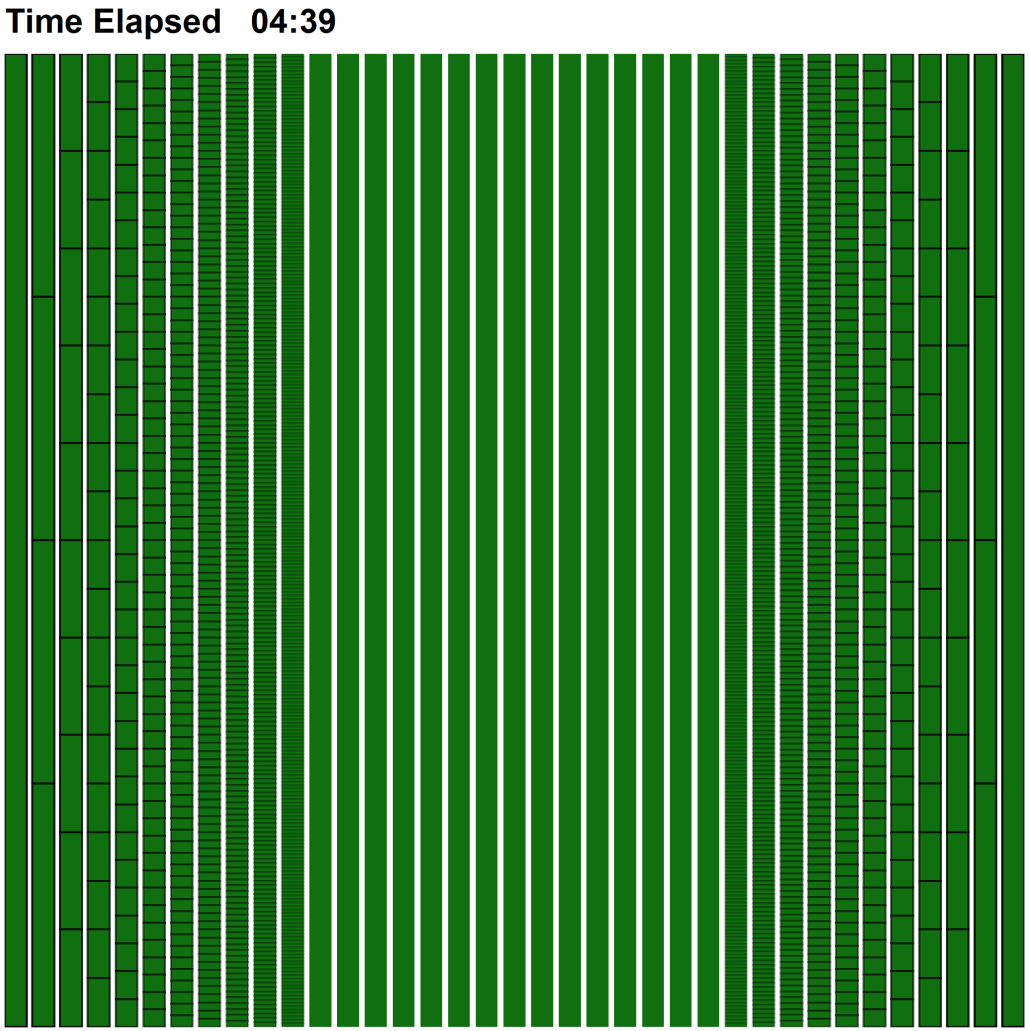
Wavefront



Wavefront



Wavefront



Implementing Wavefront

- HTCondor is not the only job scheduling system; there's also:
  - sdag (Slurm version)
  - Pegasus and Wings (built on DAGMan)
  - CWL
  - Cylc (designed for cyclic workflows, but you can also implement those in DAGMan with RETRY).

Where to Use HTCondor

- CHTC
- Open Science Grid
- JEODPP (European Commission JRC)
- Google Cloud

Dependencies

- Running jobs on another computer can sound difficult. How can you be sure that it will have the programs and files required?
- The solution: Docker containers!
  - (Can also transfer an executable)

Docker

- Docker containers run jobs in the exact same environment every time; as though you went on the machine and installed a certain operating system and all the programs you need.
- But because the host computer already has the Linux kernel, Docker can set up the container quickly and easily (it's generally less than 100 MB).
- MATLAB (with a network license), Mathematica, and even Stata are available in containers, and you can add additional software to them.

Docker Hub

- GitHub for Docker containers
- Free hosting
- Public (so machines can pull your images)
- With HTCondor's Docker universe, the execute nodes pull containers from Docker Hub by default.

