Constructible sheaves and their cohomology for asynchronous logic and computation

14 January 2010

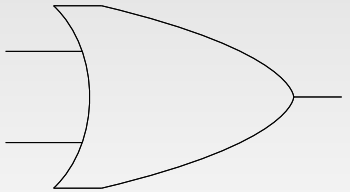
Michael Robinson

Acknowledgements

- This is a preliminary report on progress in a larger project on applied sheaf theory
  - More substantial results are to come!
- It's joint work with
  - Robert Ghrist (Penn)
  - Yasu Hiraoka (Hiroshima)
- The focus is on logic here, but is part of
  - AFOSR MURI on Information Dynamics in Networks
  - PI: Rob Calderbank (Princeton)

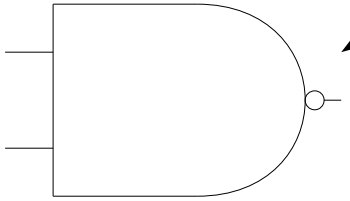
Logic gates

AND

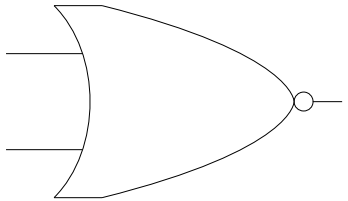


OR

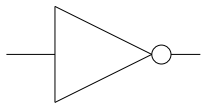
“bubble” indicates negation



NAND

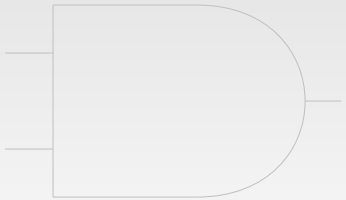


NOR

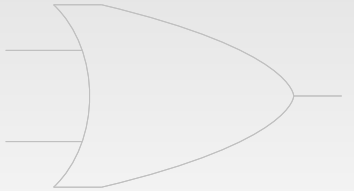
1  0

NOT

Logic gates

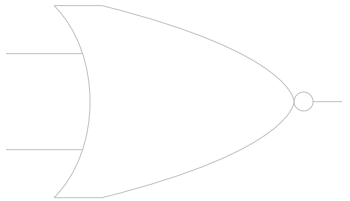


AND



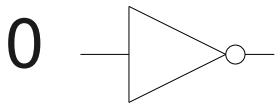
OR

NAND



NOR

A change occurs...



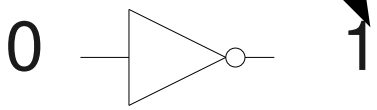
0 0

NOT

Logic gates

Propagation delay varies from device to device

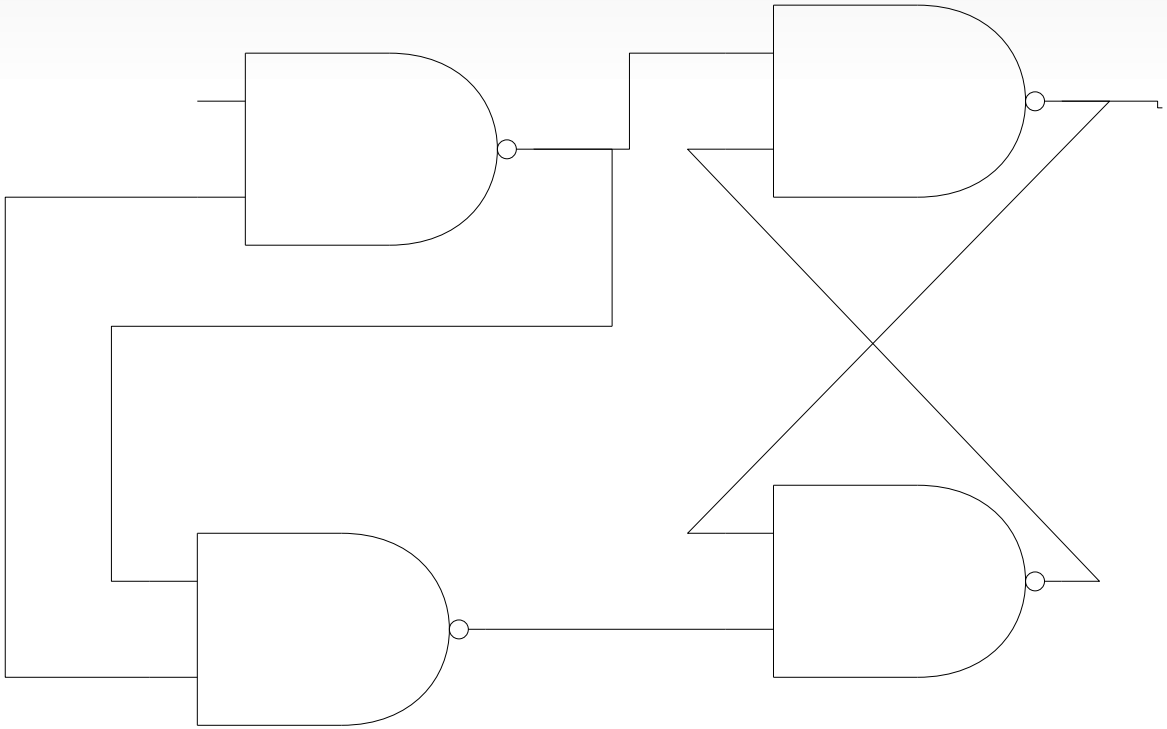
... eventually changes the output



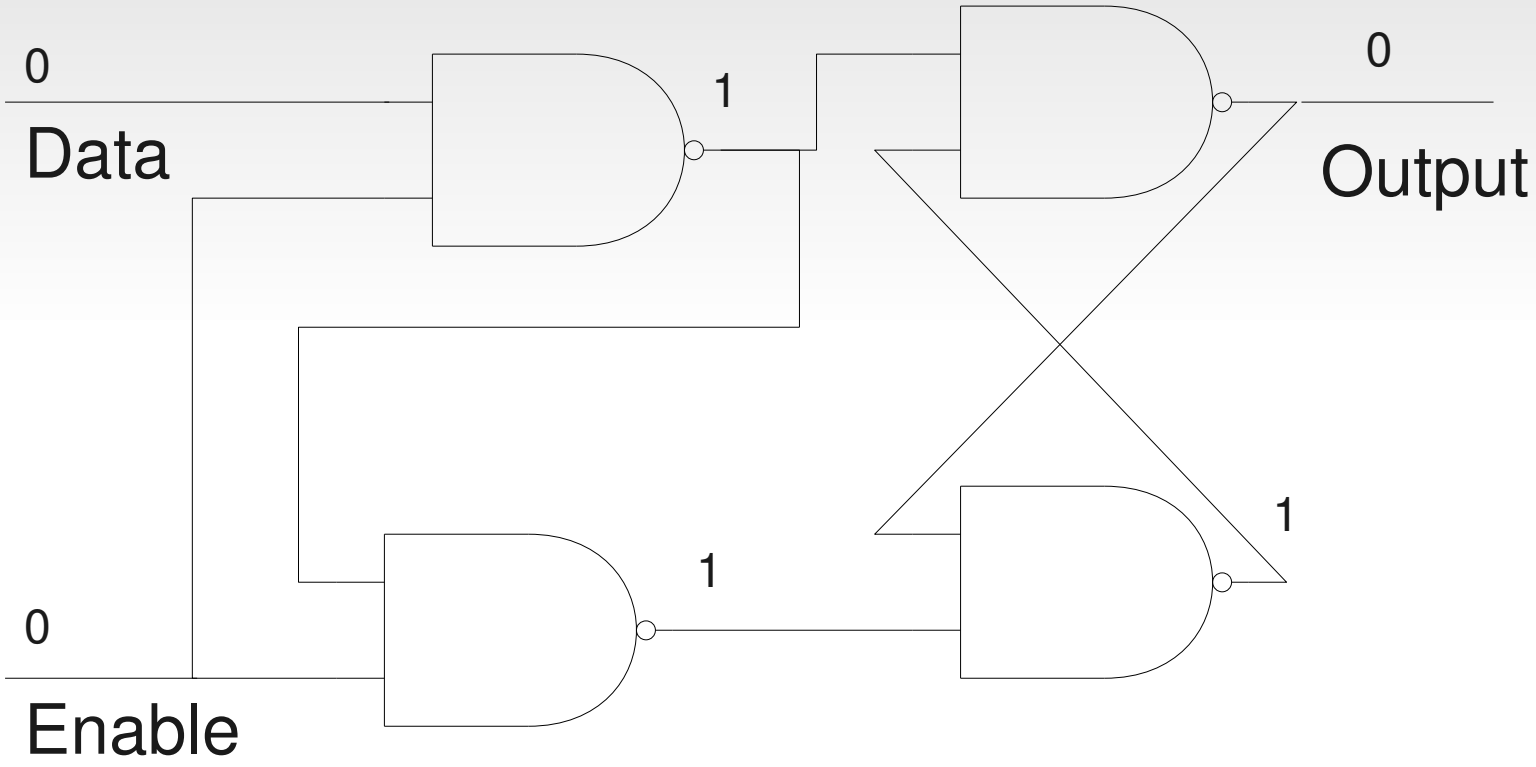
NOT

Problem: time-bound logic

- Propagation delays along connections and within gates!
- Feedback – can hold state
- Race conditions:
  - Hazards
  - Glitches
  - Oscillations
  - Lock-ups

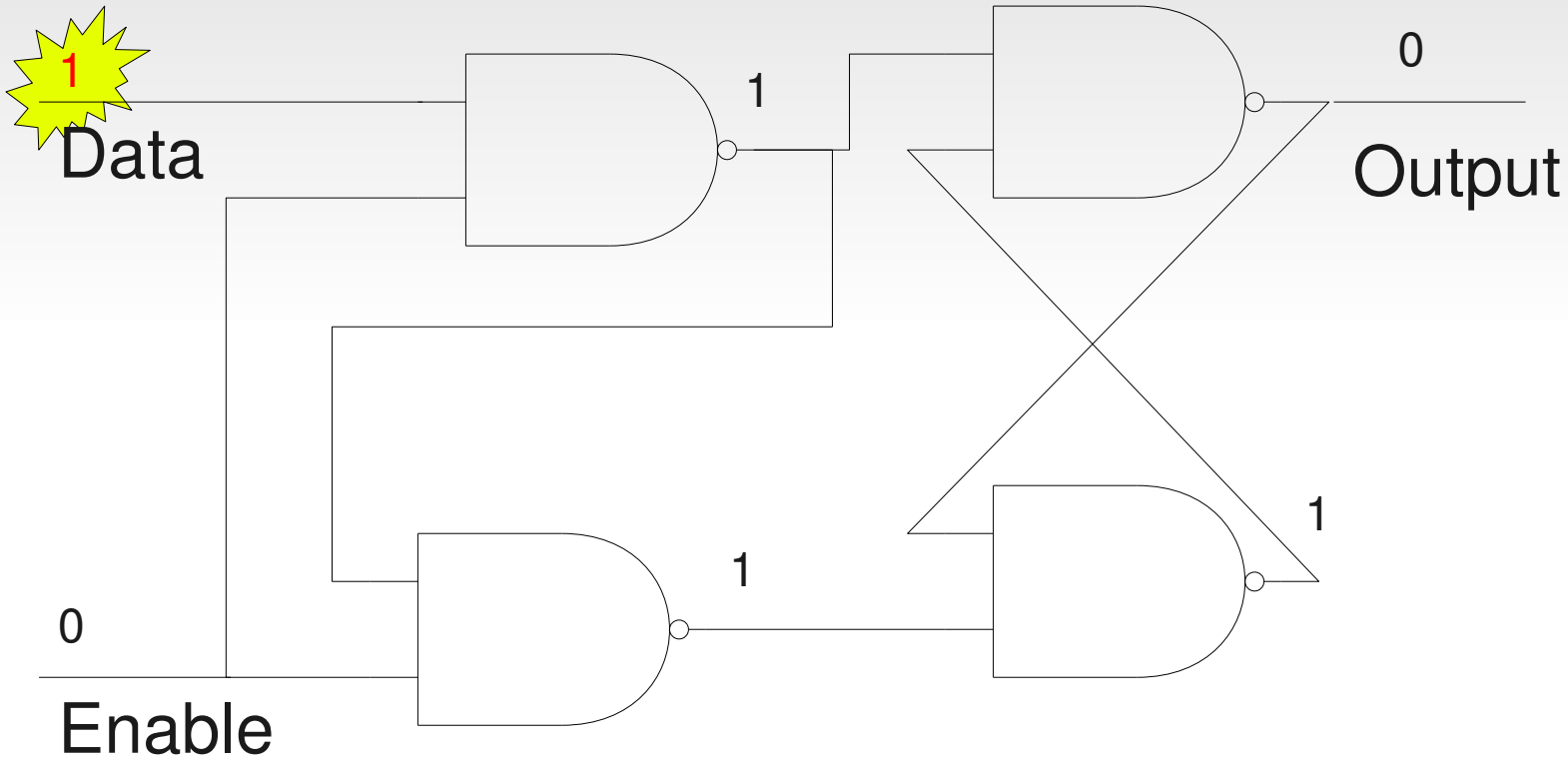


Example of timebound logic



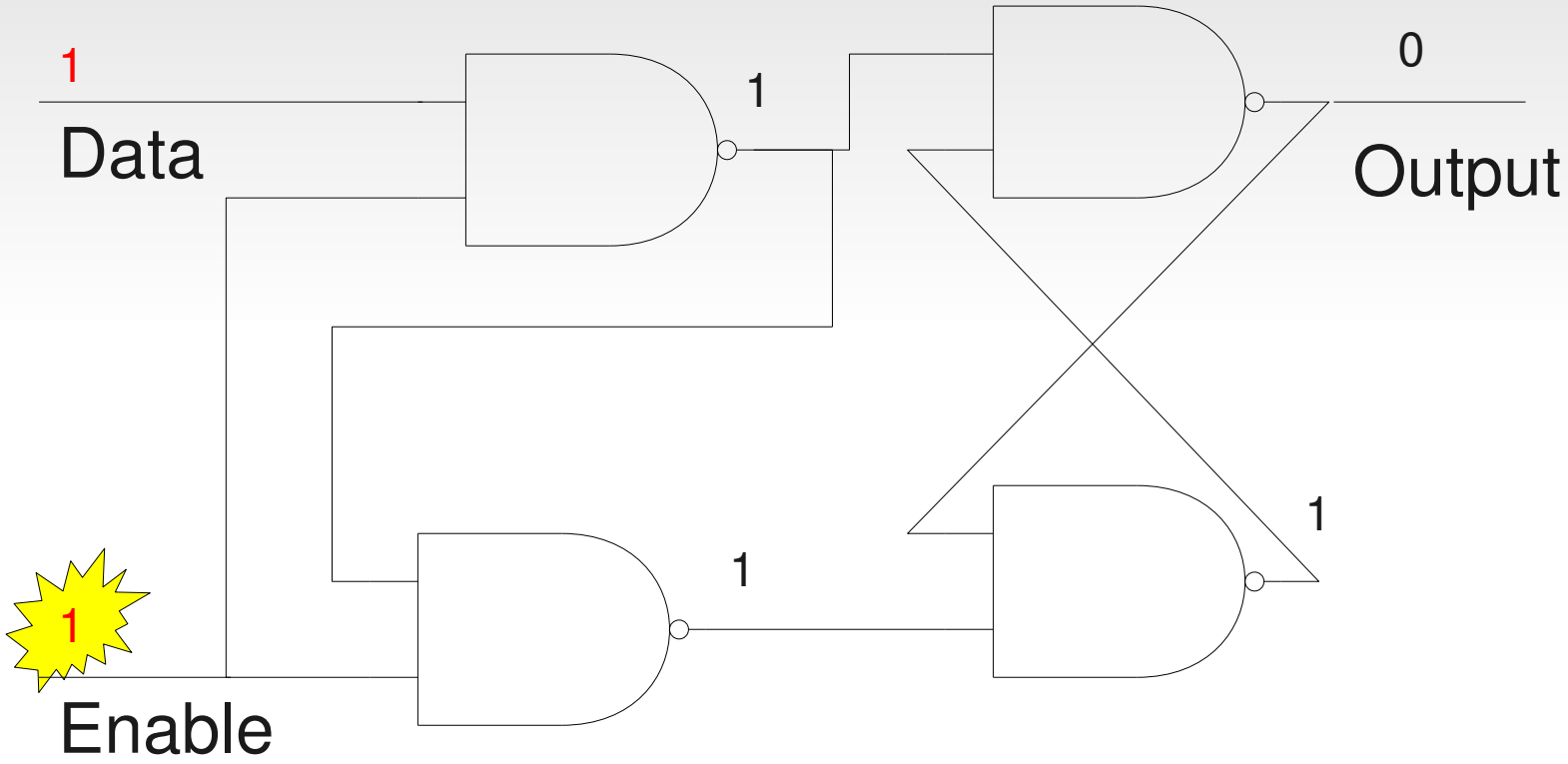
This is an E flip-flop circuit, a basic memory element. It's initially storing the value 0

