Polychrony and the Process View of Computation

Colin Klein*†

Some realistic models of neural spiking take into account spike timing, yet the practical relevance of spike timing is often unclear. In Eugene Izhikevich's model, timing plays a crucial role by allowing for the natural formation of polychronous circuits. In such circuits, individual elements may figure in a number of distinct assemblies, their role in each determined by their timing relative to other neurons. I show that this reflects a distinct organizational principle from notions of pluripotency, redundancy, or reuse and argue that properly understanding this phenomenon requires a shift to a time-sensitive, process-based view of computation.

1. Introduction. Familiar neural network models abstract away from temporal factors. The connectivity between neurons matters, but not the time it takes for signals to propagate through the network. Everyone knows that this is an omission. Neural signals can take a long time to propagate, on a timescale that can be orders of magnitude faster than the chemical processes that drive synapses. This places strong engineering constraints on the design of brains (Sterling and Laughlin 2015).

Neural network models that include propagation delays and associated temporal factors are less familiar to most philosophers. They are interesting for two reasons. First, such networks are scientifically interesting: including the impact of actual time allows such networks to capture a variety of biological phenomena. Second, they are interesting for philosophy of computation. Computations are processes and unfold in time (Smith 1996). For many purposes, computational analyses can ignore time. Yet these networks show

*To contact the author, please write to: School of Philosophy, Australian National University, Canberra ACT 2601, Australia; e-mail: colin.klein@anu.edu.au.

†This work was supported by Australian Research Council grant FT140100422. Thanks to Felipe de Brigard and two anonymous reviewers for helpful feedback on previous drafts.

Philosophy of Science, 87 (December 2020) pp. 1140–1149. 0031-8248/2020/8705-0031$10.00
Copyright 2020 by the Philosophy of Science Association. All rights reserved.

https://doi.org/10.1086/710613 Published online by Cambridge University Press

ways in which time matters, in a computational sense, for biologically realistic computations. That is true even though the networks themselves are considered at a relatively high level of abstraction.

What follows sketches out the first point on the way to explaining the second. I begin by introducing Eugene Izhikevich's concept of a *polychronous* network, that is, a network in which each neuron supports multiple distinct computations by belonging to dynamic groups on different timescales. I give some details of modeled polychronous networks, focusing on how these interesting properties are a consequence of simple learning and activation dynamics. I suggest a refined definition of polychrony and then turn to the computational upshot, arguing that polychronous networks emphasize a difference between *timeless* and *process-based* senses of computation. I suggest that polychrony requires thinking carefully about processes and that philosophers ought to pay more attention to the latter.

2. Polychronous Networks. Izhikevich (2006) introduced the term 'polychronous' to describe the self-organizing features of certain simulated neural networks. The networks themselves are characterized by three important features, each inspired by actual cortical circuits (Izhikevich, Gally, and Edelman 2004). First, they involve spiking neurons with precise timing. Second, there are finite and nonuniform propagation delays between neurons in the network. Third, the networks incorporate spike-timing-dependent plasticity (STDP). STDP strengthens synaptic weights when input occurs before the firing of the postsynaptic neuron and weakens weights when they occur after. Both effects are strongest when input and firing are close in time.

Setting the initial weights randomly and letting the network settle, the resulting dynamics have a number of interesting features. Chief among them is the self-organization of groups of neurons that fire in a regular sequence. These groups are formed and maintained by STDP. Massive changes to synaptic weights will eliminate groups, but groups are stable and self-healing in the face of perturbations to individual synapses. Some of the groups formed are remarkably stable, lasting through the entire simulation. Although I will mostly discuss the effects seen in networks with stochastic firing, driving portions of the same networks using strong excitatory stimuli also forms similar networks with further interesting properties.

Stable groups often overlap. A single neuron can be a member of multiple different groups; in Izhikevich's (2006) simulation, an average group had 25 neurons and an average neuron was a member of 131 groups. It is possible for there to be far more groups in the network than there are neurons, because when different groups are active at different times they do not interfere with one another (257).

