Logic Synthesis and Implementation Styles in Asynchronous Circuits Design

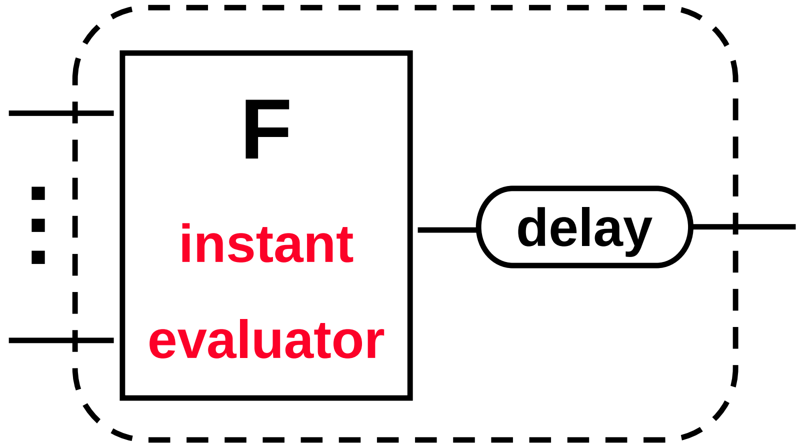
Victor.Khomenko@ncl.ac.uk

School of Computing Science,

Newcastle University, UK

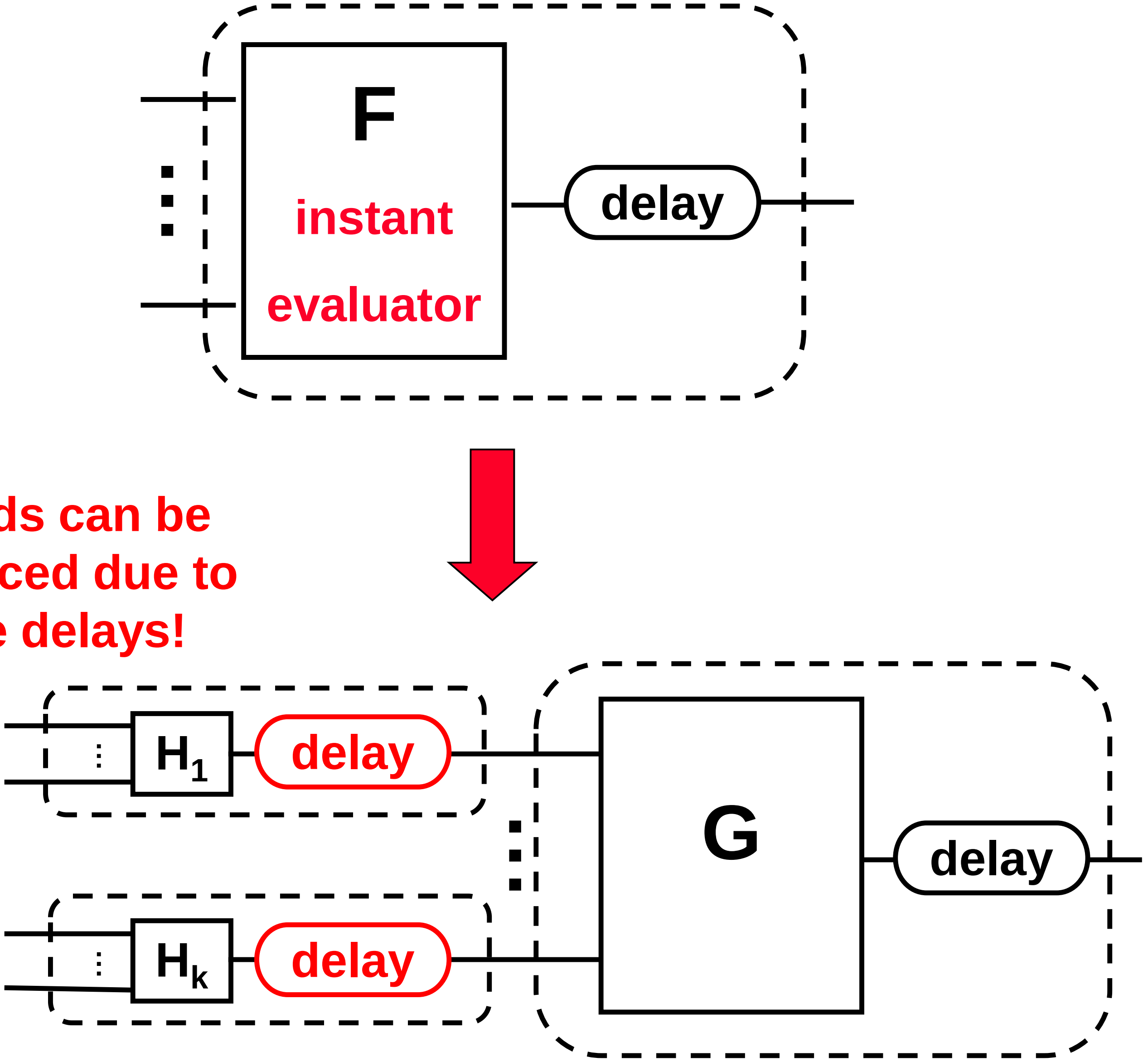
Speed-independence assumptions

- Gates/latches are *atomic* (so no internal hazards)



- Gate delays are positive and finite, but variable and unbounded
- Wire delays are negligible (SI)
- Alternatively, [some] wire forks are isochronic (QDI), i.e. wire delays can be added to gate delays

SI decomposition



Hazards can be introduced due to these delays!