Example of timebound logic



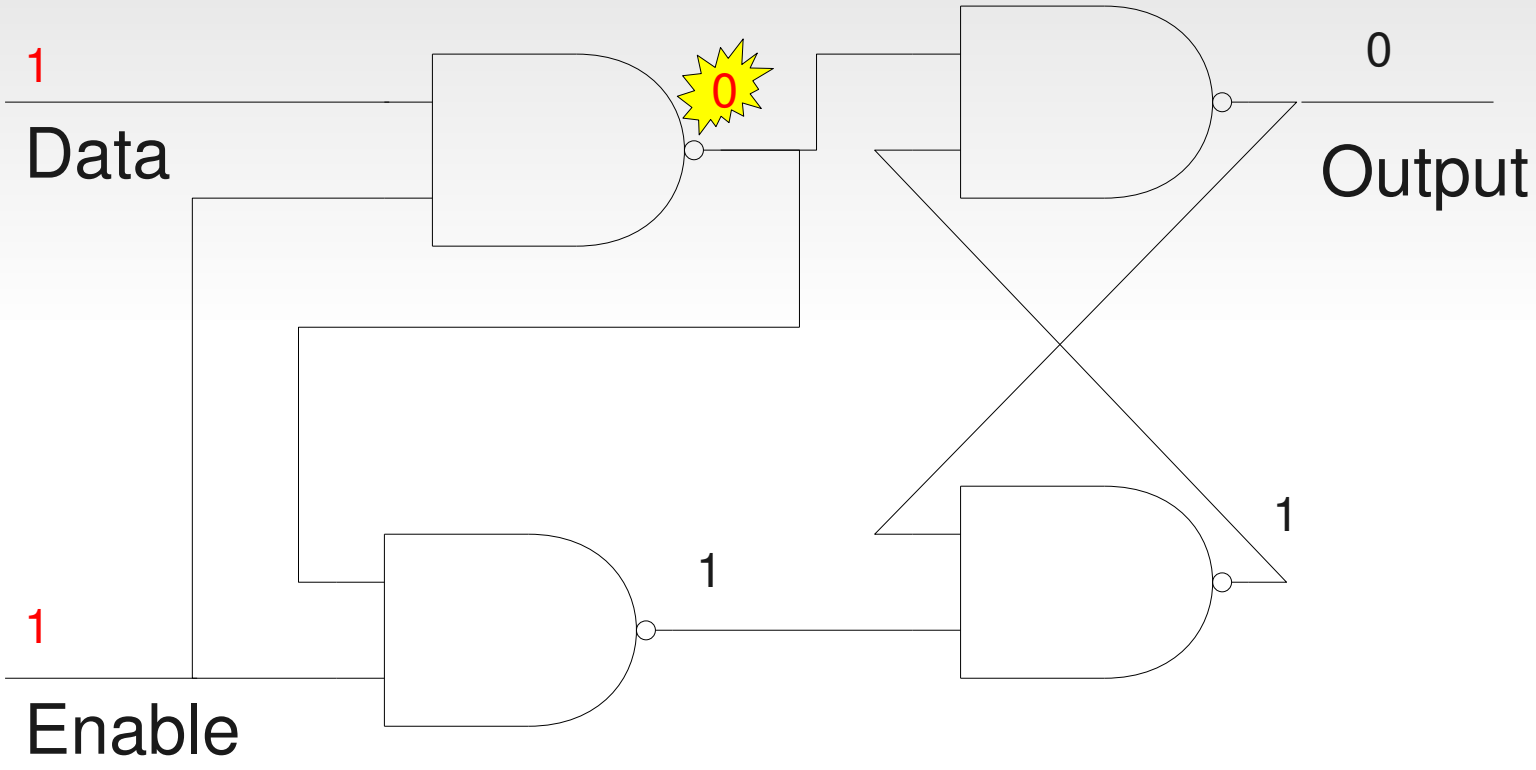
If we change the Data input to 1, nothing exciting happens...

Example of timebound logic



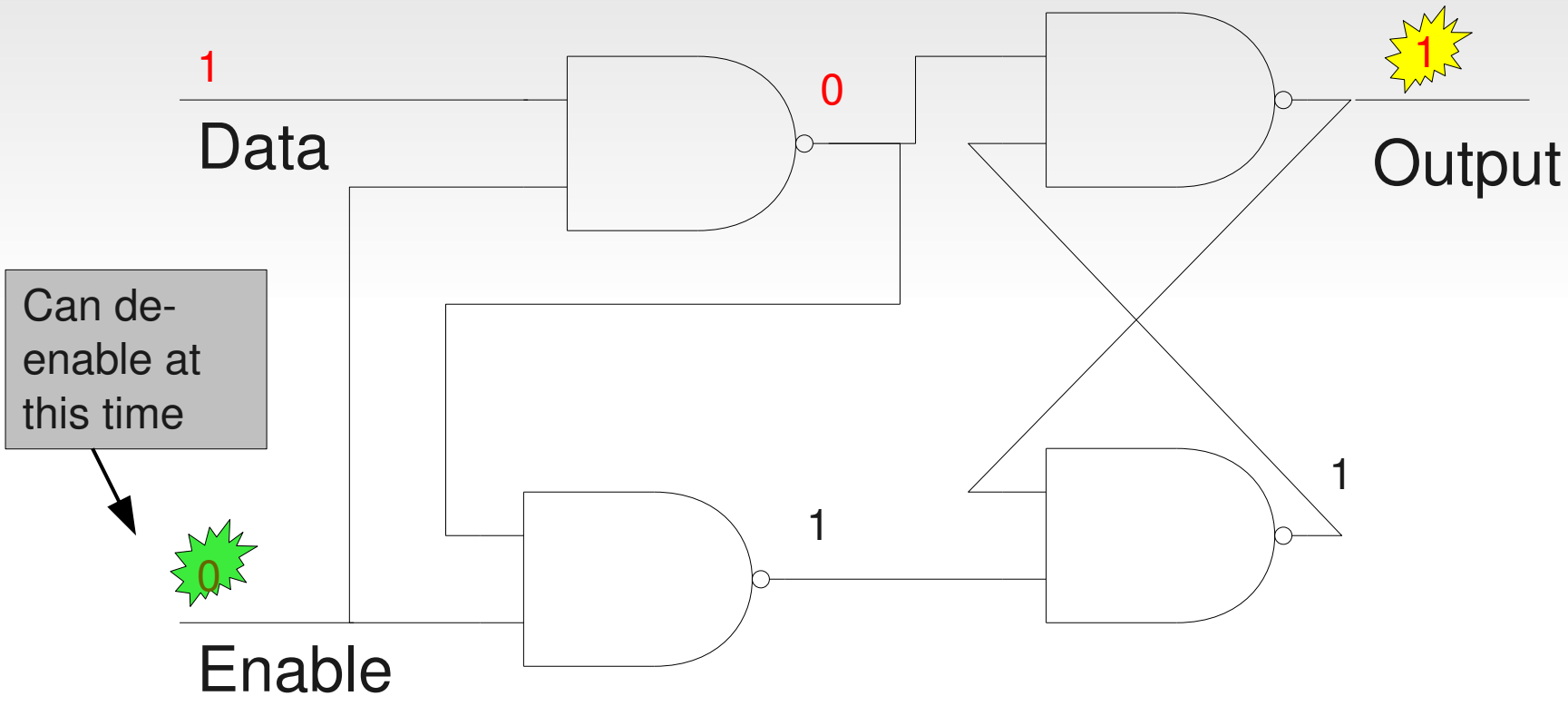
Pulsing the Enable input to 1 causes the Data input to be “read” and “stored”...

Example of timebound logic



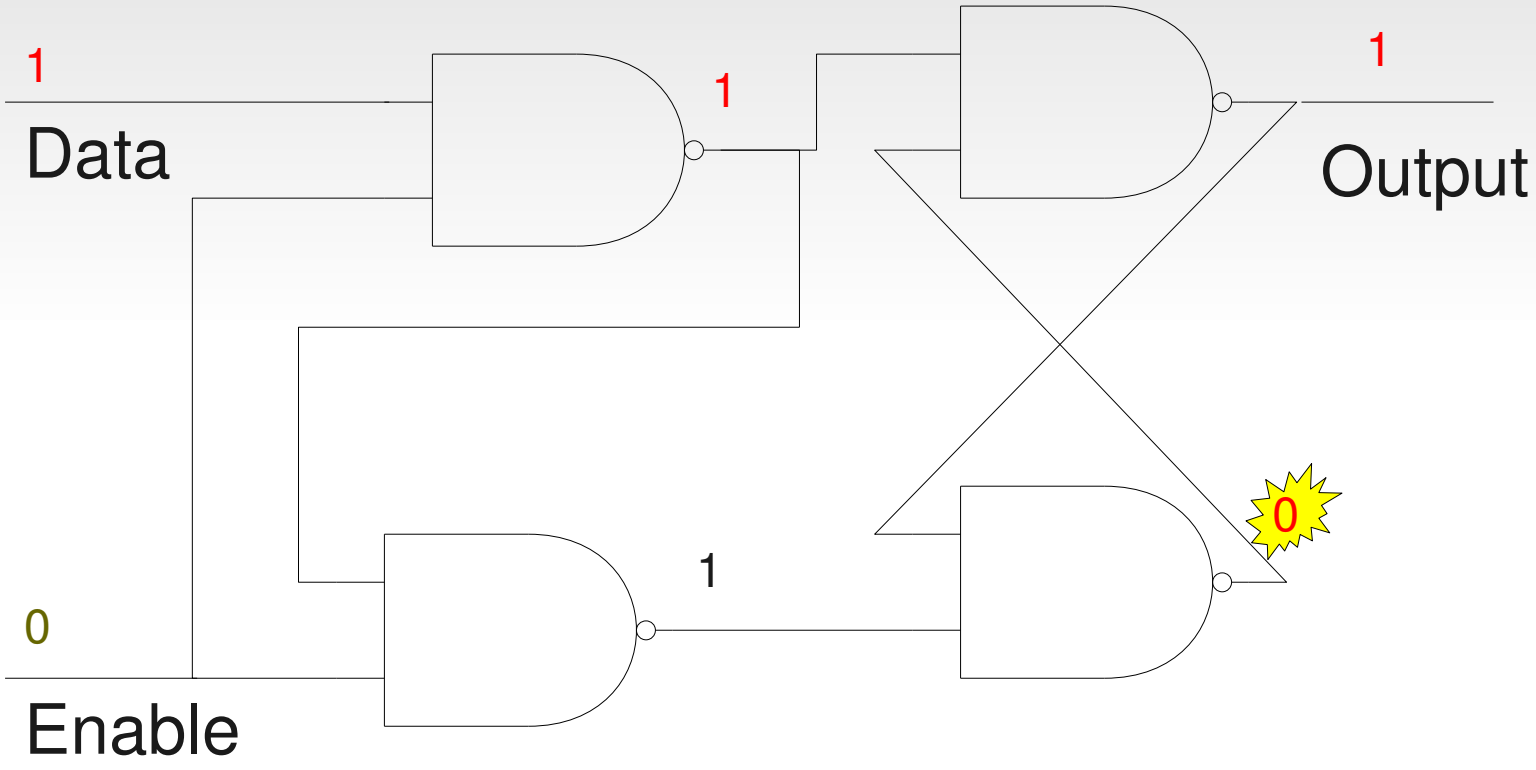
... but it takes time... t=1

Example of timebound logic



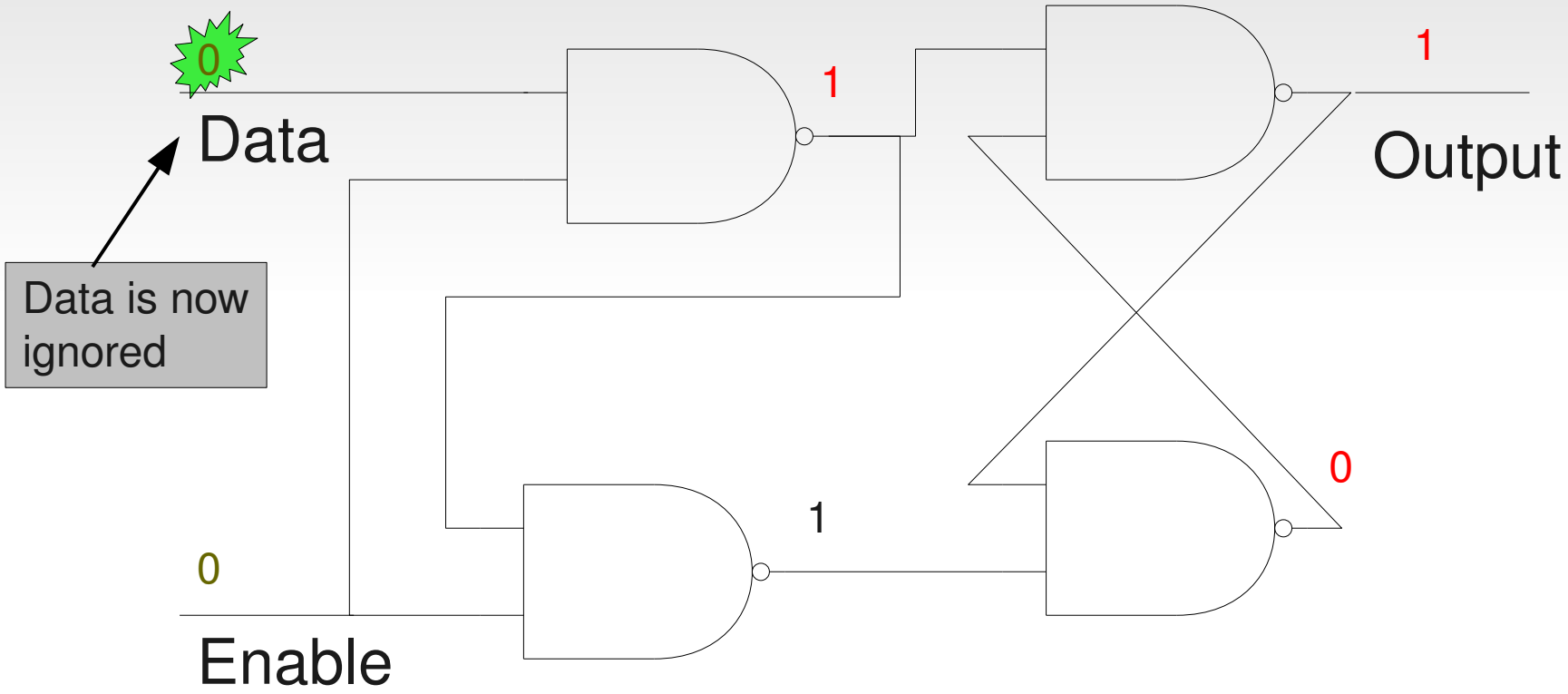
... but it takes time... t=2

Example of timebound logic



... but it takes time... t=3

Example of timebound logic

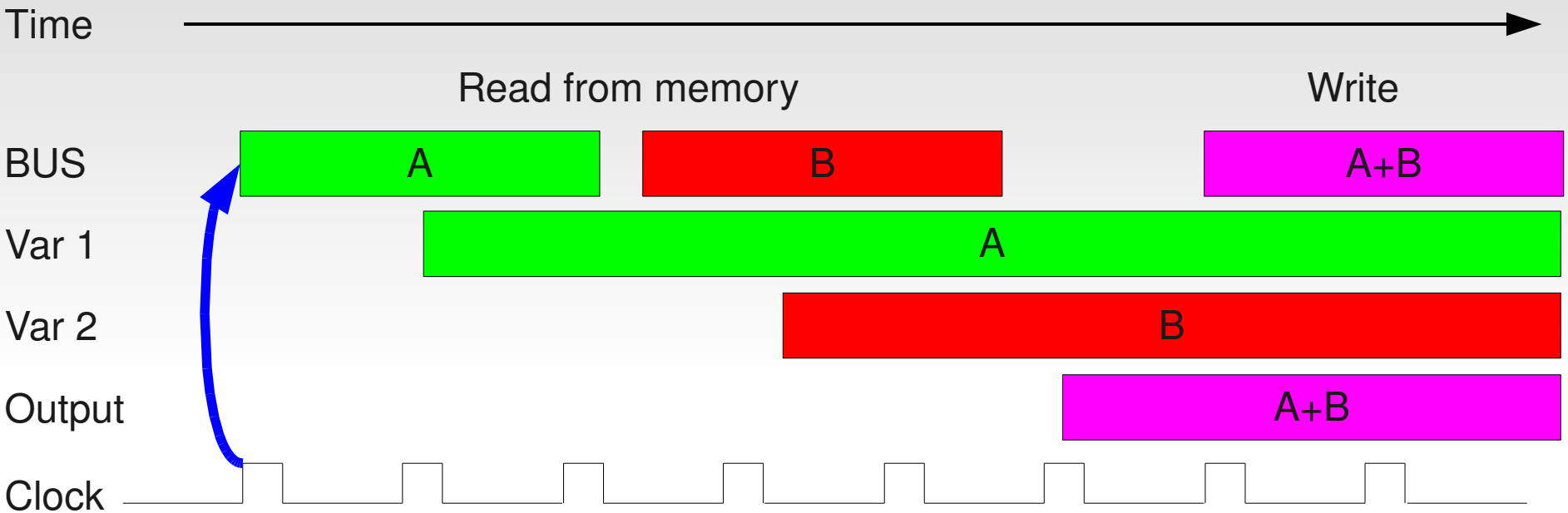


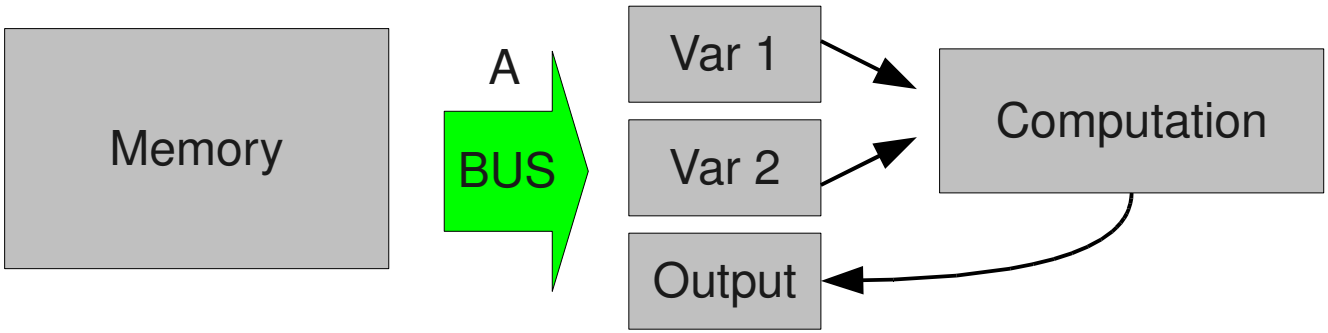
... and will hold the new value!