As figure 1 illustrates, the sequence of firing is crucial to generating overlap. Figure 1*a* emphasizes how differing propagation delays matter. Since

https://doi.org/10.1086/710613 Published online by Cambridge University Press

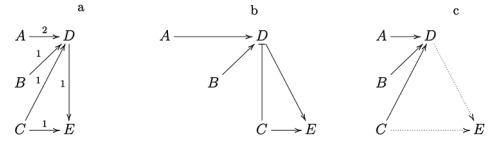


Figure 1. Simple polychronous network. *a*, Static structure. Numbers indicate propagation delays. We assume downstream neurons need simultaneous input from two neurons to fire. *b* and *c*, Two ways of unravelling the network in time. The *A*–*D* influence also arrives too late and is downgraded.

signals take time to travel from one part of the network to another, which downstream neurons are influenced depends on the timing of earlier neurons in the network. Figure 1*b* shows one possibility: firing *A*, *B*, and *C* in order will fire *D* and *E*. The influence of *C* on *D* arrives too late, however, and so STDP ensures that the synapse is suppressed in future rounds. In figure 1*c* a different temporal pattern results in a different outcome: the simultaneous firing of *A*–*C* will fire *D*, but now *E* will remain silent because the influence from *C* and *D* arrives at different time steps.

Since different polychronous processes can overlap, merely ordinal information is not enough to determine the time evolution of a polychronous system. Consider the system depicted in figure 2. Assuming we need simultaneous input to drive a neuron, there are two polychronous processes possible in this system. Using subscripts to indicate the time of firing, $\{A_{t=0}, B_{t=1}, C_{t=2}, D_{t=3}\}$ and $\{A_{t=0}, B_{t=2}, D_{t=3}\}$ are both possible processes. (Note that in the former, *D* is driven by the firing of *C* and *A*, while in the latter

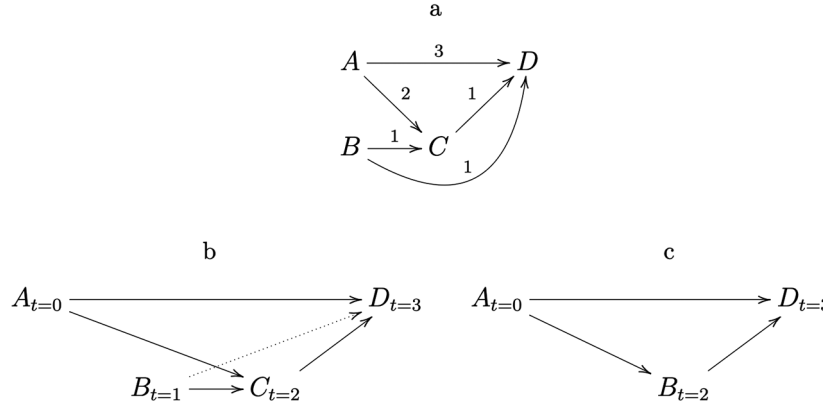


Figure 2. Simple system in which order of inputs alone does not determine the firing path. *a*, Static structure. *b* and *c*, Two ways of unfolding this network in time. Subscripts show time of firing, and dotted arrows are paths that arrive at the wrong time to influence the downstream neuron.

https://doi.org/10.1086/710613 Published online by Cambridge University Press

it is driven by *A* and *B*.) So just knowing whether *A* and *B* obtained, and the order in which they obtained, is not enough to know whether they are part of the former process or the latter one. Which is to say that there are two computational processes that overlap, and that cannot be distinguished on the basis of merely ordinal information about inputs.

Of course, there is an easy way to solve this question: by looking at the actual times that *A* and *B* took place. But that is to acknowledge that merely ordinal information is not enough to distinguish the groups; we need to look at the actual time at which different parts of the process happened. That, I submit, is the key to polychrony and its consequences.

3. Defining Polychrony

3.1. The Core Idea. Izhikevich, in explaining his choice of terms, writes that "since the firings of these neurons are not synchronous but time-locked to each other, we refer to such groups as polychronous, where *poly* means many and *chronous* means *time* or *clock* in Greek" (2006, 249). He distinguishes polychrony from asynchrony, because the latter "usually describes noisy, random, nonsynchronous events" (249).