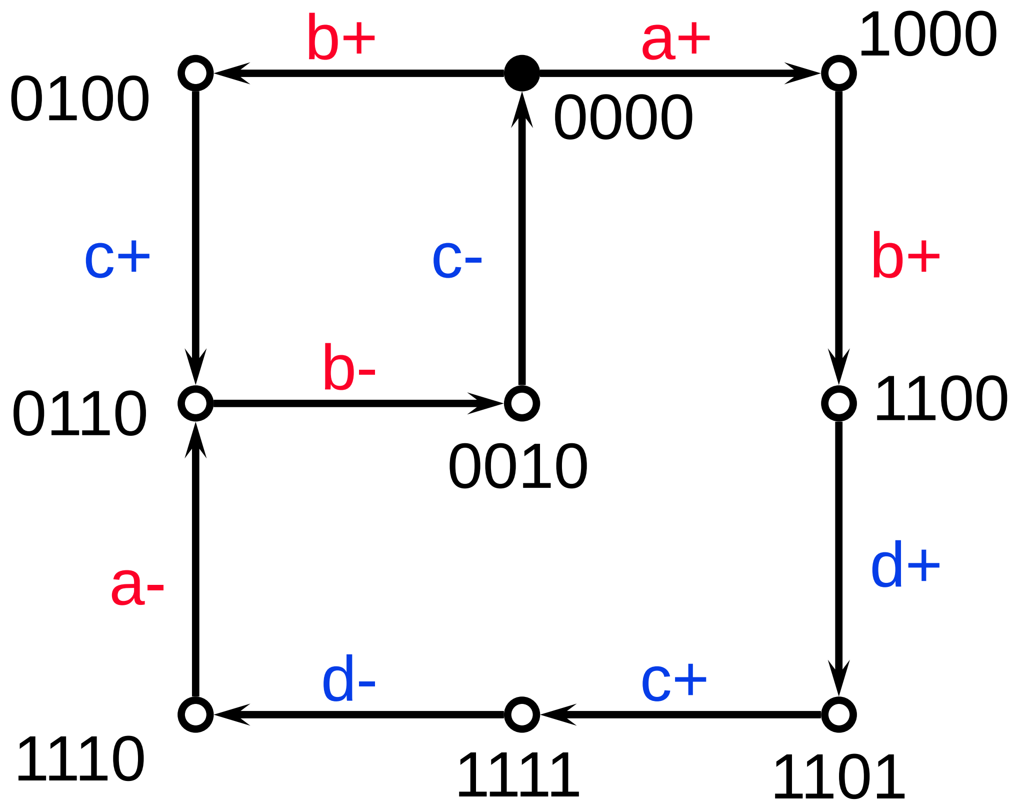
Gates & latches

- Good citizens: unate gates/latches, e.g. BUFFER, AND, OR, NAND, NOR, AND-OR, OR-AND, C-element, SR-latch, RS-latch
  - Output inverters ('bubbles') can be used liberally, e.g. NAND, NOR, as the invertor's delay can be added to the gate's delay
  - Input inverters are suspect as they introduce delays, but in practice are ok if the wire between the inverter and the gate is short
- Suspects: binate gates, e.g. XOR, NXOR, MUX, D-latch – may have internal hazards, but may still be useful

Logic synthesis