Synchronous design

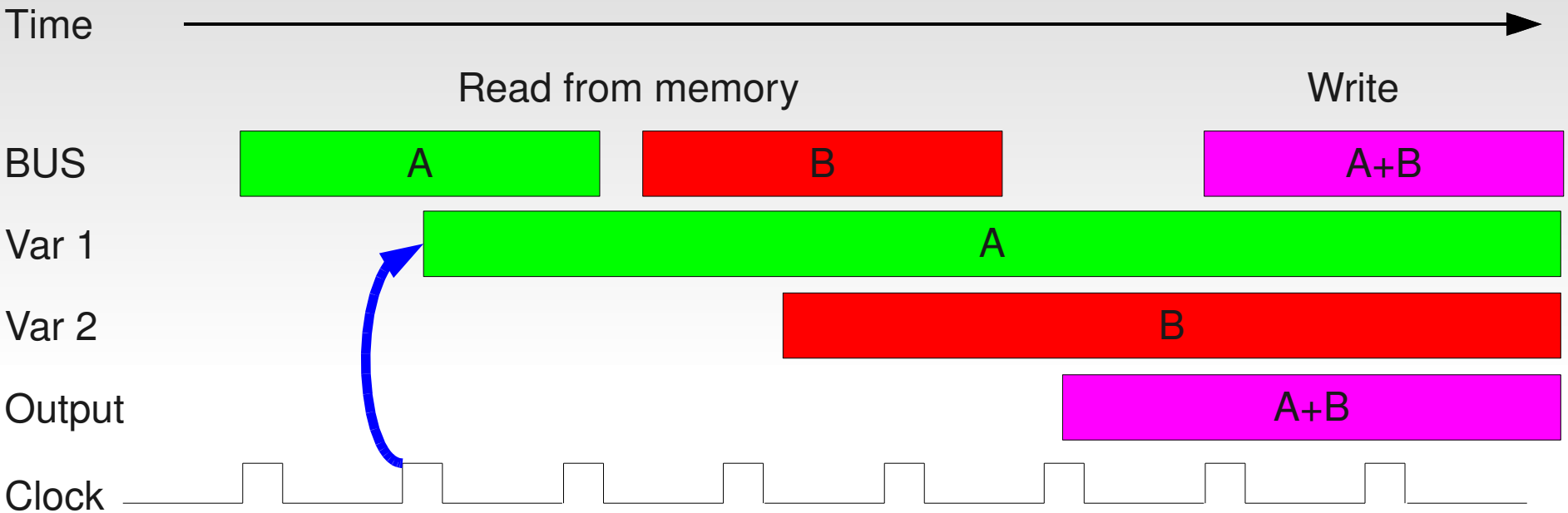
- Can avoid race conditions by polling after transients are finished
  - Unavoidable limitation: limited by the slowest circuit
- Synchronous solution: circuits poll their inputs only at specific points in time – **a global clock**
- But...
  - Biggest single drain of power in modern CPUs is the clock
  - Clock distribution and skew a major problem
  - Correcting clock skew requires additional circuitry and power usage

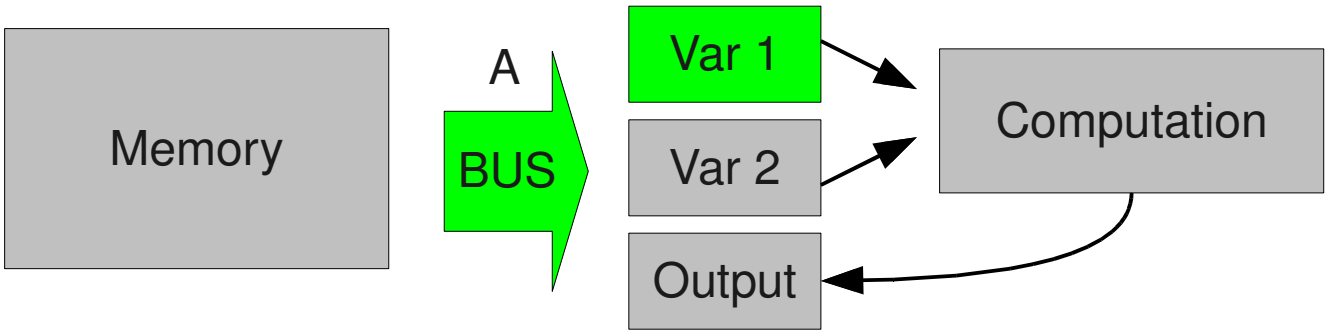
Example logic timeline (synchronous)



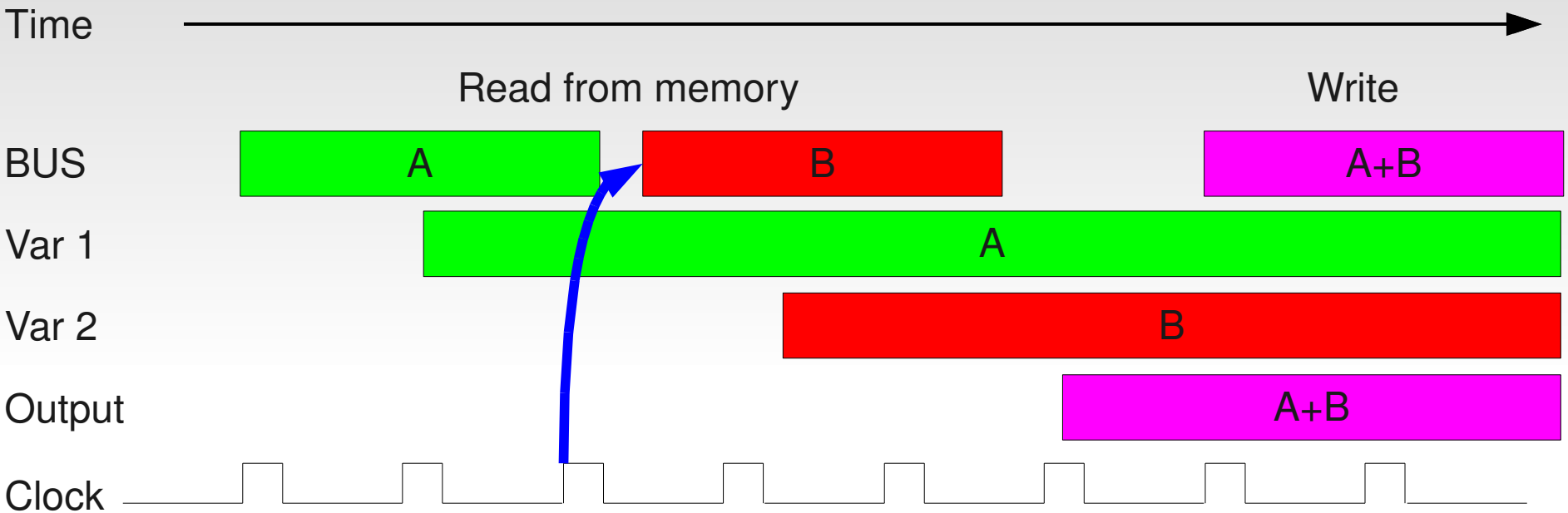


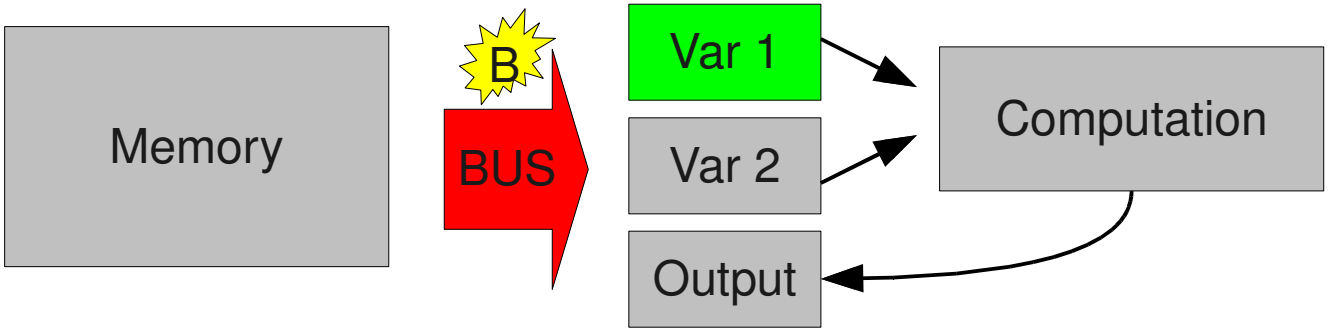
Example logic timeline (synchronous)



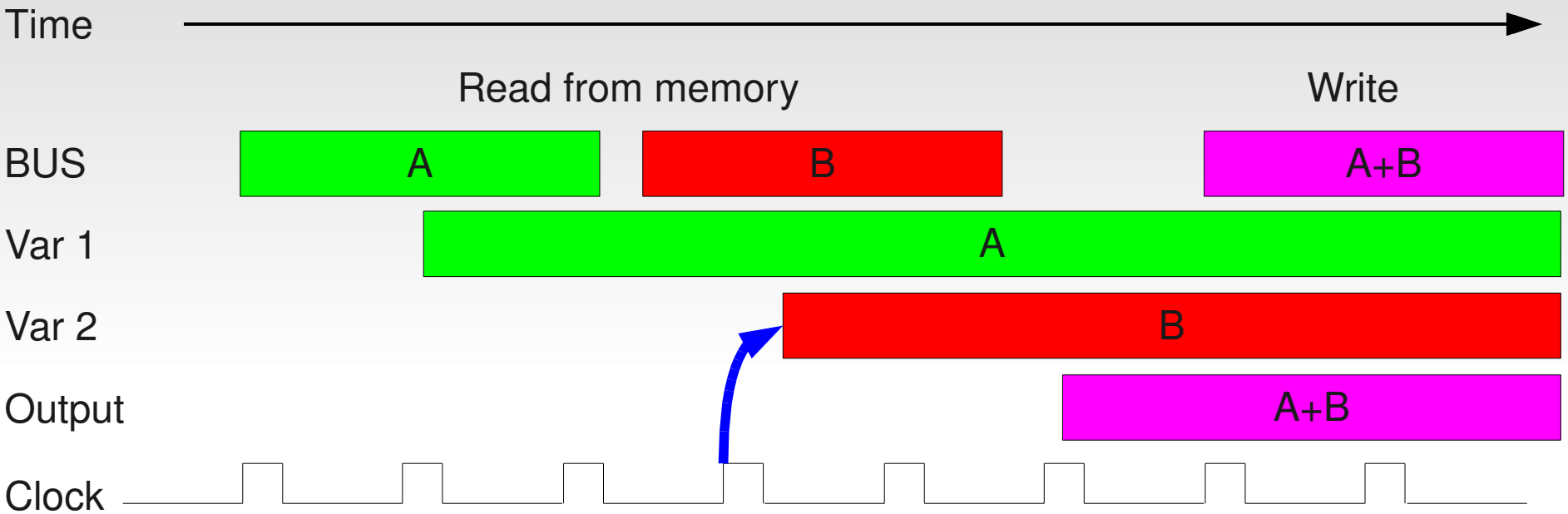


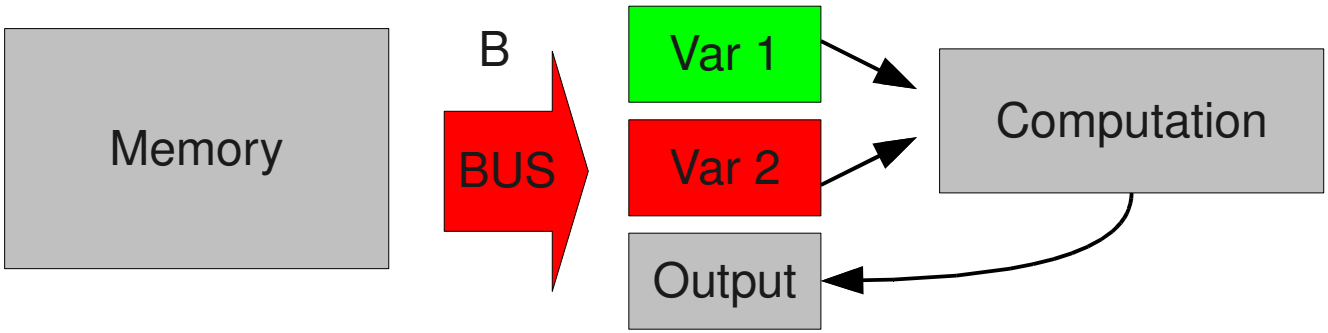
Example logic timeline (synchronous)



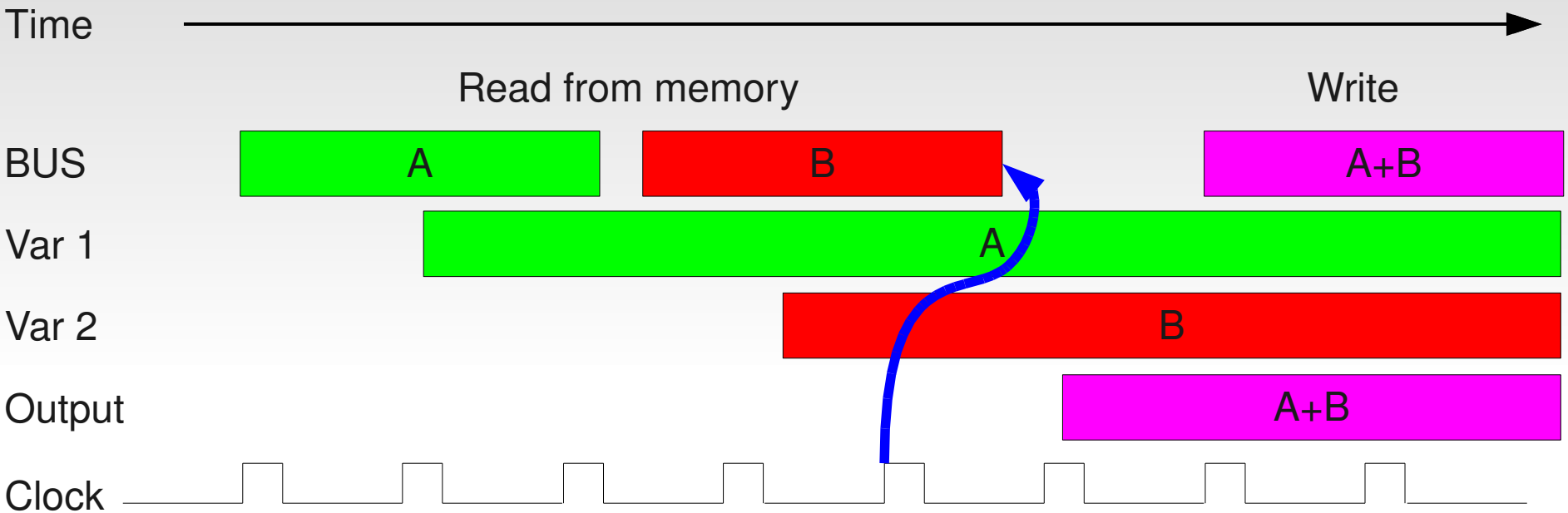


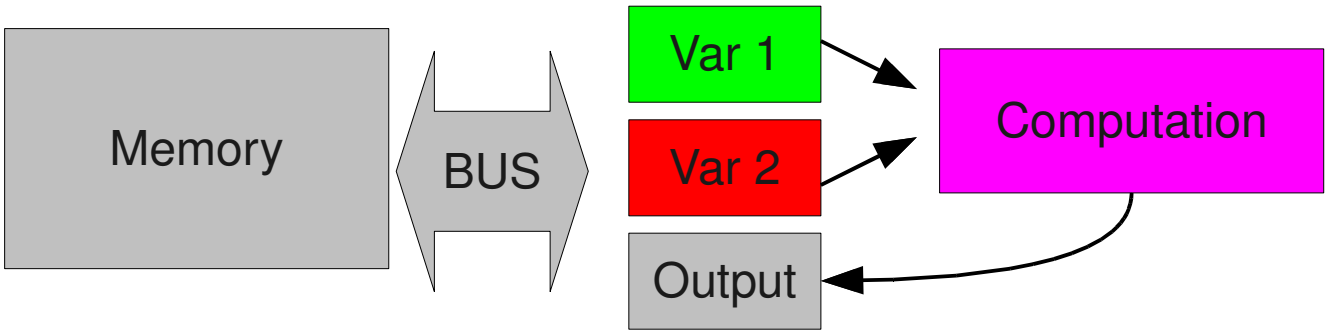
Example logic timeline (synchronous)