R&D Investment and Policy

(A continuous time adaptation of the model in Chapter 7 of Mordecai Kurz's *Capital and Wealth in the Age of Technology*, forthcoming)

- This is a general equilibrium model of optimal R&D investment, which we're hoping to use to study optimal antitrust policy.
- State variable: R&D stock
- Choice variable: R&D investment
- Because larger firms affect the aggregate price more, firms with greater market share face a smaller elasticity of demand (Marshall's "Second Law of Demand," Autor et al., 2020).

R&D Model: Firms and Production

y^B is an outside good with price 1, zeta is aggregate productivity, and v is an investment cost with quadratic adjustment costs. y_t is a CES aggregate produced costlessly from the intermediate goods.

- Production: $$y_{jt} = RD_{jt}^{\gamma}\zeta_t l_{jt}$$
- Outside good production: $$y_t^B = \zeta_t l_t^B$$
- Final good production: $$y_t = \left(\sum_J y_{jt}^{\frac{\chi-1}{\chi}}\right)^{\frac{\chi}{\chi-1}}$$
- Investment cost: $$\nu_{jt} = i_{jt} + \frac{\kappa}{2}\frac{i^2}{RD_{jt}}$$
- Profit: $$\Pi_{jt} = y_{jt} - w_t l_{jt} - \nu_{jt}$$

R&D Model: Consumers' Problem

- Labor aggregation: $l_t = \sum_J l_{jt} + l_t^B$
- Utility: $u_t(c_t^B, c_t, l_t) = \log(c_t^B) + B\log(c_t) - \frac{H}{1+\eta} l_t^{1+\eta}$
- Income: $G_t = w_t l_t + \sum_J \Pi_{jt} - \sum_J \nu_{jt}$
- Optimization: $\max u_t(c_t^B, c_t, l_t) \text{ s.t. } P_t c_t + c_t^B \leq G_t$
- FOCs: $c_t^B = \frac{1}{B+1} G_t, c_t = \frac{B}{B+1} G_t, l_t = \frac{w_t}{H c_t^B}$
- Price index: $P_t = (\sum_J p_{jt}^{1-\chi})^{\frac{1}{1-\chi}}$
- Demand: $D(y_{jt}, p_{jt}) = y_{jt} - (\frac{p_{jt}}{P_t})^{-\chi} c_t$
- Market clearing: $c_t^B = y_t^B - i_t, c_t = y_t$

R&D Model: Dynamics

- R&D stock transition: $\mathbf{RD}_t = \mathbf{RD}_0 + \epsilon$
- Jumps: $P(\epsilon_j = n) = \dfrac{(\int_0^t \phi_{jt} dt)^n \exp(-\int_0^t \phi_{jt} dt)}{n!}$
- Transition hazard rate: $\phi_{jt} = i_{jt} Z (\dfrac{RD_{jt}}{\sum_J RD_{jt}})^{\Lambda}$

R&D Model: Solution

We get the optimal price in closed form, so we just solve for the dynamically optimal investment (we rule out collusion and other equilibria that aren't Markov perfect). We take ρ to be exogenous.

- Elasticity of demand: $\theta_{jt} = \chi(1 - (\frac{p_{jt}}{P_t})^{\chi})$
- Pricing: $w_t = \zeta_t, p_{jt} = \frac{(1-\chi)\theta_{jt} - \chi}{(1-\chi)\theta_{jt}} \frac{1}{RD_{jt}^{\gamma}}$
- Investment FOC:

$$0 = (V_{jt}(\mathbf{RD}+\mathbb{I}) - V_{jt}(\mathbf{RD}))\frac{\phi_{jt}}{i_{jt}} - 1 - \kappa\frac{i_{jt}}{RD_{jt}}$$

- Bellman equation:

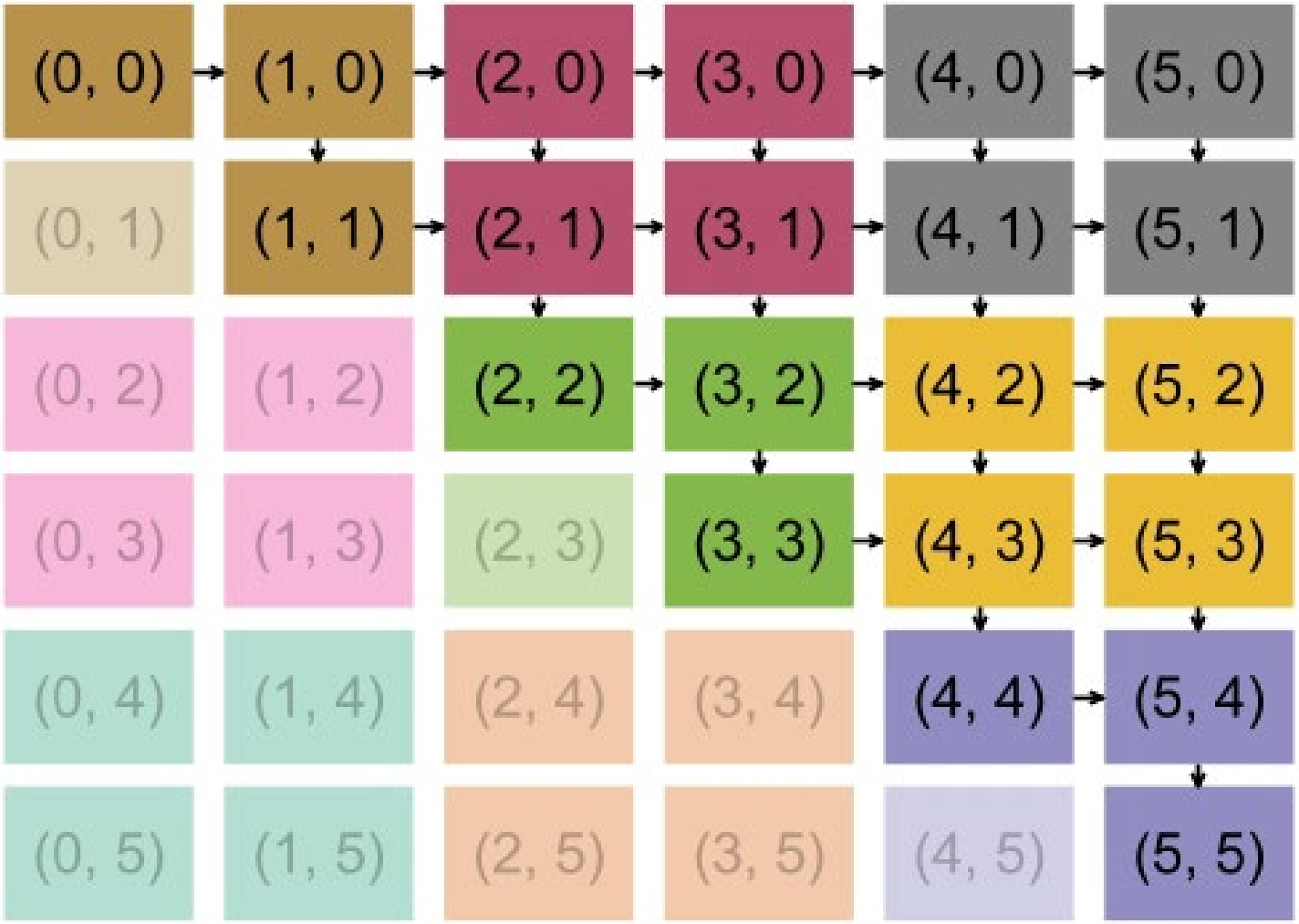
$$\rho V_{jt}(\mathbf{RD}) = \Pi_{jt} + (V_{jt}(\mathbf{RD}+\mathbb{I}) - V_{jt}(\mathbf{RD}))\phi_{jt}$$

Symmetry

- Every firm starts out the same; the differences that arise are only due to the stochastic, self-reinforcing nature of the R&D process.
- With identical firms, the state (0, 1) has the same solution as the state (1, 0). And in higher dimensions, the state (0, 1, 2, 3) has the same solution as (3, 0, 2, 1) and the 22 other permutations.
- The states to compute form a simplex, and pin down solutions for the rest of the cube, vastly reducing the computational burden.

Symmetry

Same parallel blocks (or, rather, their projection onto the simplex)



Future Work

- Cyclic games
  - Idea: path follow from the unidirectional solution
  - Iterating on the transition matrix moving with one direction of travel is operator decomposition

$$0 = f(V_{j0} \dots V_{jT}, x_{j0} \dots x_{jT}) = g(V_{j0} \dots V_{jc}, x_{j0} \dots x_{jc}, h(V_{jc} \dots V_{jT}, x_{jc} \dots x_{jT}))$$

$$g^{-1}(0) = \{V_{j0} \dots V_{jc}, x_{j0} \dots x_{jc}, h(V_{jc} \dots V_{jT}, x_{jc} \dots x_{jT}))\}$$

$$g^{-1}(0) \approx \{V_{j0} \dots V_{jc}, x_{j0} \dots x_{jc}, h(\hat{V}_{jc} \dots \hat{V}_{jT}, \hat{x}_{jc} \dots \hat{x}_{jT}))\}$$

Gauss-Seidel-style; good convergence properties

- Challenges:
  - Equilibrium multiplicity
  - Dead ends

Conclusion

- Grid computing and workflow languages allow the parallelization of dynamic programming problems and dynamic games that can otherwise be difficult to parallelize.
- HTCondor and DAGMan are useful for implementing this.