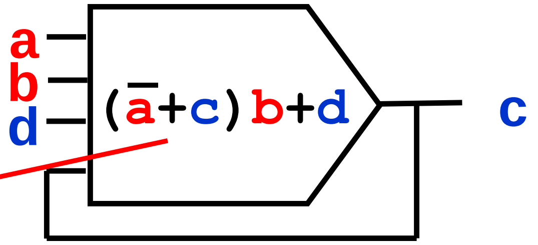
- Encoding (CSC) conflicts must be resolved first
- Several kinds of implementation can then be derived automatically:
  - complex-gate (CG)
  - generalised C-element (gC)
  - standard-C implementation (stdC)
- Can mix implementation styles on per-signal basis
- Logic decomposition may still be required if the gates are too complex

Example: complex-gate synthesis



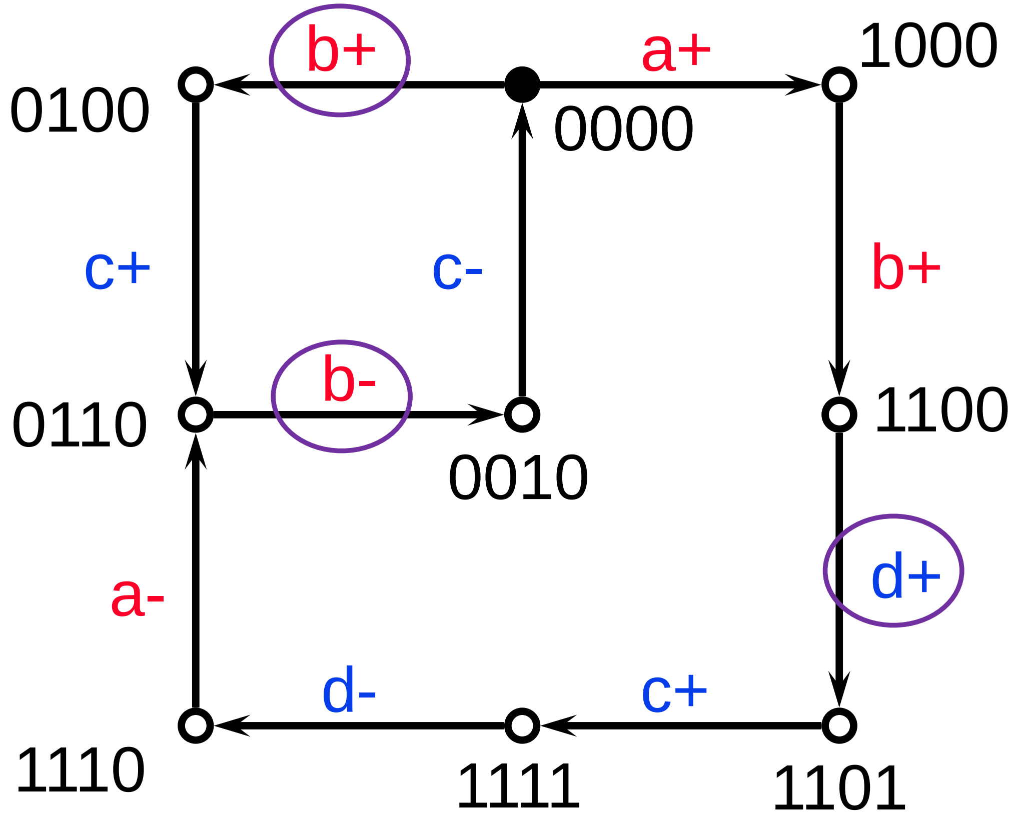
Code	Nxt_c
0100	1
0000	0
1000	0
0110	1
0010	0
1100	0
1110	1
1111	1
1101	1
else	-
Eqn	$(\overline{a}+c)b+d$