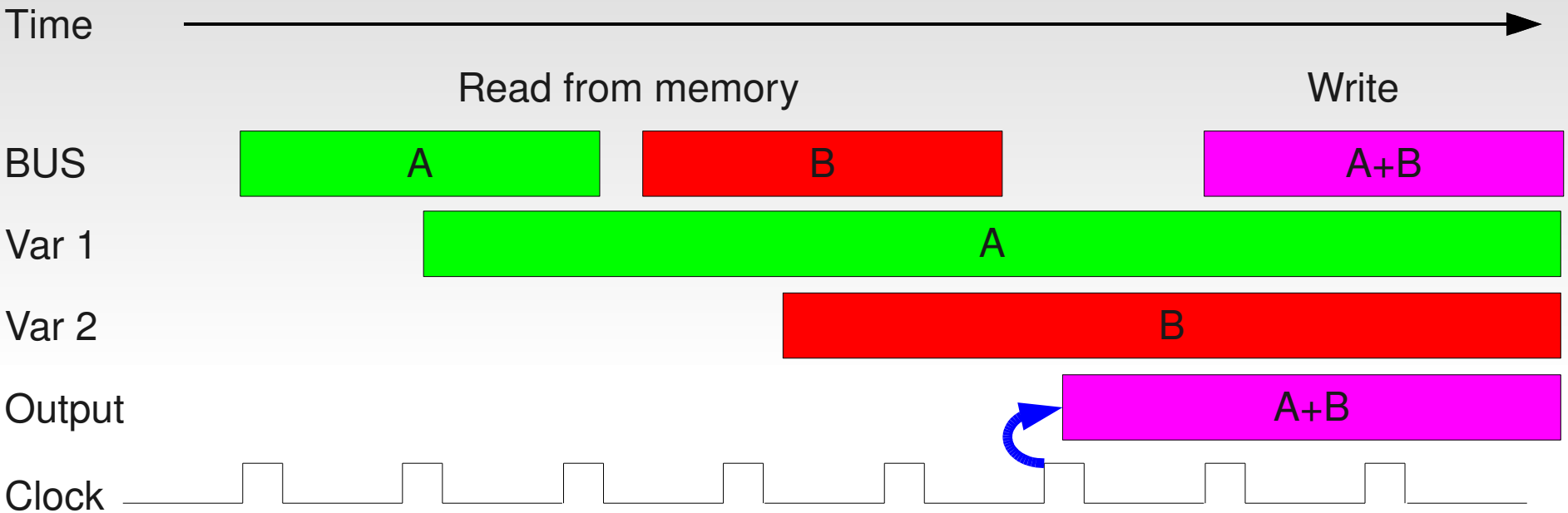


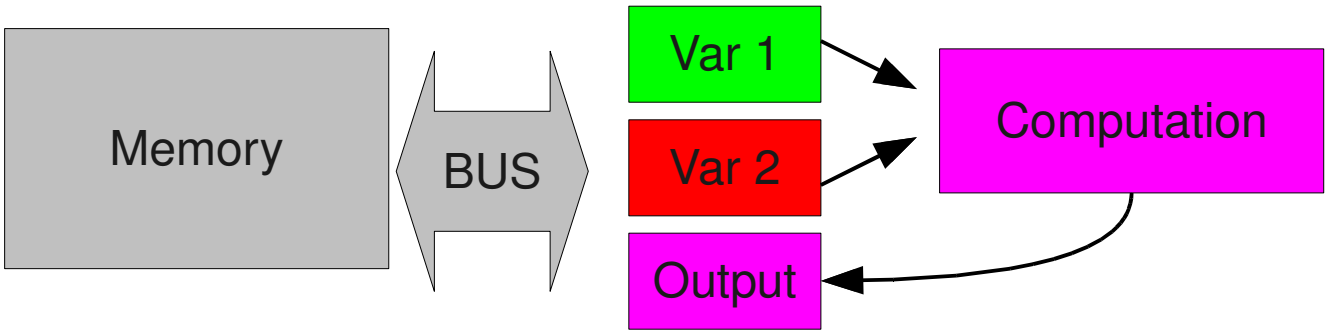
Example logic timeline (synchronous)



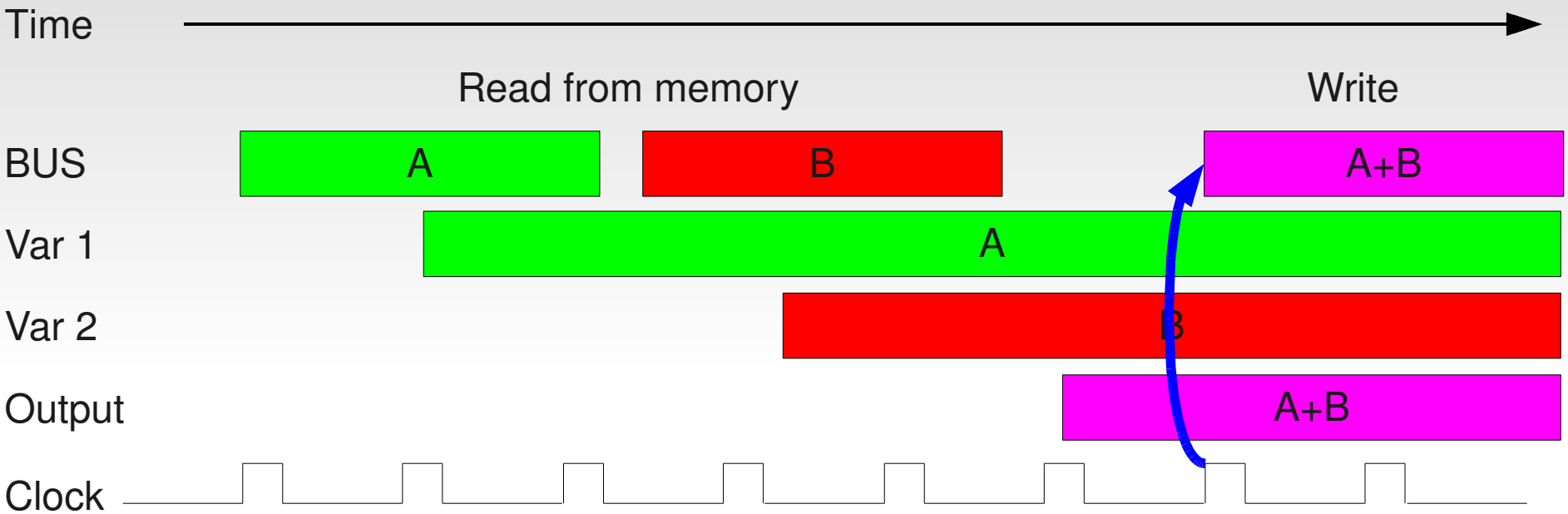


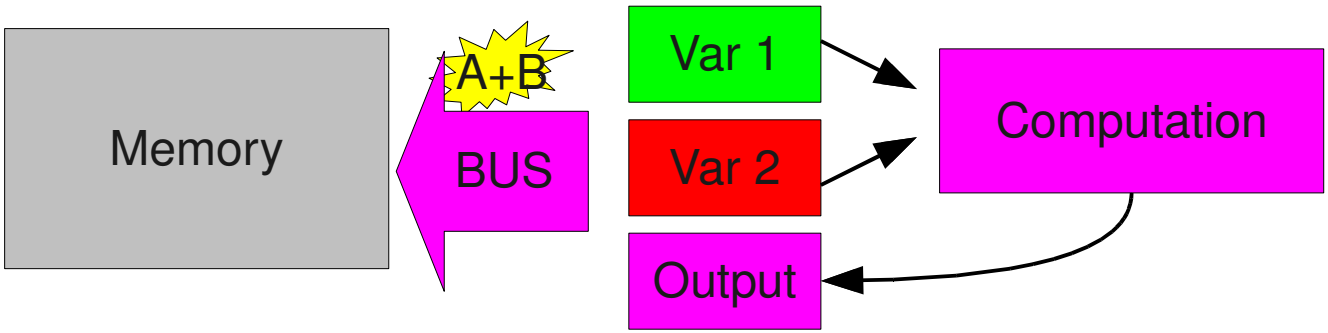
Example logic timeline (synchronous)



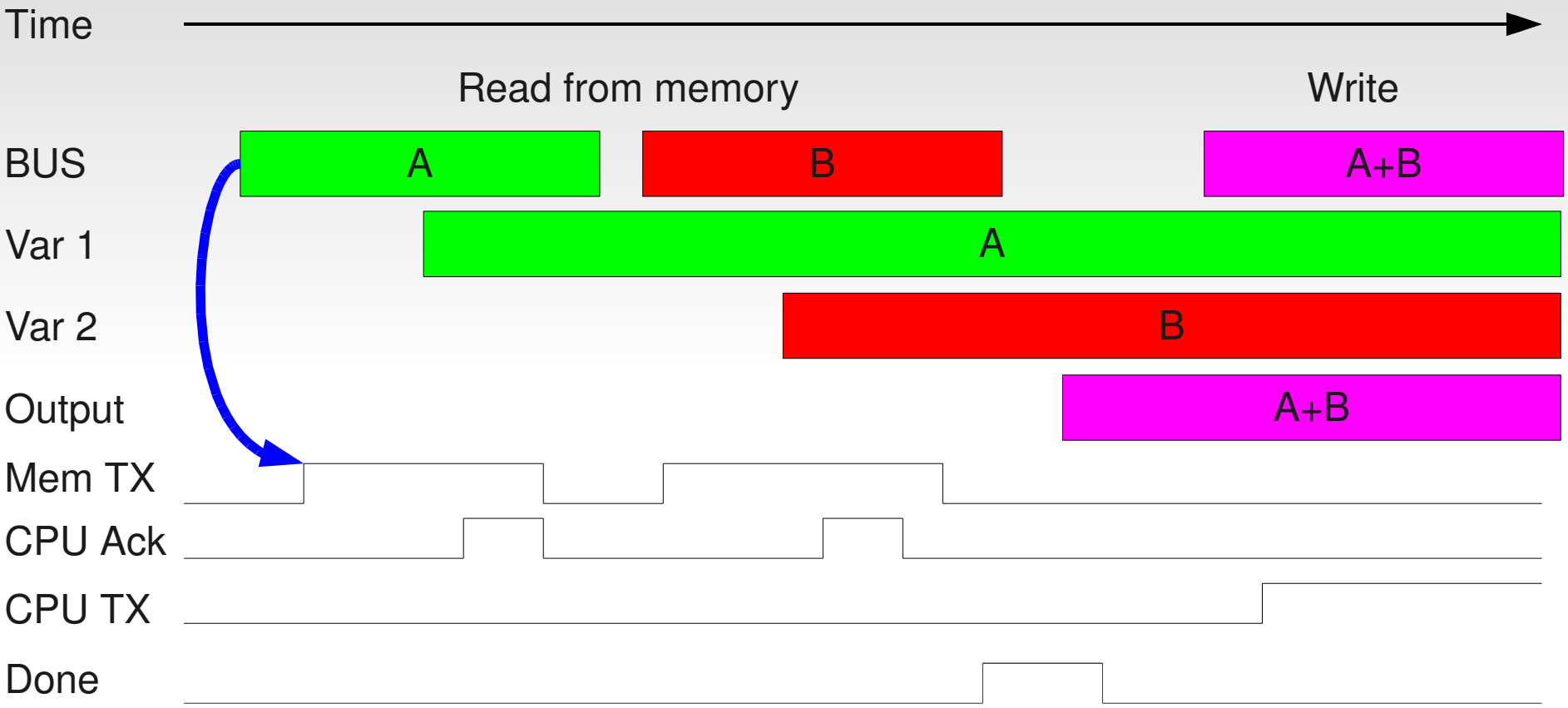


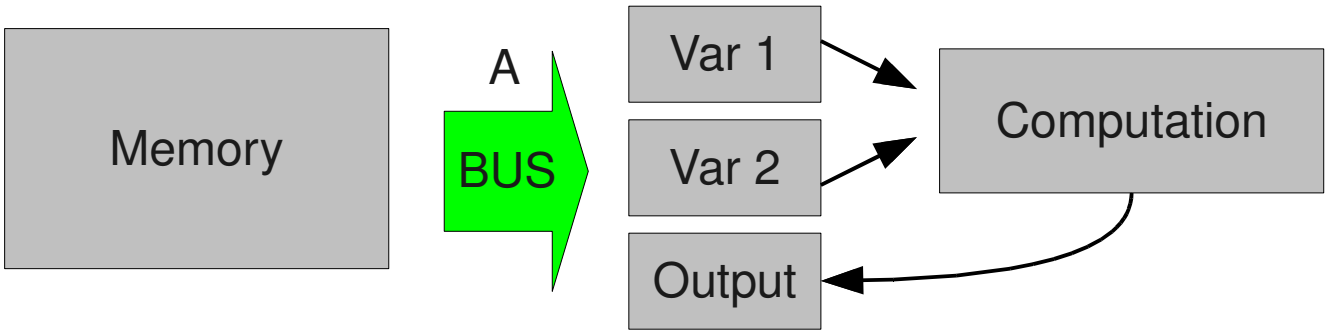
Example logic timeline (synchronous)



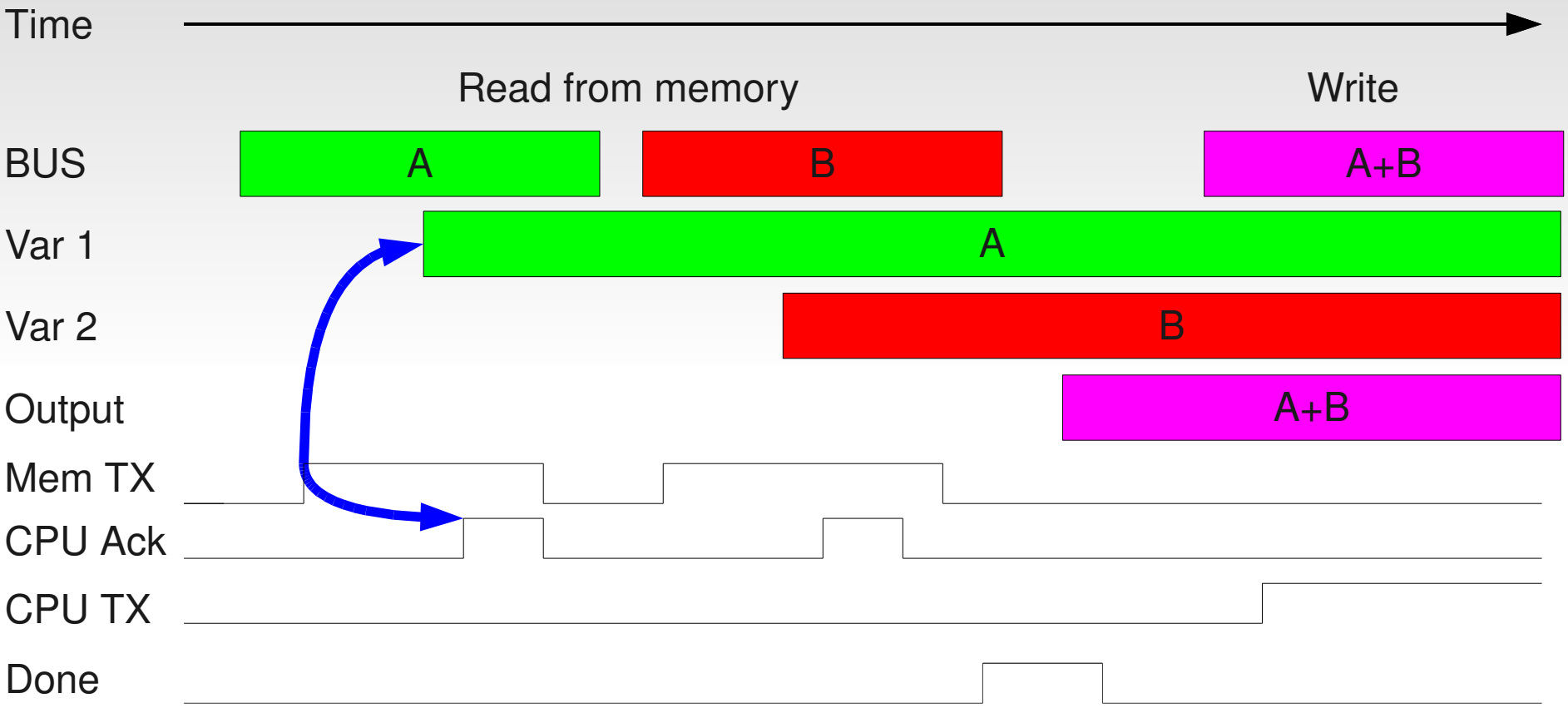


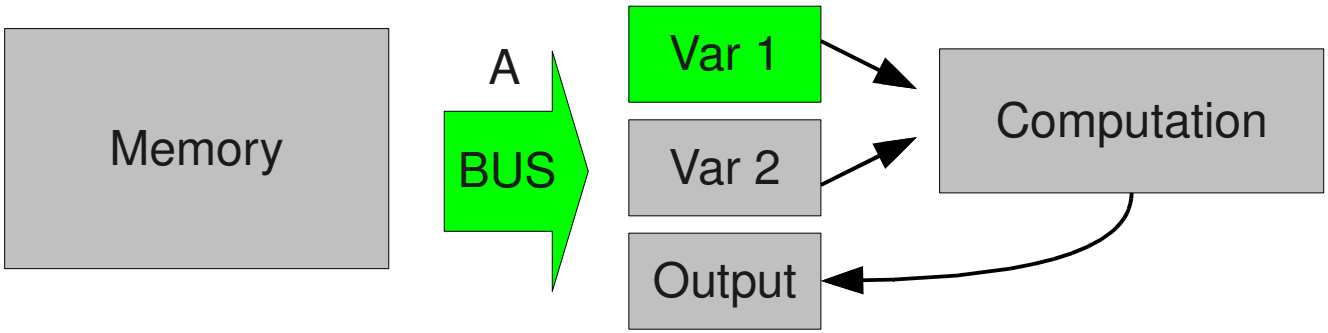
Example logic timeline (asynchronous)