The point about synchrony strikes me as slightly off. What is important is not really synchrony or time locking of individual groups of neurons. Instead, the key idea is the relationship between different groups of neurons and the idea that they run on 'different clocks'. Consider the possible set of relationships between two different processes. Synchrony, or any other stable time ordering, is a contingent relationship. (I drop the qualifier, from here on, but talk of synchrony is meant to be a shorthand for a variety of stable relationships.) Processes can also be *synchronized* when there are mechanisms that ensure that processes are kept in a reliable relationship with one another and are synchronized when that happens. Conversely, unsynchronized or desynchronized processes could be synchronized but are not.

There is also a third possible relationship. For some pairs of processes, we effectively cannot talk about the temporal relationships between them because they run reliably but on *incommensurate timescales*.

That is, I submit, the key feature of polychrony. Polychronous networks cannot be synchronized (in some important sense of 'cannot' to be fleshed out), because from the point of view of the individual networks, we cannot talk meaningfully about the temporal relationships between networks. They run, as Izhikevich suggests, on too many clocks. That might seem odd, because time is time. I think the notion is less odd than it might seem at first.

3.2. Refinements. To cash out this definition of polychrony, I start with a nontemporal analogy. Suppose the university has a committee for appointing visiting fellows and a committee for course approvals. Each has its laws

https://doi.org/10.1086/710613 Published online by Cambridge University Press

and rules. Neither answers to the other, and they have no formal reason to interact. Officially speaking, one committee need not care about what another committee does, except and unless the business of the other committee is represented as part of that committee's agenda. To appropriate a term, each committee forms a *monad* with respect to their rules and regulations: what is possible qua committee is fully spelled out by their terms of reference.[1]

Now, suppose that an unlucky group of junior faculty has been appointed to both committees. Given the quorum rules for each committee and their contingent compositions, they interfere with one another; they cannot meet at once, for example. Officially speaking, the committees are distinct. Actually speaking, since committees are instantiated processes, they can interfere in ways outside their official scope.

Of course, individuals can step outside their monadic assignments and recognize the interference. From the point of view of the committees and their scope, though, the problem is not something that can be appropriately represented and acted on. (One could try to make a committee for oversight of committees. But that way lies promotion to administration and madness.)

Processes can be *temporally* monadic as well. Any complex process has associated with it a schedule in which events internal to the process are ordered. Suppose that the airport and the subway are run by distinct authorities. The subway has a timetable, which includes arrivals at the airport. The airport also has a timetable for departures and arrivals. Trains coordinate with trains, and planes with planes. Without saying more, however, we have no way to know from the perspective of either system whether trains and planes reliably line up or indeed whether train arrivals and plane departures stand in some fixed and useful relationship to one another—whether this train will allow me to catch that plane. This may be true even though each system runs smoothly. Why? Because so far, for all we have said each process runs in a vacuum and on its own clock.

The mere fact that two processes occur in the same real timeline (as everything must) is not yet to show that, from their own perspectives, they can link up their timetables in a useful way. Synchronization requires something more. Each authority must agree on a timetable and then agree to coordinate to a common clock, ensuring synchronization down the track. This requires breaking the monadicity of the processes in order to guarantee the assumption of a shared timeline between the two authorities.

The same point carries over to computational processes. I suggest that polychrony is the state that obtains when there are multiple computational processes but when the computational ordering for each process does not constrain the ordering between processes.

1. The term is meant to evoke both the notoriously obscure concept in Leibniz and the notoriously obscure concept in Haskell.

https://doi.org/10.1086/710613 Published online by Cambridge University Press

Traditional ways of thinking about computation assume that computational processes care only about the ordering of states, not the actual times at which states occur. Formally speaking, for any computational process, we can define a partial ordering over its states.[2] Two processes are polychronous when it is not possible to give a computational ordering between those two processes given only facts about each process and its ordering.

So (e.g.) if we have two processes A and B, and two states A_n and B_m, then there is a further question about whether the computational orderings on A and B are sufficient on their own to determine whether A_n happens before, simultaneous with, or after B_m. If we cannot figure that out just from the two orderings, then the two processes are polychronous.