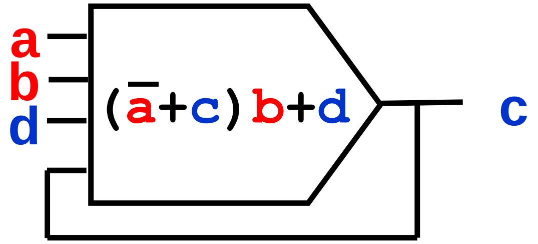
$$Nxt_z(s) = Code_z(s) \oplus Out_z(s)$$



The size of this Boolean expression is not limited!

Support, triggers and context





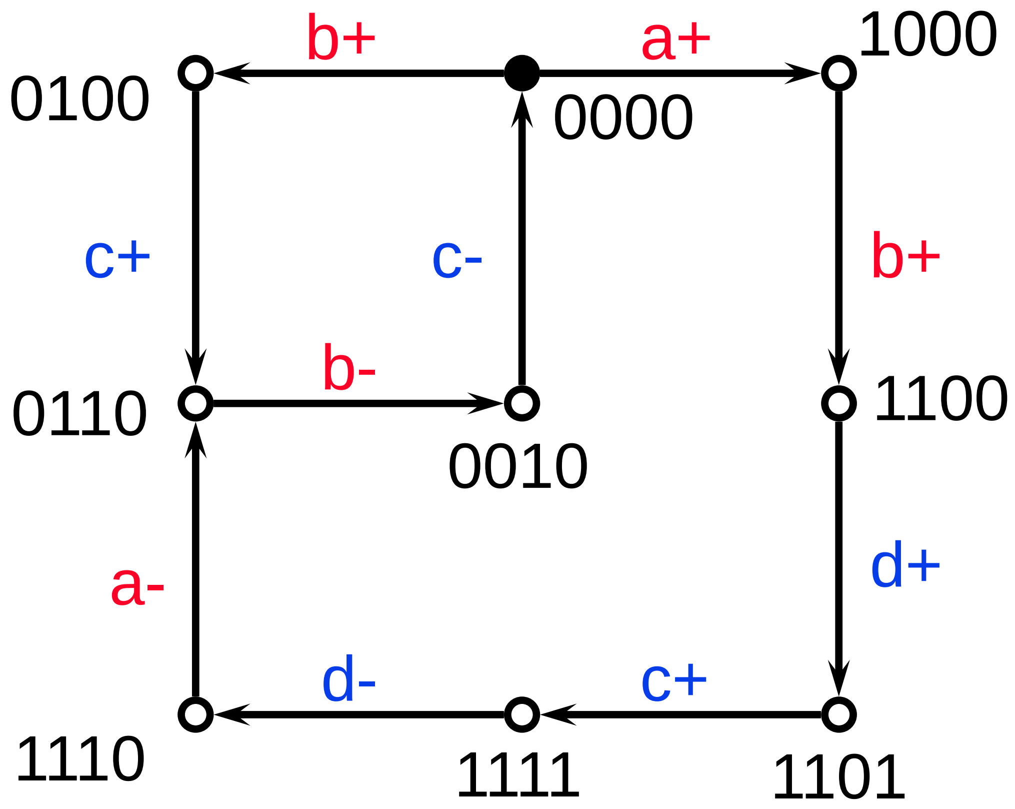
Signals that are the inputs of the gate producing a signal form its **support**, e.g. the support of c is {a,b,c,d}. Supports are not unique in general.