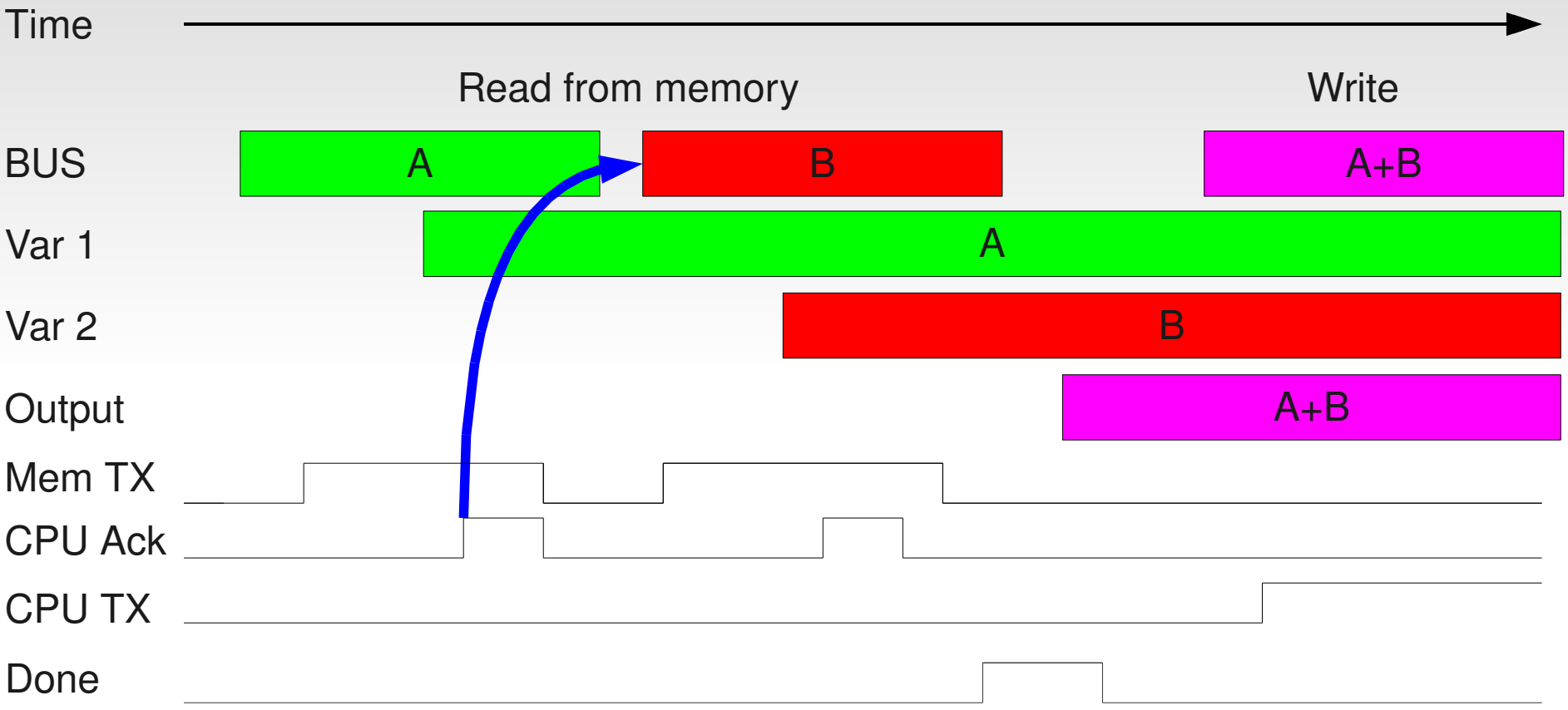


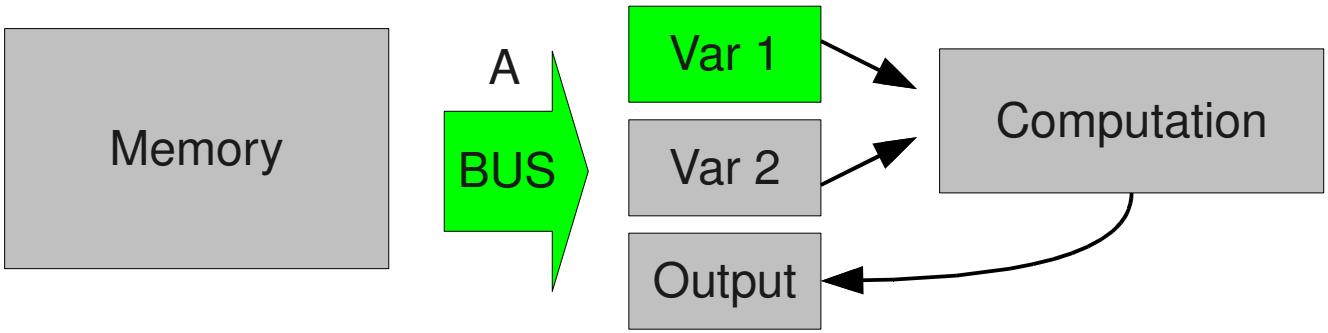
Example logic timeline (asynchronous)



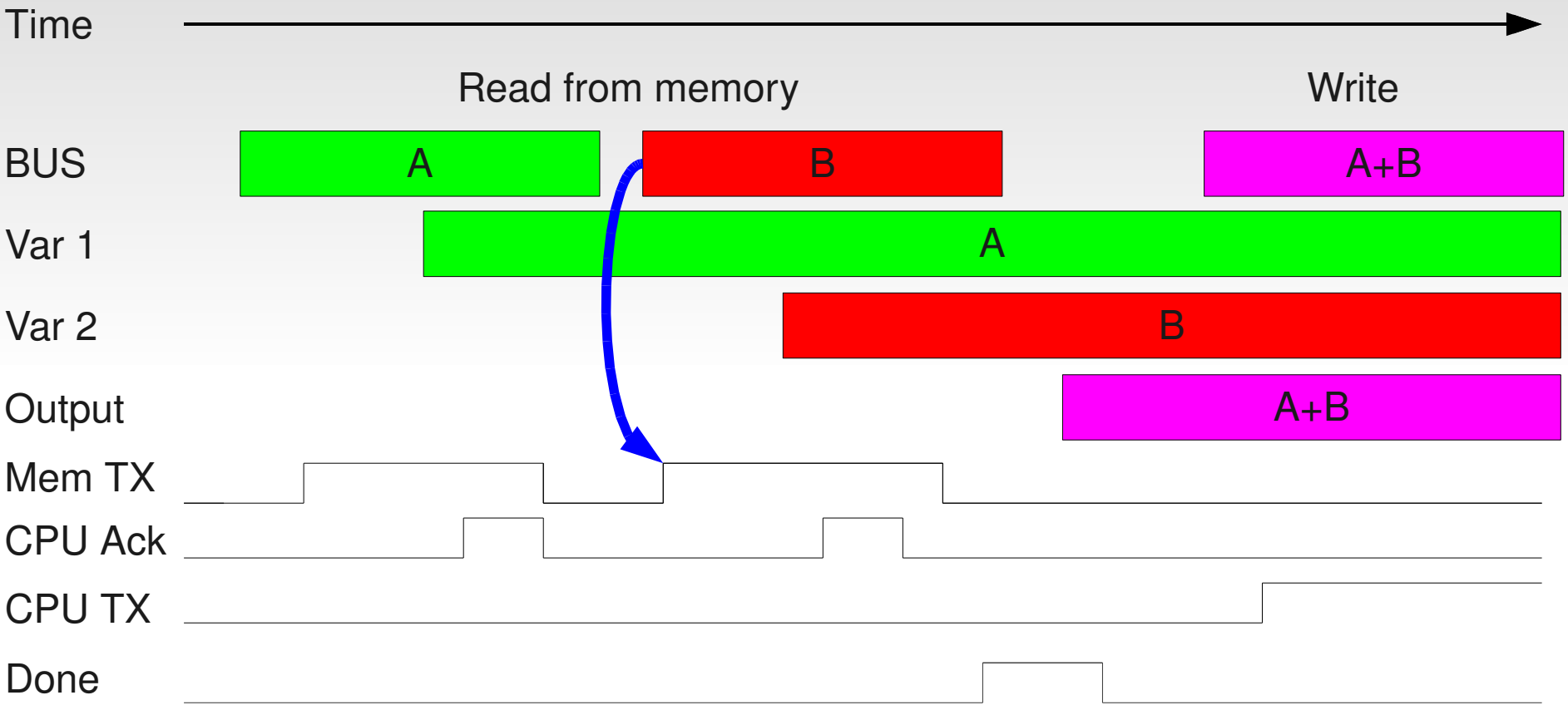


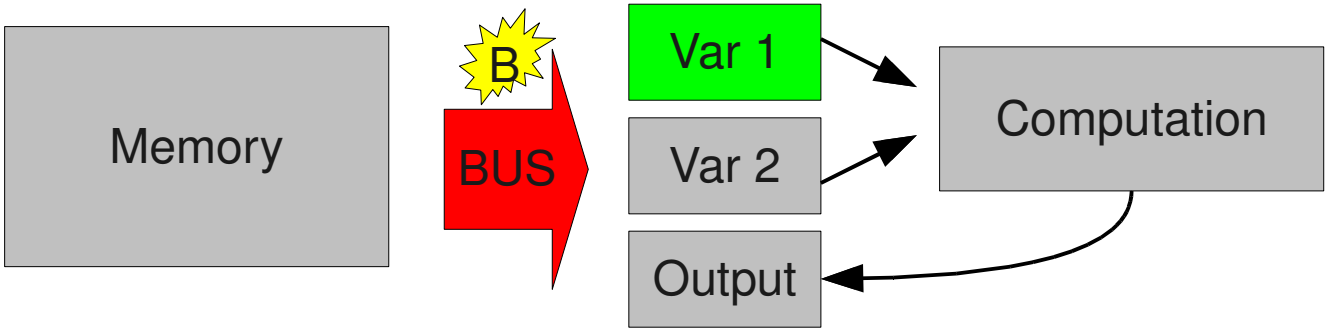
Example logic timeline (asynchronous)



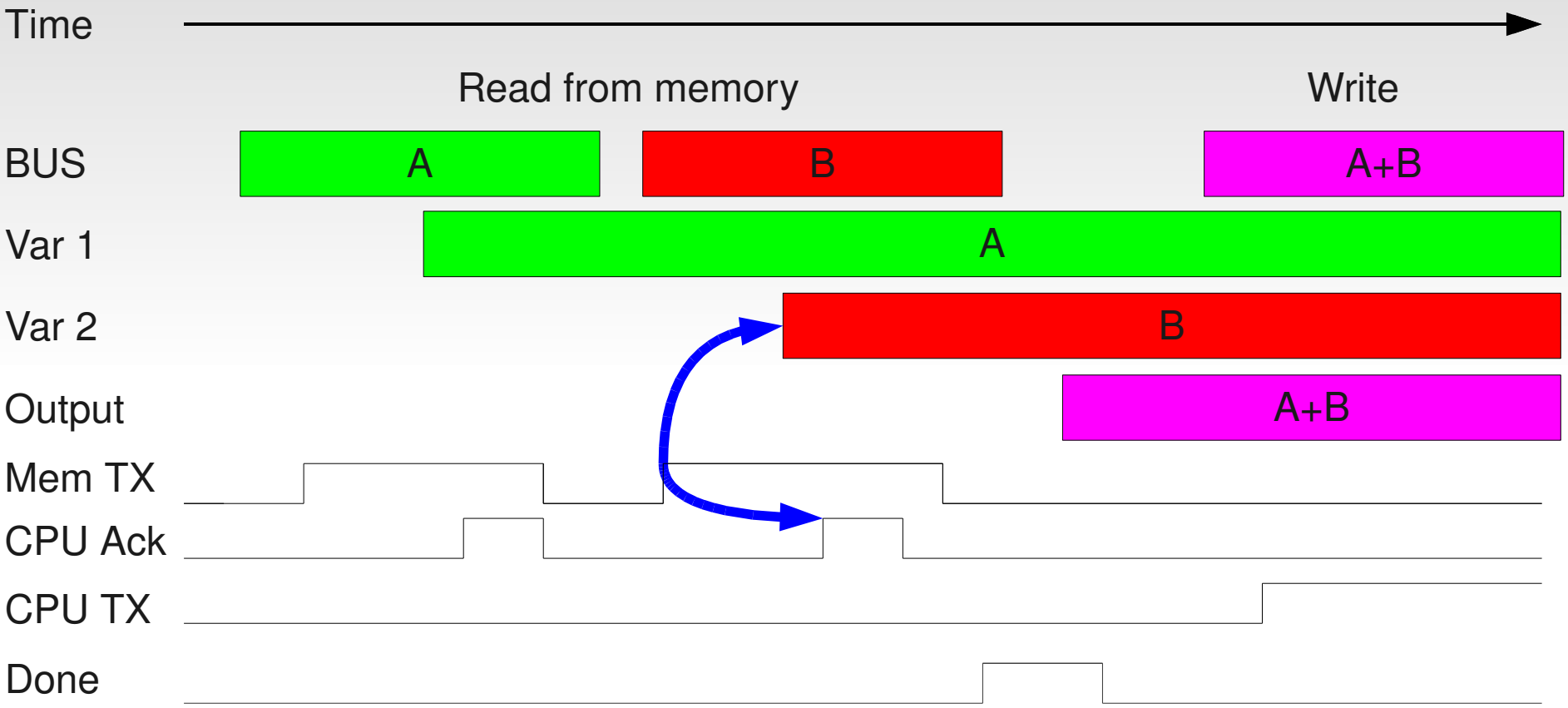


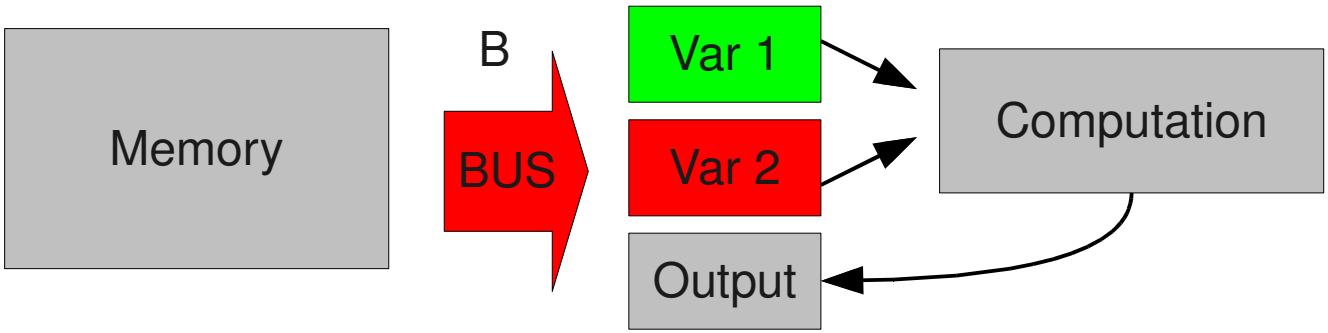
Example logic timeline (asynchronous)



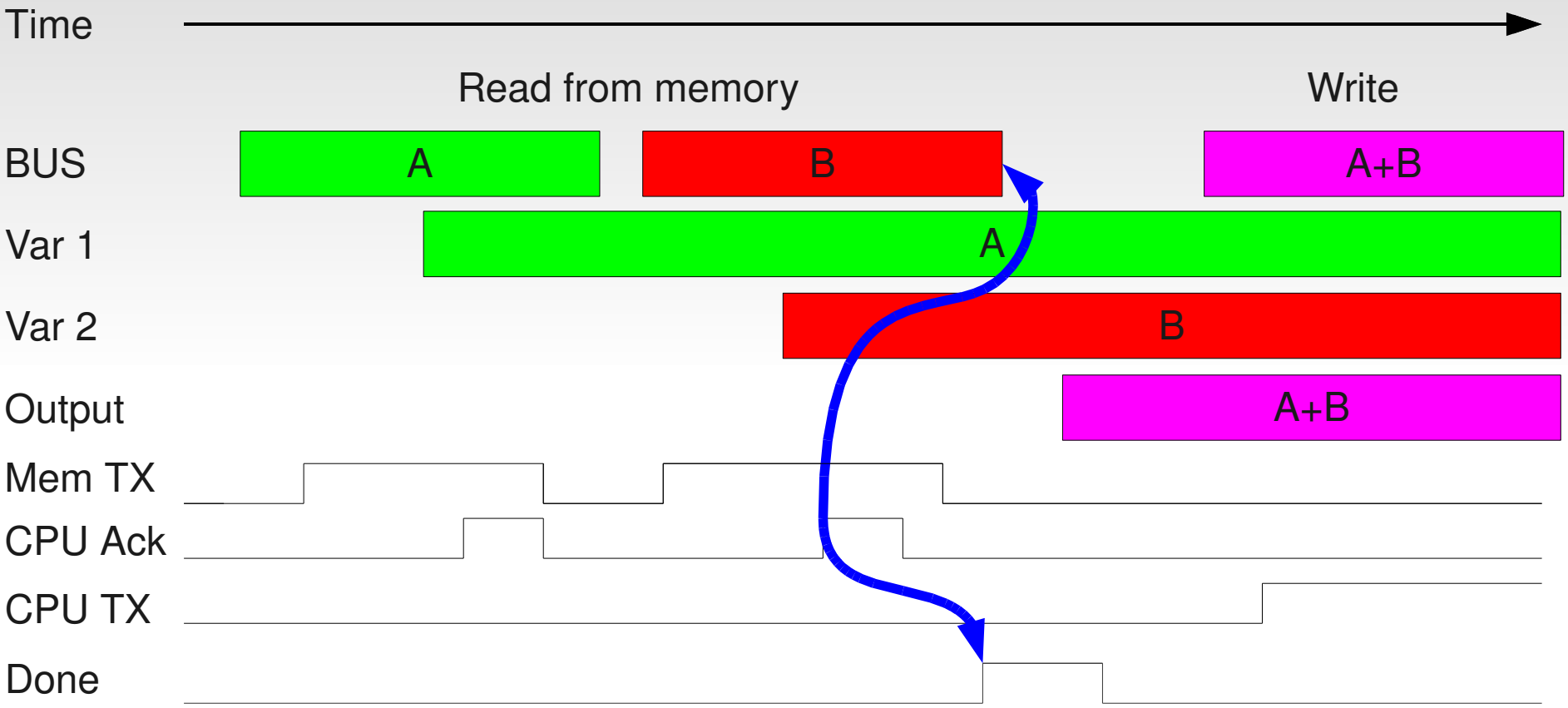


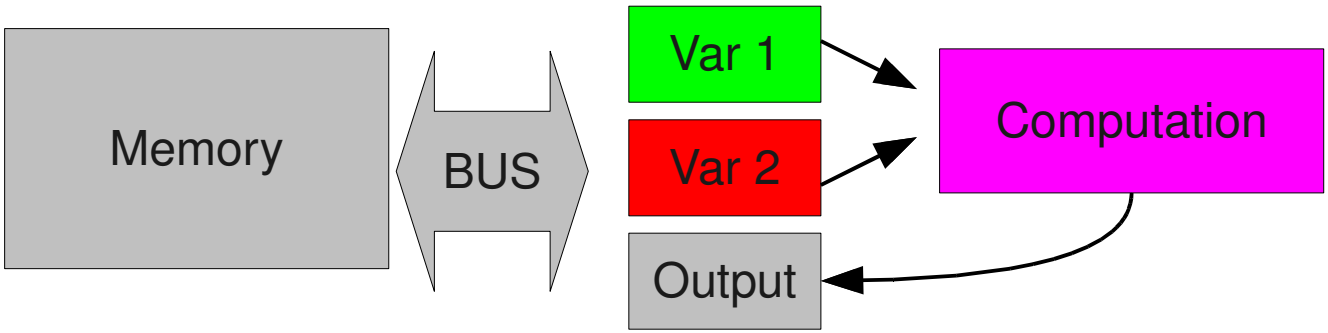
Example logic timeline (asynchronous)