Polychrony is certainly a possible relation in which many computational processes can stand. One need only note the lengths that many artifacts go to avoid polychrony—by, for example, representing shared clocks, setting up shared signals to coordinate around resource access, and so on. That kind of synchronization is often pretty easy for macroscale processes like timetables and committee meetings, so it can be easy to forget that it is a requirement at all. When everyone did not have synchronized watches ready at hand, achieving common between-process timescales occupied a great deal of time and energy (Galison 2004).

For the very small, the very fast, and the very large, achieving commensurate timescales can still be problematic. Computer chips spend considerable energy and effort keeping each bit of the die on the same clock. The primary problem solved by GPS receivers is precise clock synchronization (Doberstein 2011, sec. 1.5). Conversely, sometimes the easiest thing to do is to accept polychrony and work around it. Modern internet protocols, for example, have to deal with the fact that telecommunications systems are mostly polychronous; a great deal of reconstruction and synthesizing must take place because different communication streams cannot be assumed to be in synchrony. It is interesting in this regard to note the convergence between telecommunications strategies and arguments that Dennett and others make about the brain—in particular, that representation of temporal ordering becomes crucial when temporal ordering itself cannot do the job (Dennett 1991; De Brigard and Gessell 2016).

3.3. Distinguishing Polychrony. Polychrony strikes me as both a scientifically and a philosophically fruitful concept. It is distinct from other recent concepts in the same ballpark.

2. In most simple models (like Turing machines), this is also a strict ordering. I assume that we can be flexible enough to include overarching computational processes that can be in more than one state at once.

https://doi.org/10.1086/710613 Published online by Cambridge University Press

First, Danks and Plis (2019) have recently argued that different measurement timescales can result in problems for inference to structure and function. This is a serious problem. I think polychrony is an orthogonal issue, although one that makes the problem even more acute, as it raises the possibility of multiple uncoordinated underlying timescales.

Second, while polychrony involves reference to time, it is different from (and more conservative than) dynamic systems approaches to computation (Faries and Chemero 2019). On the one hand, an essential feature of dynamic systems is (I take it) their continuity in time. The polychronous networks I have discussed still involve discrete steps. On the other hand, since dynamic systems are tied to actual time, the idea of polychrony is hard to get off the ground; you need distinct computational monads with distinct timescales to get it running.

Third and finally, although the nodes in a polychronous network can overlap, I suspect that polychrony is distinct from the notions of pluripotency (Price and Friston 2005) or neural reuse (Anderson 2010, 2014; Rathkopf 2013; Barack 2019) or contextualism (Burnston 2016) that have shaped recent philosophy of neuroscience. Reuse, at least in its general form, need not imply anything about time itself. What is important for reuse is that one token process uses a resource in one way and another token process uses it in a different way. This gives rise to a variety of puzzles, but these can (I think) mostly be cashed out in more traditional computational terms. Thus, it seems to me that polychrony is a distinct phenomenon.

4. Process and Computational Explanation. To conclude, I want to suggest a way in which polychronous processes can also be used to drive a wedge between two different perspectives on computation and how that might affect computational explanations of behavior. Broadly speaking, there are two traditions in thinking about computation (Klein 2012). On the one hand, there is a venerable *mathematical* tradition of thinking about computations. Computers are devices for processing information (this line goes), where processing is understood primarily in terms of input-output relationships between data. Computational theory was born in mathematics, and computability is standardly elucidated in terms of sets of computable functions. The standard contenders for the first computer all share a focus on mathematical computation: Babbage's analytic engine for polynomial functions, Colossus for cryptanalysis, ENIAC for artillery firing tables.

On the other hand, there is an equally strong *engineering* perspective on computation. The Jacquard loom and naval fire-control computers were historically important because they gave an efficient, systematic handle on the world. They were useful because they were especially powerful sorts of *control processes* (Bellman 1961). The engineering perspective does not deny that computers often leverage mathematical functions, but math is just one

https://doi.org/10.1086/710613 Published online by Cambridge University Press

ingredient among many in the creation of control processes. The core of control is conditional sequencing of activity. Attach a computer to the world in the right way, and that conditional sequencing can be used to control all sorts of other things. From the engineering perspective, information processing is an adjunct to proper sequencing and control—helpful but not central.