Signals whose occurrence can immediately enable a signal are called its **triggers**, e.g. the triggers of c are {b,d}. Triggers are unique, and are always in the support.

Signals in the support which are not triggers are called the **context**, e.g. the context of c is {a,c}. Context is not unique in general.

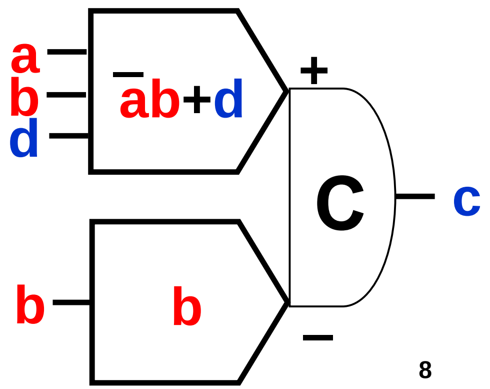
support = triggers + context

Example: gC implementation



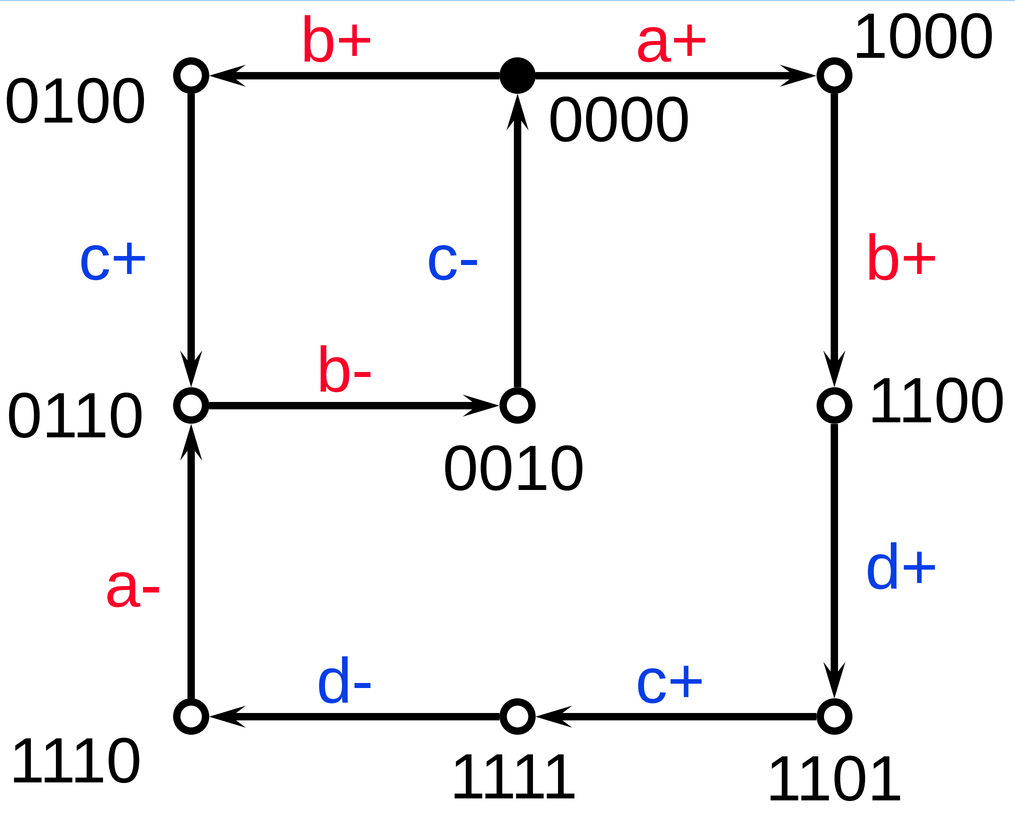
Code	Set_c	$Reset_c$
0100	1	0
0000	0	–
1000	0	–
0110	–	0
0010	0	1
1100	0	–
1110	–	0
1111	–	0
1101	1	0
else	–	–
Eqn	$\bar{a}b+d$	$\bar{b}$