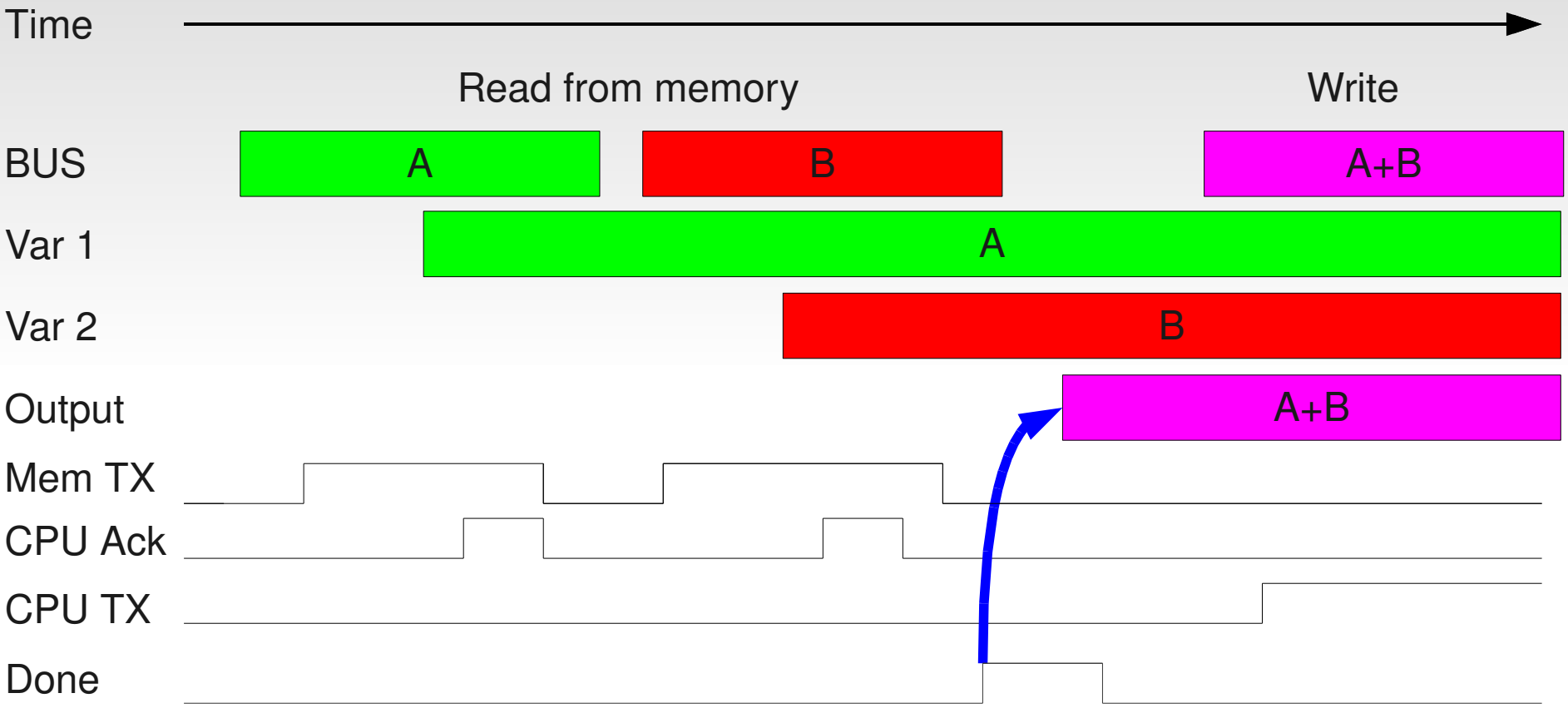


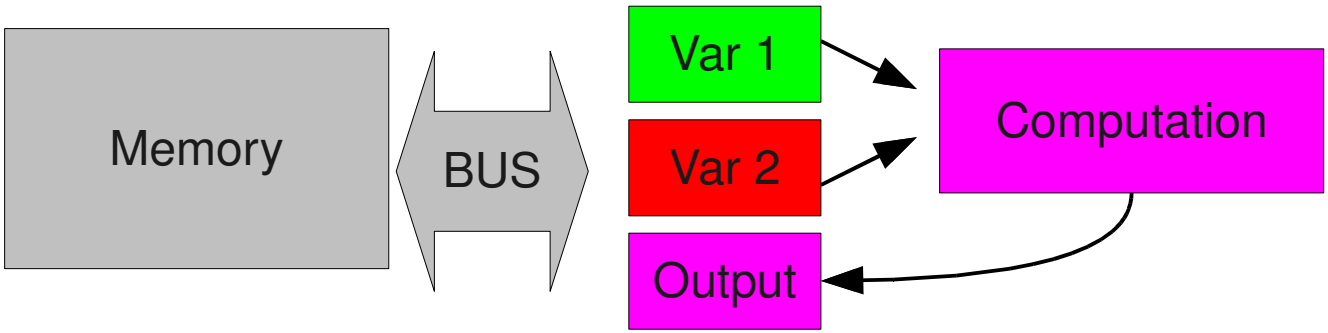
Example logic timeline (asynchronous)



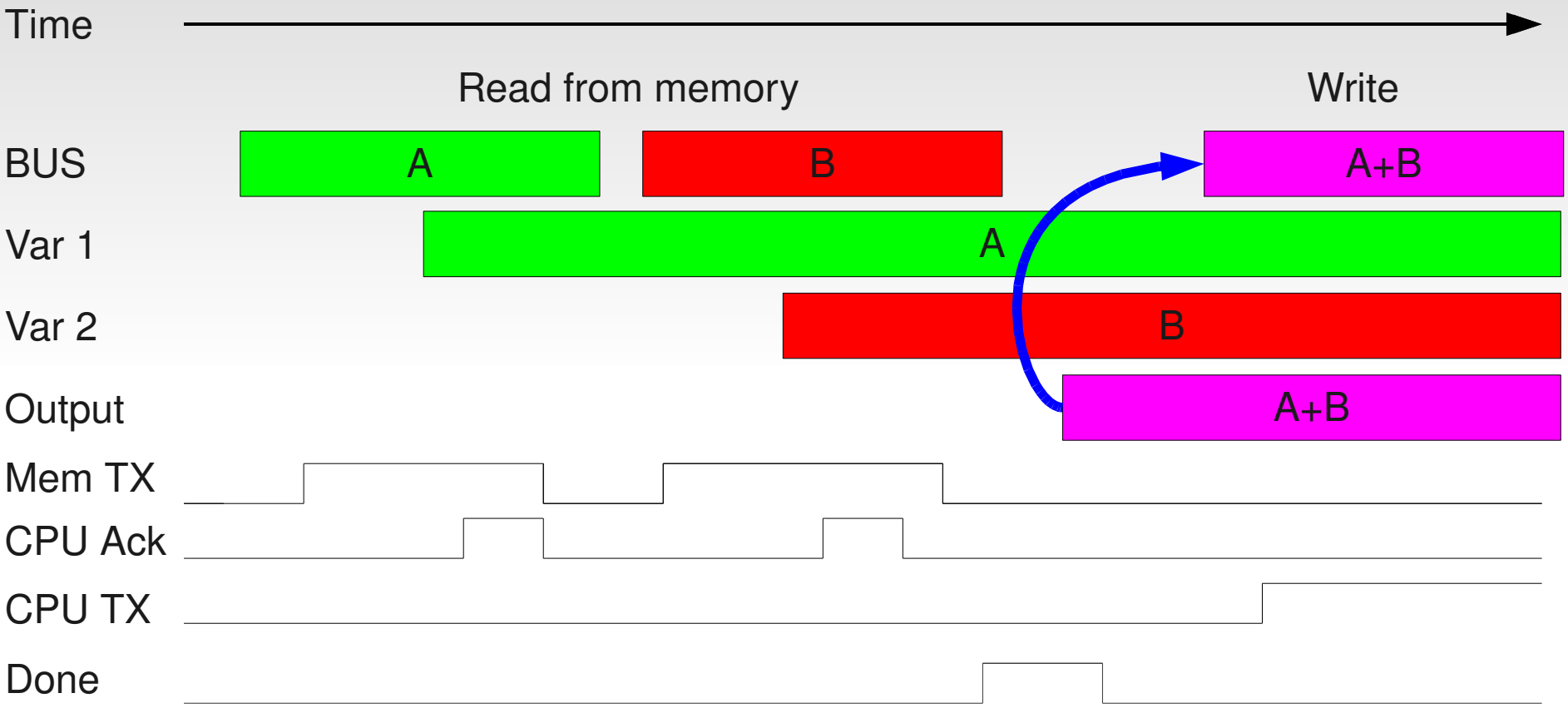


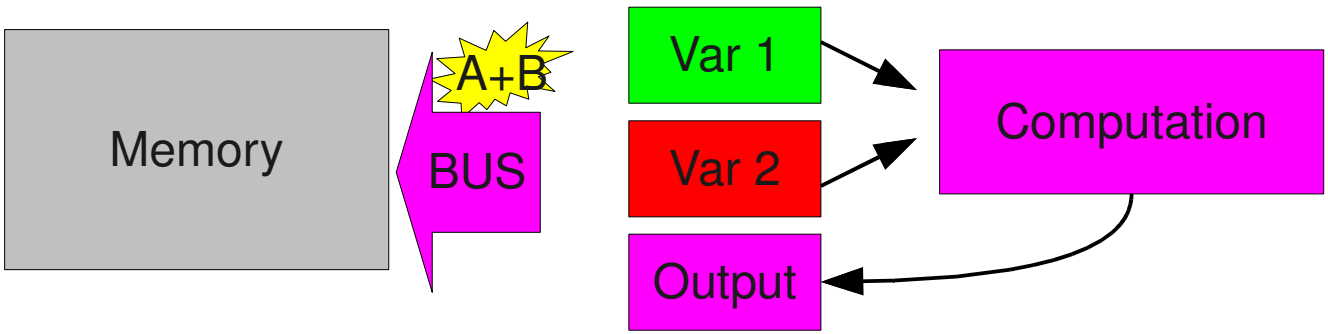
Example logic timeline (asynchronous)



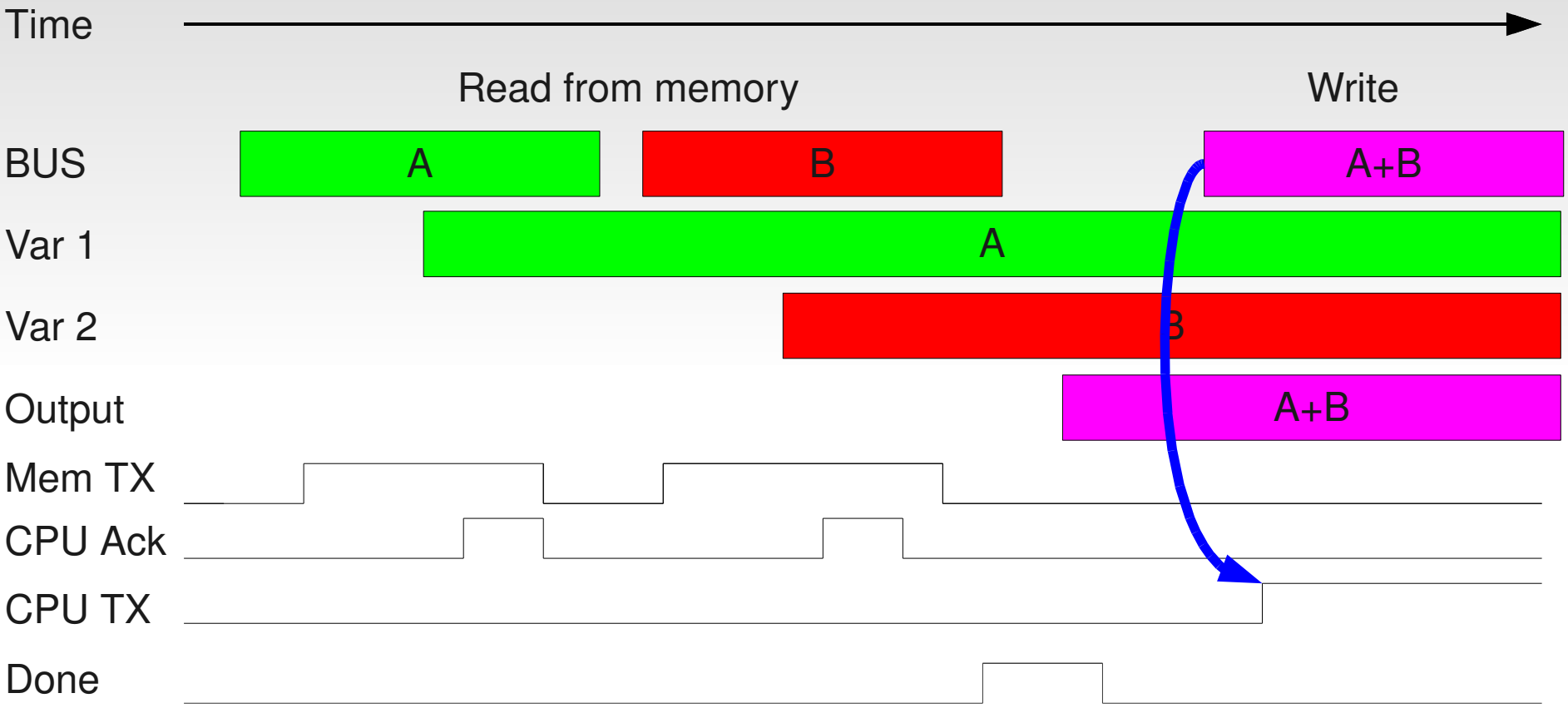


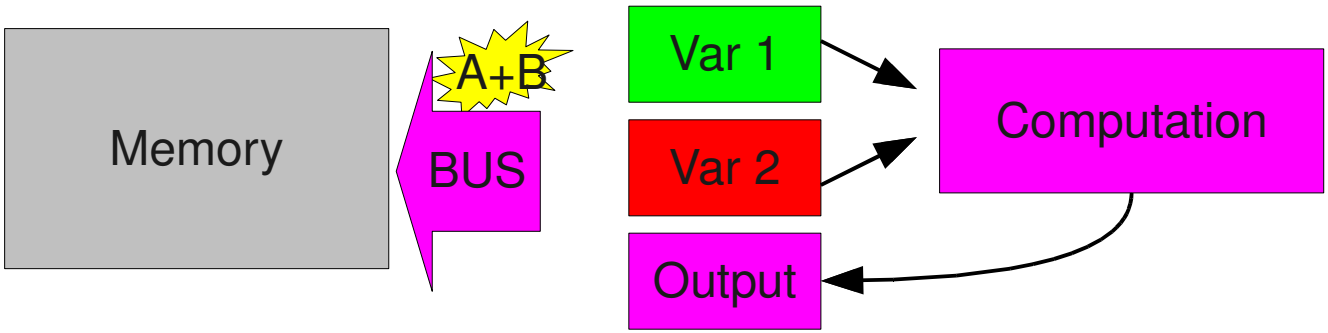
Example logic timeline (asynchronous)





Example logic timeline (asynchronous)





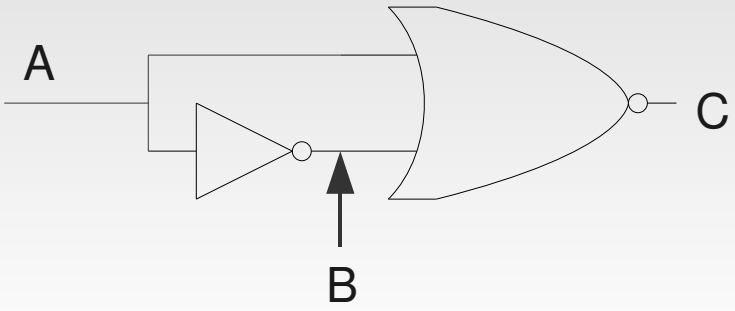
Asynchronous design

- Typical of older bus architectures and of networks
- Potential for significant power savings, space-on-die, and speed in certain areas
- Potential for better distribution of computation
- Design elegance: fewer transistors needed, less to break
- Network communication becomes more natural
  - Especially when latency is highly variable

Problems!

- Asynchronous circuits are hard to design!
- If you mistake a transient for the “final answer” of a circuit, you're faced with
  - Hazards (uncertainties in output value)
  - Glitches (very short pulses, which might confuse the underlying electronic technology)
  - Lock-ups (finite state machines getting stuck in a state where they cannot exit)
- Generally, all are the result of race conditions

Example of a glitch



Race condition between A and B causes glitch!