The mathematical tradition abstracts away from details of implementation, including details of timing. That makes sense: mathematical functions are timeless. The mathematical account of computation thus cares about a process only up to the ordering of steps. Two computational processes are equivalent just in case they have the same steps in the same order. If one process unfolds in a minute and one a millennium, they do not differ in any computationally relevant sense. Computational complexity theory similarly only cares about time to the extent that it cares about the growth rate of a process as the problem space gets larger (Aaronson 2015). Clock time is irrelevant.

Conversely, the engineering tradition is quite concerned with timing, because for some tasks timing matters quite a bit. You have to care about clock time if you are building an embedded system for industrial robots or a video game or a flight computer. The dependence on actual time of these systems can be obscured by the fact that extant computers are very, very fast. On a fast enough system, the problem of real-time control can be ignored; assuming that the computation can be performed quickly enough, clock time can be treated as a mere parameter by the underlying implementation.

Yet that speed is a convenient feature of modern hardware. If we want a computational understanding of brains, it is clear that clock time matters quite a bit. As Jékely, Keijzer, and Godfrey-Smith (2015) have recently argued, a similar pair of perspectives appears when thinking about the early evolution of the nervous system. They contrast *input-output* and *internal coordination* views of the early nervous system. The input-output view emphasizes the role of neurons in transforming information, to allow for increasingly complex responses. The internal coordination view, by contrast, emphasizes the role of neurons in the efficient sequencing of internal events like muscle contractions. Much interesting behavior requires this sequencing: perceiving threats is no use if your only response is flopping around ineffectually. More generally, brains are under tremendous evolutionary and metabolic pressure to do more with less (Sterling and Laughlin 2015). They must respond to the environment in real time. Neurons are not particularly fast, though. Whatever brains do can be represented by mathematical functions. But understanding brains requires understanding the temporal constraints under which they act and the strategies of control they have developed to work within those constraints.

I think polychrony helps show why the engineering perspective, and the focus on actual time, is ineliminable in certain kinds of computational explanations. Polychrony occurs when distinct but interacting processes cannot

https://doi.org/10.1086/710613 Published online by Cambridge University Press

be assumed to have the same timescales. When dealing with polychronous processes, the actual extension in time of the processes is crucially important to understanding computational phenomena that involve them. So a full theory of interacting polychronous computational processes cannot be a fully computational theory of the same sort that we use to understand the individual processes themselves.

As an example, consider two computational processes that interact via their use of a limited resource like memory or processor time. It is clear that explanations of the overall system that contains both processes will need to take into account the possibility of resource starvation (Klein 2018). To determine whether two processes compete requires knowing when they try to access a resource, where this is determined by a timescale common to both. Yet if the processes are polychronous, the possibility of resource competition cannot be adequately represented by either process—because internally they do not have a commensurate timescale.

This is a simple point, but I think it is easy to overlook its importance. It is tempting to assume that whenever we have two interacting computational processes, there must be a third unified process of which the different subprocesses are components. Yet once stated, the unattractiveness of this move becomes clear. For one, polychronous processes may run on completely distinct computational architectures; positing a joint architecture merely obliterates useful distinctions. For another, thinking about distinct interacting processes often gives a simpler, more fruitful theory than a single monolithic model. Forcing everything into a single overarching process was one of the issues with the machine-state functionalism that Putnam (1967/1991) proposed and later criticized. Rather than thinking that we are a succession of complex and often sui generis total states, it is much easier to say that we can do multiple things at once. Finally, recognizing the unique demands of interacting polychronous processes can reveal unexpected and novel strategies for coordination. Calcott et al.'s (2008) work on publish-subscribe models of gene regulatory networks, for example, strikes me as another field in which polychrony might be a fruitful concept. Indeed, because polychronous networks must solve coordination problems without substantial coordination, polychronous networks can often be extended by adding new nodes—more neurons, new genes—without disrupting existing connections.