Limitations in current methods

- Traditional asynchronous design requires either
  - Very careful and exhaustive reasoning (time-dependent theorem-provers, concurrency theory), or
  - Detailed high-fidelity simulation (at sampling rate determined by the "GCD" of the propagation speeds)
- Bookkeeping is difficult, but essential
  - Difficult to test in stages, especially in testing *response* of circuitry to glitches
  - Exhaustive simulation is essentially impossible for large designs (*e.g.* CPUs)

Sheaf theory in logic circuits

- Provides some computational and conceptual tools
  - It's primarily a bookkeeping mechanism
- Building-up local models (gates and wires) into global ones (computational units)
  - The primary tool for this local-to-global transition is called *cohomology*
  - Sheaf cohomology organizes the computations effectively, and extracts lots of information!
  - Hierarchical design can be examined by *local* sheaf cohomology and sheaf direct image functors

Past work

A decidedly non-exhaustive list of some highlights:

- Sheaves over categories of interacting objects
  - Bacławski, Goguen (1970s)
- Concurrency & sheaf theory (not cohomological)
  - Lillius (1993), Van Glabbeek (2006)
- Constructible sheaves
  - Rota, Shapira, MacPherson (1960s)
- Quantum graphs (original motivating example)
  - Gutkin, Smilanski (2001), Kuchment (2003)
- Our focus is more strongly on *cohomology*

Sheaves: definition

A sheaf on a topological space X consists of

- A contravariant functor F from **Open**(X) to some subcategory of **Set**; this is a "sheaf of sets"
  - $F(U)$ for open U is called the *space of sections* over U
  - The inclusion map $U \subset V$ is sent to a *restriction map* $F(V) \to F(U)$. Usually it is the restriction of functions.
  - Given a point $p \in X$, the direct limit of $F(U_\alpha)$, for all U_α satisfying $p \in U_\alpha$ is called the *stalk* at p. It's a generalization of the germ of a smooth function
- And a gluing rule...

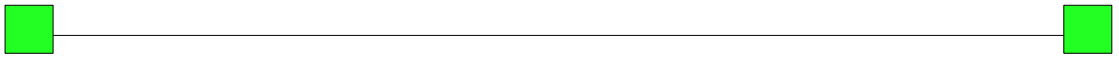
Sheaves: gluing

- The gluing rule: if U and V are open sets, then two sections defined on U and V that agree on $U \cap V$ come from a unique section defined on $U \cup V$

Sheaves: gluing

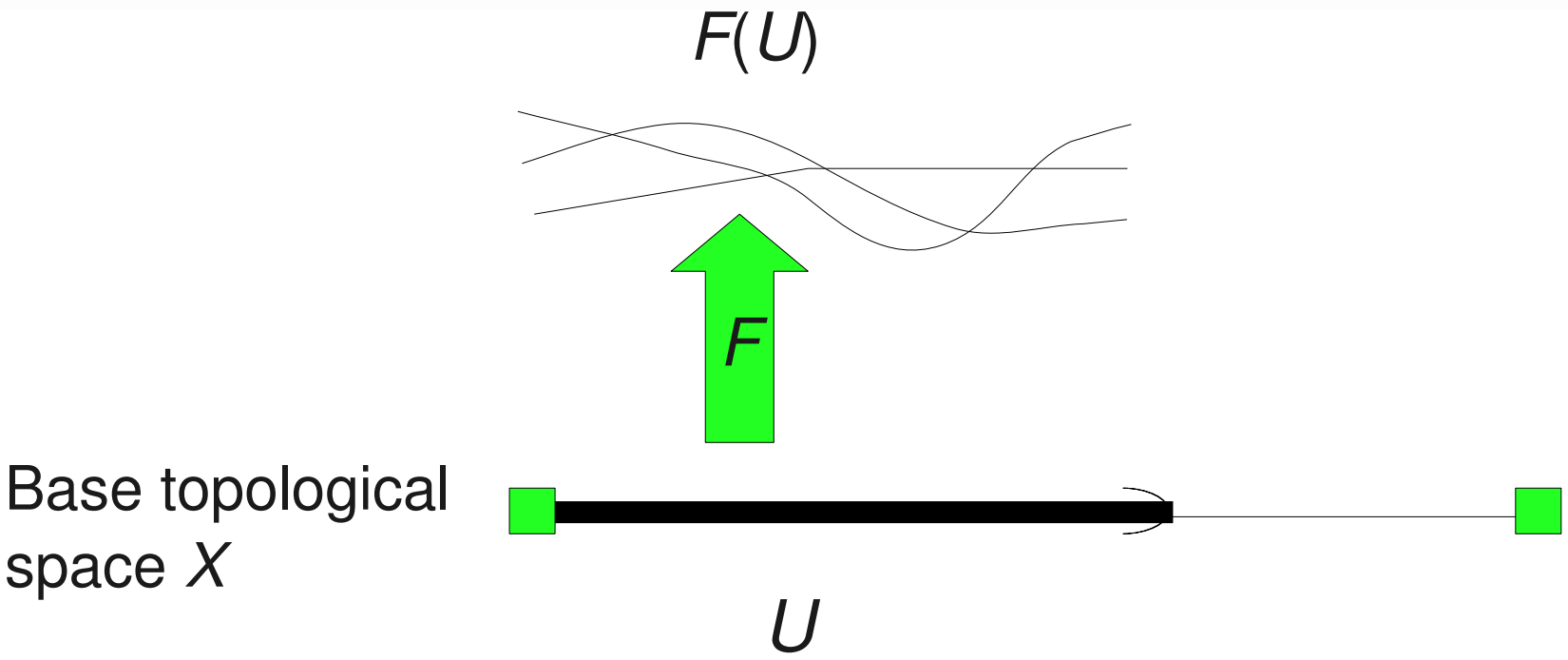
- The gluing rule: if U and V are open sets, then two sections defined on U and V that agree on $U \cap V$ come from a unique section defined on $U \cup V$

Base topological space X



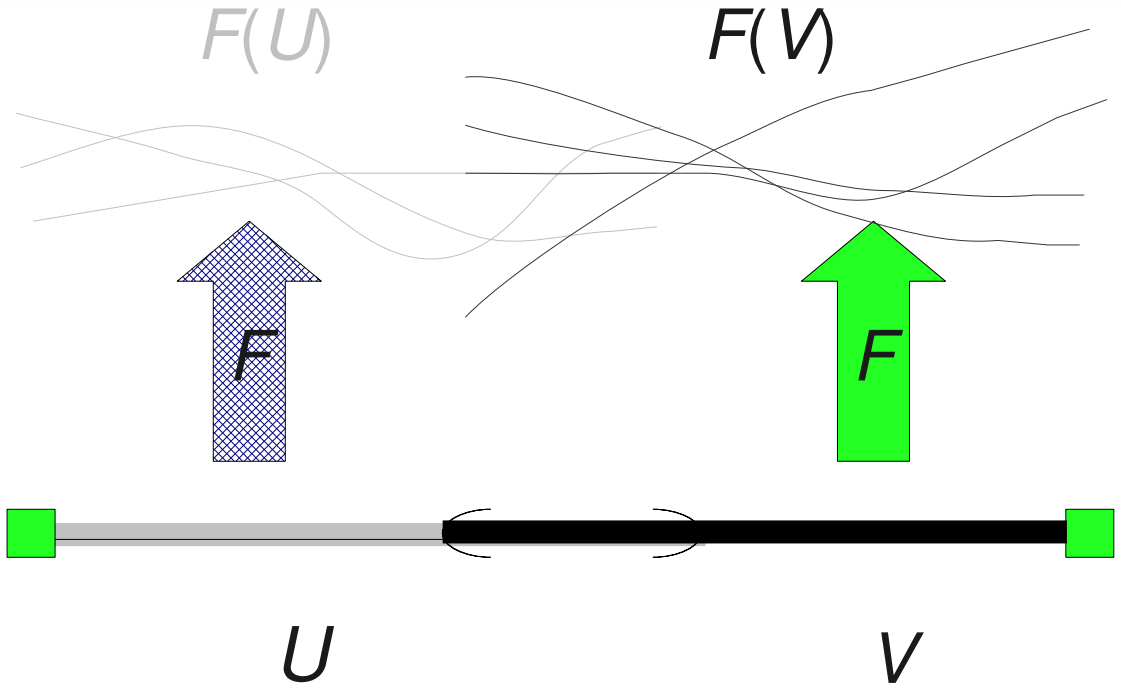
Sheaves: gluing

- The gluing rule: if U and V are open sets, then two sections defined on U and V that agree on $U \cap V$ come from a unique section defined on $U \cup V$



Sheaves: gluing

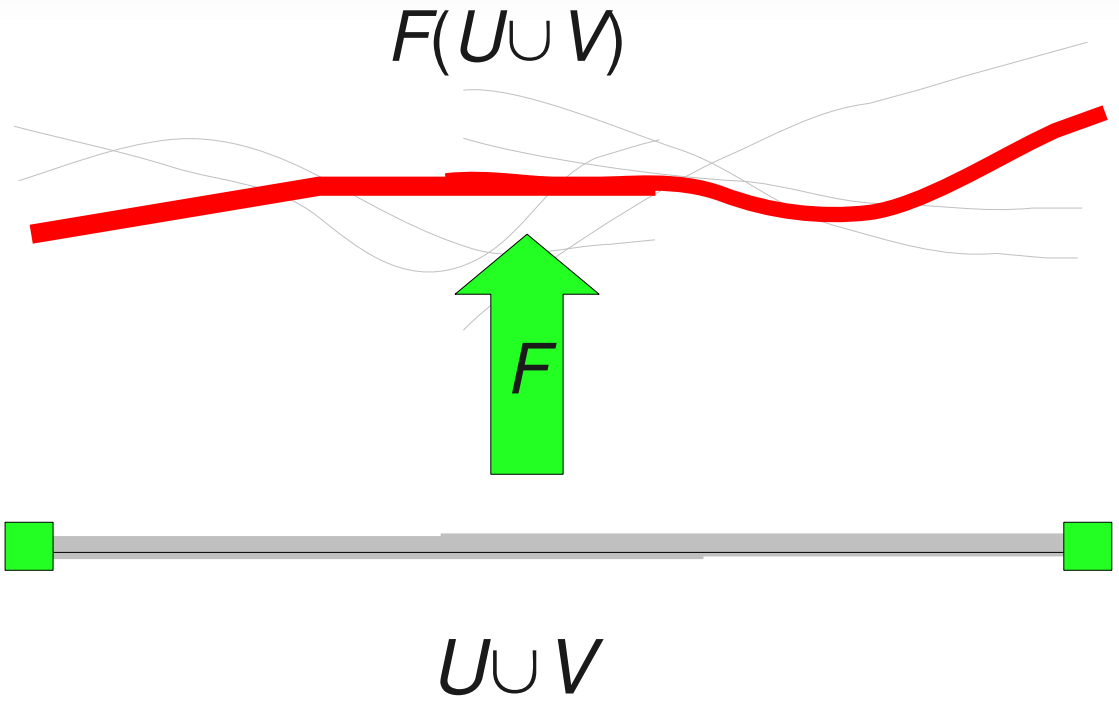
- The gluing rule: if U and V are open sets, then two sections defined on U and V that agree on $U \cap V$ come from a unique section defined on $U \cup V$



Base topological space X

Sheaves: gluing

- The gluing rule: if U and V are open sets, then two sections defined on U and V that agree on $U \cap V$ come from a unique section defined on $U \cup V$



Examples and non-examples

Examples of sheaves:

- Locally constant functions on a topological space
- Continuous functions
- Analytic functions on a manifold

Non-examples (they violate the gluing rule):

- Constant functions
- L^2 functions on unbounded domains

Constructible sheaves

- Suppose X has a filtration, $X_0 \subset X_1 \subset \ldots \subset X_k$ in which each X_i is "tame"
- A sheaf F on X is constructible (with respect to the filtration) if it is locally constant on each stratum: $X_i \backslash X_{i-1}$
- Constructible sheaves have constrained structure, especially if the filtration is finite
- In the case of topological graphs, we'll use the natural filtration structure induced by the graph

Cohomology

- The cohomology functor is a tool for extracting global information from a sheaf
  - Provided it's a sheaf of abelian groups
  - It is homotopy invariant
- It tells you all of the global sections, and obstructions for extending local sections to global ones
  - For instance $H^0(X;F) \cong F(X)$ (all global sections)

Čech cohomology

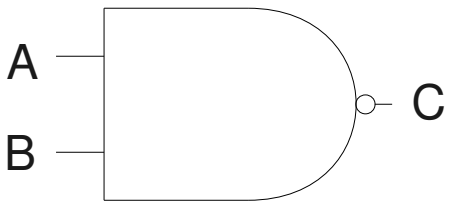
- Select a cover $\{U_\alpha\}$ of X and form the sequence of spaces and maps (the Čech cochain complex)

$$0 \to \oplus F(U_\alpha) \to \oplus F(U_\alpha \cap U_\beta) \to \oplus F(U_\alpha \cap U_\beta \cap U_\gamma) \to \ldots$$

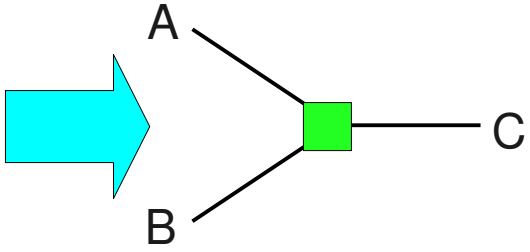
- The maps are called "coboundaries" and come from the differences between restrictions maps
- The homology of this sequence is the Čech cohomology of F with respect to $\{U_\alpha\}$.
  - Theorem: (Leray) If the cover is "good", then the Čech cohomology is a homotopy invariant, and therefore independent of the choice of cover

Problems with logic and sheaves

- If we use binary-valued ($\mathbb{Z}_2$-valued) sheaves in the obvious way, we run into a problem: most logical operations don't support the functoriality of any sheaf in a way that's compatible with cohomology
  - Put another way, logical operations aren't all $\mathbb{Z}_2$-linear!



Logic circuit



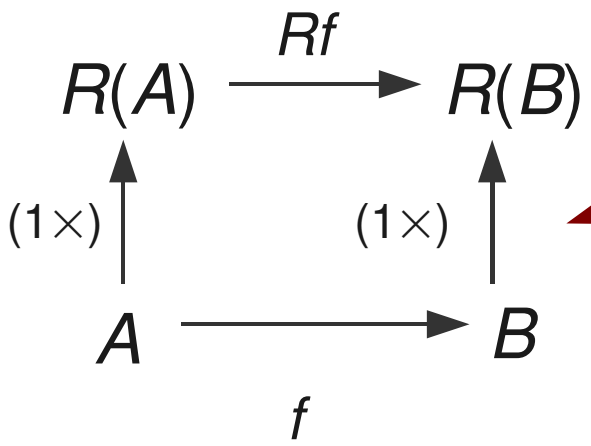
Connection graph

Not linear!

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

A (standard) algebraic trick!

- Instead, consider any function between sets $f{:}A \to B$
- Let R be a ring with unit, and $R(A)$ be the R-module generated by A
  - That is, generators of $R(A)$ are elements of A
- Then f lifts uniquely to an R-module homomorphism

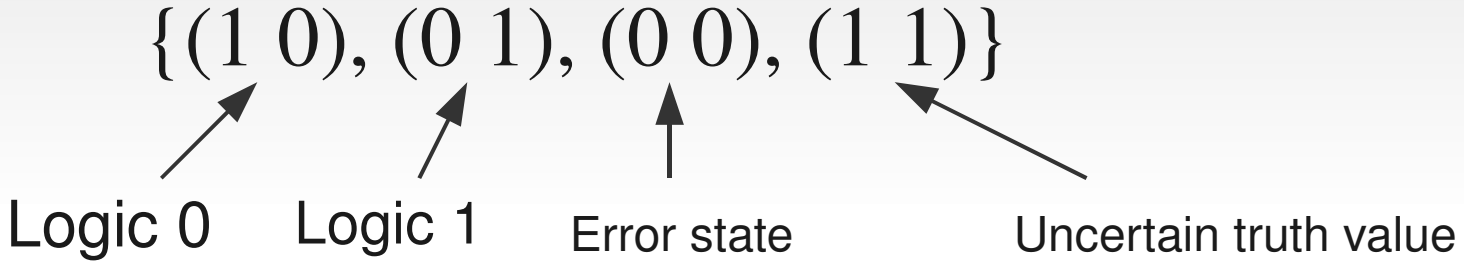


Notice that generally we cannot recover a unique element of B from R(B).

But we can if we've used $Rf \circ (1\times)$

Lifted logic values

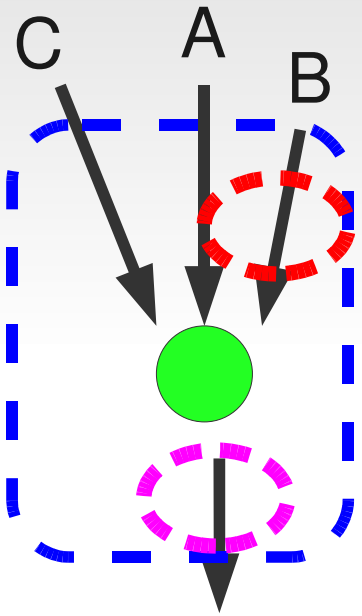
- Our logical value is represented by an element of $\mathbb{Z}_2^{\,2}$ (or R^2 where R is a ring with unit):



- Put another way, a logical value is $aq+bQ$, where
  - q=(1 0), represents a logic 0
  - Q=(0 1), represents a logic 1
  - $a,b\in\mathbb{Z}_2$ can be interpreted as a flag of whether Q or its inverted copy q is a *possible realization* of this value

Switching sheaves

- A *switching sheaf* over a directed graph is constructible with respect to stratification by the graph structure and
  - Stalks over points in an edge are $\mathbb{Z}_2^2$
  - A stalk over a vertex is the tensor product of n copies of $\mathbb{Z}_2^2$, where n is the incoming degree
  - Restriction maps from an open set containing a single vertex to a connected set in the interior of an edge are given by the diagram at right



Contraction of A, C

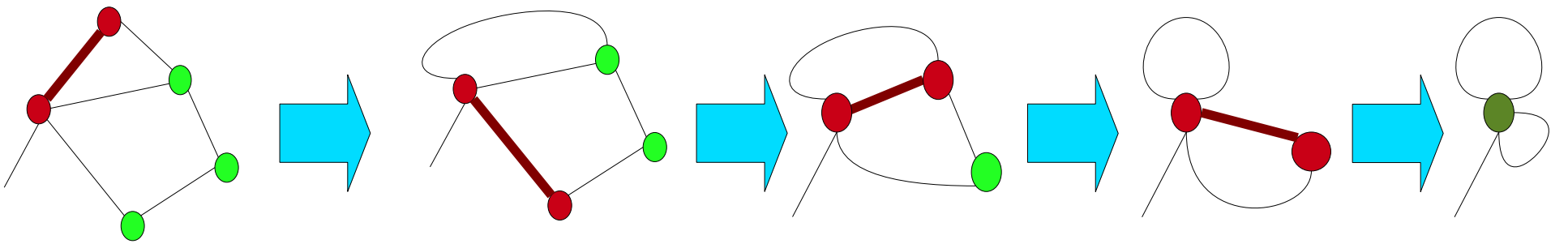
The lift into $\mathbb{Z}_2^8 \to \mathbb{Z}_2^2$ of a logic function

Edge collapse

- The benefit of the sheaf formalism is that useful sheaf functors are already well-known.
- The direct image functor (pushforward) relates to hierarchical design:
- Consider a continuous map $X \to Y$ that collapses an edge with distinct ends. This takes a constructible sheaf F on X to a constructible sheaf $f_* F$ on Y.
  - For switching sheaves, this also induces an isomorphism on cohomology (by the Vietoris mapping theorem)
- Big win conceptually and computationally!

Collapsed graphs

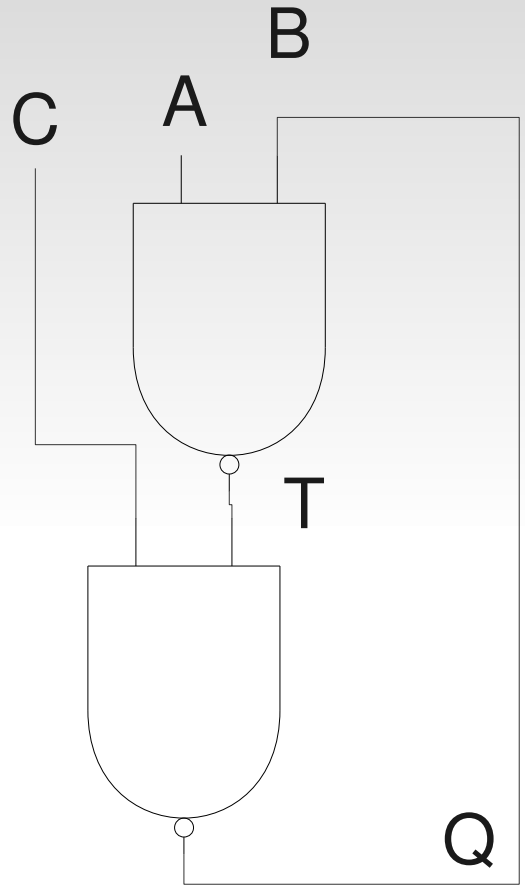
- We construct a spanning tree T for X, and a sequence of trees $T_1, T_2, \dots, T_N = T$ such that $T_{i+1} \setminus T_i$ consists of exactly one edge
- We can work with collapsed graphs X/T_i, on which the cohomology is easier to compute
  - Vietoris Mappring theorem: isomorphic cohomology



Cohomology of switching sheaves

- As noted earlier, $H^0(X;F) \cong F(X)$, so H^0 is generated by all of the allowable states of the logic circuit
  - Switching sheaves don't incorporate time explicitly, but one can still extract time-dependent information in H^0...
  - Appears to track hazard-related transitions between states
- $H^k(X;F)=0$ for $k>1$, since dim $X = 1$
- $H^1(X;F)$ appears to describe the states related to hazards

Example: flip-flop

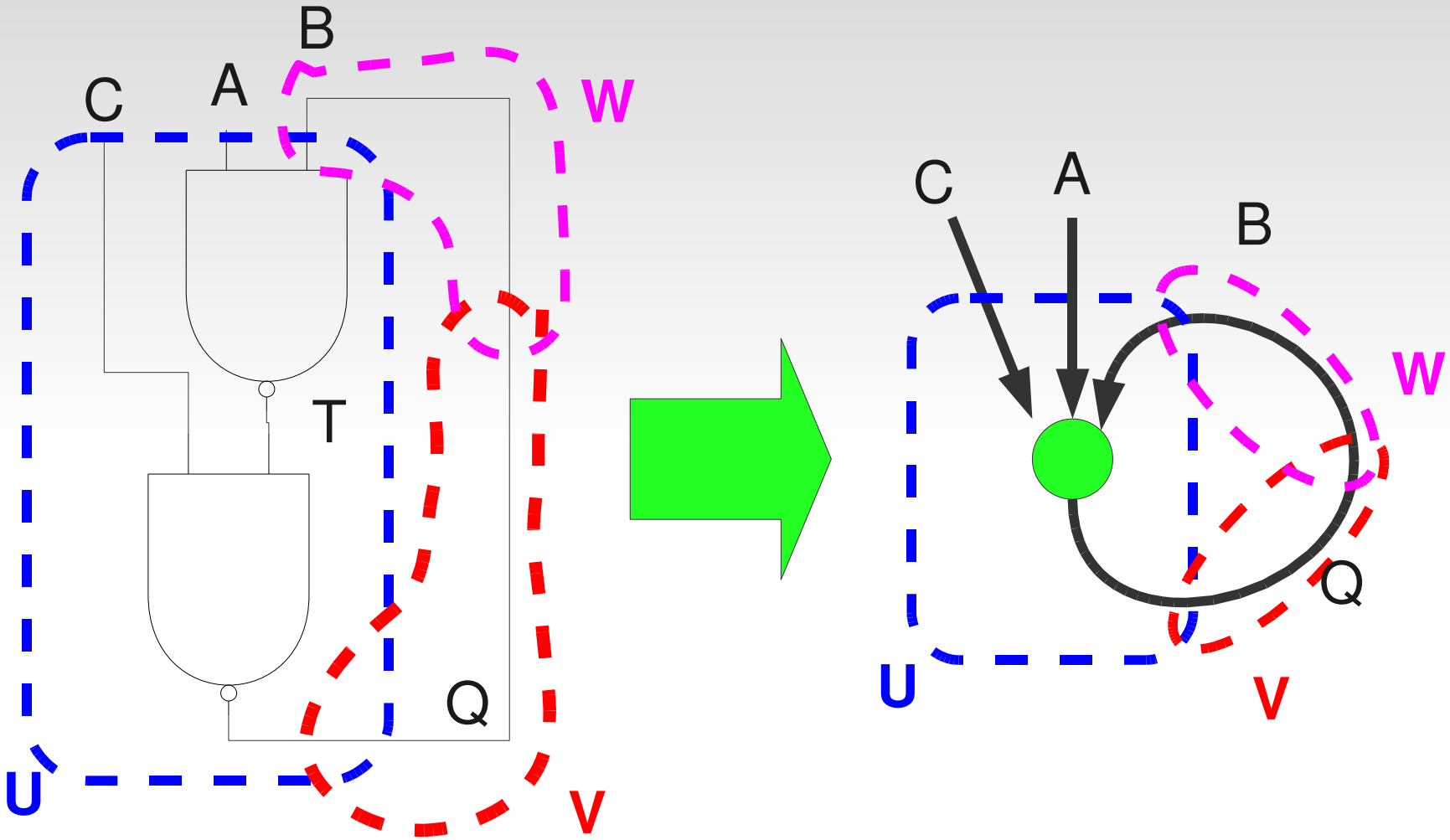


C	A	B	T	Q	
0	0	0	1	1	
0	0	1	1	1	Hazard!
0	1	0	1	1	
0	1	1	0	1	Set
1	0	0	1	0	Reset
1	0	1	1	0	
1	1	0	1	0	Hold
1	1	1	0	1	

Transition out of the hazard state to the a hold state causes a race condition

This is what traditional analysis gives...
5 possible states

Conversion to graph

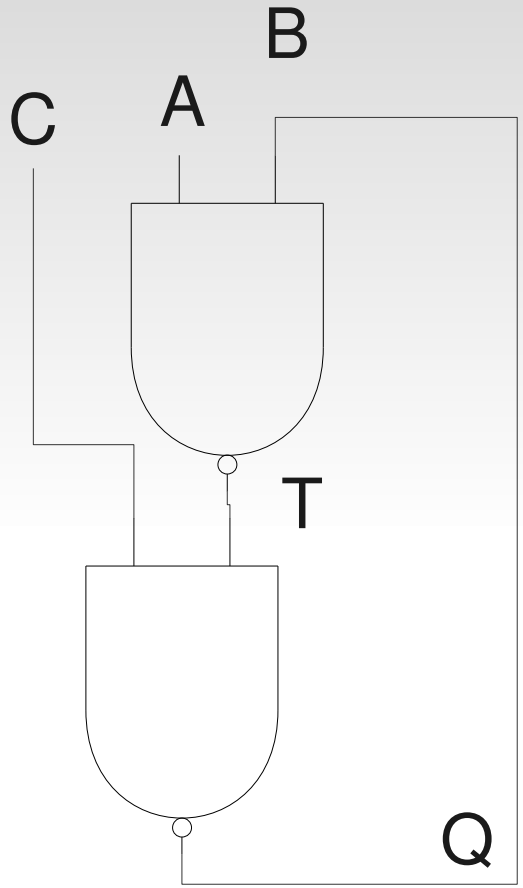


Čech cochain complex:

$$0 \to F(U) \oplus F(V) \oplus F(W) \to F(U \cap V) \oplus F(U \cap W) \oplus F(V \cap W) \to 0$$

$$0 \to \mathbb{Z}_2^{\,8} \oplus \mathbb{Z}_2^{\,2} \oplus \mathbb{Z}_2^{\,2} \to \mathbb{Z}_2^{\,2} \oplus \mathbb{Z}_2^{\,2} \oplus \mathbb{Z}_2^{\,2} \to 0$$

Flip-flop cohomology



$H^1(X;F) \cong \mathbb{Z}_2$

$H^0(X;F) \cong \mathbb{Z}_2^{\,7}$

Generated by:

a⊗B⊗c
A⊗B⊗c
a⊗b⊗C
A⊗b⊗C
A⊗B⊗C

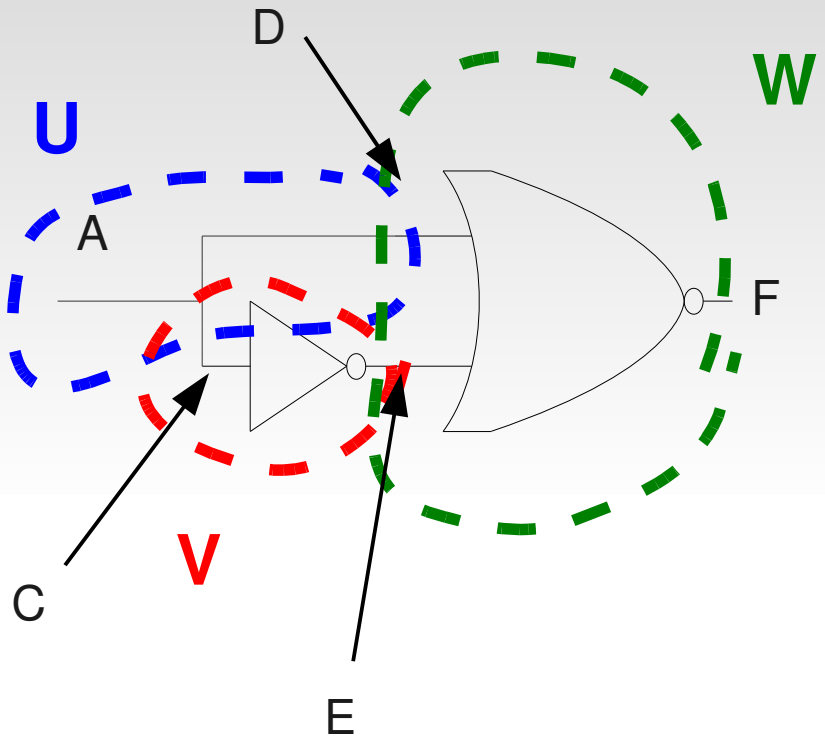
a⊗b⊗c+a⊗B⊗C
a⊗b⊗c+A⊗b⊗c

These states describe the possible transitions out of the hazard state – something that takes a bit more trouble to obtain traditionally

States from the traditional approach

(upper case means the generator corresponding to logical 1)

Example: glitch generator



$H^0(X;F)$ is generated by

A+C+D⊗e
a+c+d⊗E

A+a+C+c+d⊗e+D⊗E

$H^1(X;F)\cong\mathbb{Z}_2$

Hazard transition state

Čech cochain complex:

$$0\to F(U)\oplus F(V)\oplus F(W)\to F(U\cap V)\oplus F(U\cap W)\oplus F(V\cap W)\to 0$$

$$0\to \mathbb{Z}_2{}^2\oplus\mathbb{Z}_2{}^2\oplus\mathbb{Z}_2{}^4\to\mathbb{Z}_2{}^2\oplus\mathbb{Z}_2{}^2\oplus\mathbb{Z}_2{}^2\to 0$$

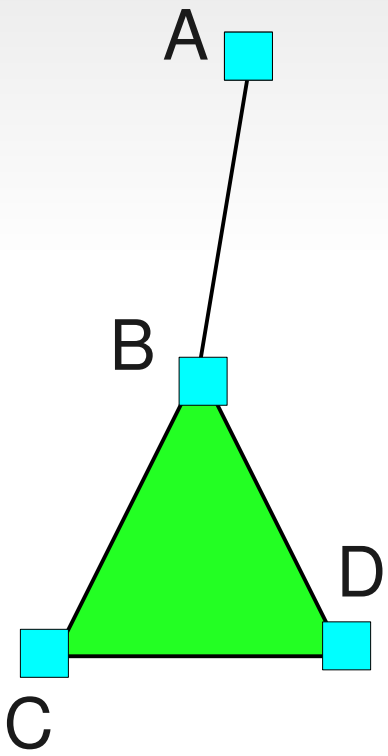
Computational aspects

- It's not immediately clear how one might store a (representation of a) constructible sheaf in a computer
- One needs to specify a vector space for each open set; there are various ways of doing this
  - The most obvious way to do this is to write the sheaf as a function, but then how does one store a vector space?
  - Possibly use type-level programming in Haskell? We could instantiate the sheaf as a type of Functor...
- Seriously, though, it seems to be an impediment to automating computation in constructible sheaves

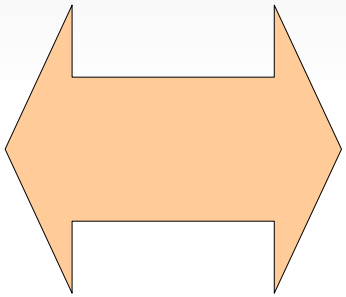
Category theory to the rescue!

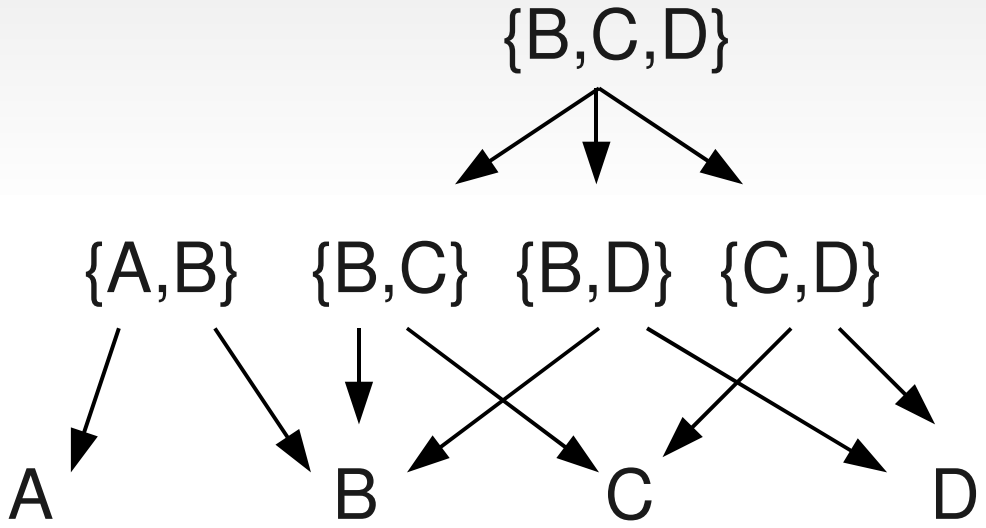
- It turns out that there's a different way:
- Theorem: (MacPherson) The category of constructible sheaves on an abstract simplicial complex K is isomorphic to the category of *pre*sheaves over a certain category associated to K
  - By presheaf, we mean a contravariant functor from a category to a subcategory of **Set**
  - The category in question here is the face category: objects are simplices, and morphisms describe boundaries (*i.e.* $A \to B$ if B is a face of A)

Simplicial complexes and the face category



Simplicial complex





Face category

Presheaves on a face category

- If our graph is a cell complex, we therefore only need to know the restriction maps and the stalks over each cell.
- This seems like a minimal amount of information
- Further, the construction is functorial, so we can transfer computation of sheaf cohomology to this context
- This relates to HDA in concurrency theory!

What's next?

- Theoretical directions
  - Figure out how exactly glitches and hazards are represented in the cohomology of a switching sheaf
  - Related: what is the physical meaning of $H^1(X;F)$?
  - Extend edge collapse methodology to other direct images; aiming towards a hierarchical approach to sheaf cohomology computation
- Computational directions
  - Run some more complicated examples of cohomology computations
  - Implement the cohomology computation on a computer