5. Conclusion. Polychronous networks demonstrate a way in which computational explanations need to be sensitive to actual timing of processes—that is, to more than what the traditional mathematical perspective on computation would take into account. This does not show that the mathematical perspective is wrong, of course. What it shows is that computational explanation in complex systems will require more than the simple, ordinal conception of timing on which the mathematical account rests.

https://doi.org/10.1086/710613 Published online by Cambridge University Press

REFERENCES

Aaronson, S. 2015. "Why Philosophers Should Care about Computational Complexity." In *Computability: Gödel, Turing, Church, and Beyond*, ed. B. J. Copeland, C. Posy, and O. Shagrir, 261–327. Cambridge, MA: MIT Press.

Anderson, M. 2010. "Neural Reuse: A Fundamental Organizational Principle of the Brain." *Behavioral and Brain Sciences* 33 (4): 245–66.

———. 2014. *After Phrenology: Neural Reuse and the Interactive Brain*. Cambridge, MA: MIT Press.

Barack, D. L. 2019. "Cognitive Recycling." *British Journal for the Philosophy of Science* 70 (1): 239–68.

Bellman, R. E. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton, NJ: Princeton University Press.

Burnston, D. C. 2016. "A Contextualist Approach to Functional Localization in the Brain." *Biology and Philosophy* 31 (4): 527–50.

Calcott, B., D. Balcan, P. A. Hohenlohe, and A. Califano. 2008. "A Publish-Subscribe Model of Genetic Networks." *PloS One* 3 (9): e3245.

Danks, D., and S. Plis. 2019. "Amalgamating Evidence of Dynamics." *Synthese* 196 (8): 3213–30.

De Brigard, F., and B. Gessell. 2016. "Time Is Not of the Essence: Understanding the Neural Correlates of Mental Time Travel." In *Seeing the Future: Theoretical Perspectives on Future-Oriented Mental Time Travel*, ed. K. Michaelian and S. B. Klein, 153–80. Oxford: Oxford University Press.

Dennett, D. 1991. *Consciousness Explained*. Boston: Little, Brown.

Doberstein, D. 2011. *Fundamentals of GPS Receivers: A Hardware Approach*. New York: Springer.

Faries, F., and A. Chemero. 2019. "Dynamic Information Processing." In *The Routledge Handbook of the Computational Mind*, ed. M. Sprevak and M. Colombo, 134–48. London: Routledge.

Galison, P. 2004. *Einstein's Clocks, Poincaré's Maps: Empires of Time*. New York: Norton.

Izhikevich, E. M. 2006. "Polychronization: Computation with Spikes." *Neural Computation* 18 (2): 245–82.

Izhikevich, E. M., J. A. Gally, and G. M. Edelman. 2004. "Spike-Timing Dynamics of Neuronal Groups." *Cerebral Cortex* 14 (8): 933–44.

Jékely, G., F. Keijzer, and P. Godfrey-Smith. 2015. "An Option Space for Early Neural Evolution." *Philosophical Transactions of the Royal Society* B 370 (1684): 20150181.

Klein, C. 2012. "Two Paradigms for Individuating Implementations." *Journal of Cognitive Science* 13 (2): 167–79.

———. 2018. "Mechanisms, Resources, and Background Conditions." *Biology and Philosophy* 33 (36): 1–14.

Price, C. J., and K. J. Friston. 2005. "Functional Ontologies for Cognition: The Systematic Definition of Structure and Function." *Cognitive Neuropsychology* 22 (3): 262–75.

Putnam, H. 1967/1991. "The Nature of Mental States." In *The Nature of Mind*, ed. D. M. Rosenthal, 197–210. New York: Oxford University Press.

Rathkopf, C. A. 2013. "Localization and Intrinsic Function." *Philosophy of Science* 80 (1): 1–21.

Smith, B. C. 1996. *On the Origin of Objects*. Cambridge, MA: MIT Press.

Sterling, P., and S. Laughlin. 2015. *Principles of Neural Design*. Cambridge, MA: MIT Press.

https://doi.org/10.1086/710613 Published online by Cambridge University Press