$$Set_z(s) = \begin{cases} 1 & \text{if } Out_{z+}(s) = 1 \\ 0 & \text{if } Nxt_z(s) = 0 \\ - & \text{otherwise} \end{cases} \quad Reset_z(s) = \begin{cases} 1 & \text{if } Out_{z-}(s) = 1 \\ 0 & \text{if } Nxt_z(s) = 1 \\ - & \text{otherwise} \end{cases}$$

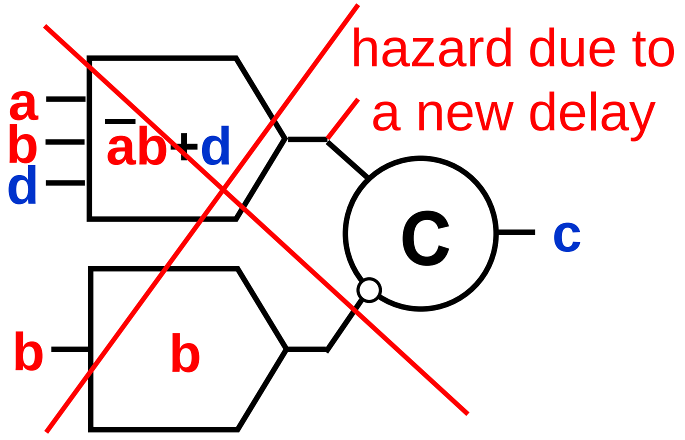


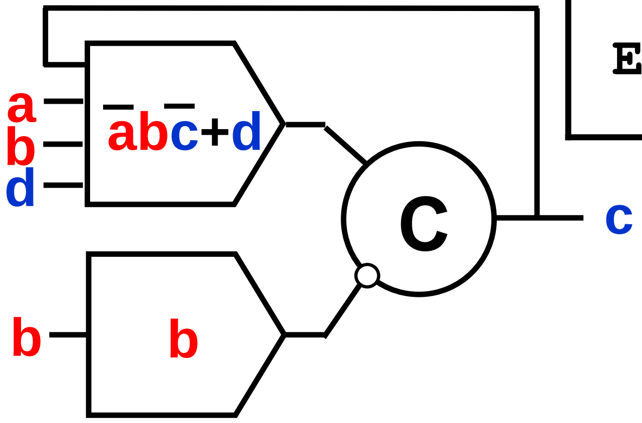
Implemented as pull-up and pull-down networks of transistors and a ‘keeper’; assumed to be **atomic**

Example: stdC implementation



Code	Set_c	$Reset_c$
0100	1	0
0000	0	–
1000	0	–
0110	–	0
0010	0	1
1100	0	–
1110	–	0
1111	–	0
1101	1	0
else	–	–
'Monotonic cover' constraints		
Eqn	$\bar{a}b\bar{c}+d$	$\bar{b}$





Logic Decomposition

- **Often complex-gates are too complex to be mapped to a gate library, and so logic decomposition is required**
- **Cannot naïvely break up complex-gates – this is likely to introduce hazards (at least, timing assumptions are required)**
- **Decomposition is one of the most difficult tasks – no guarantee that automatic decomposition will succeed**
- **Online tutorial on logic decomposition and technology mapping:**

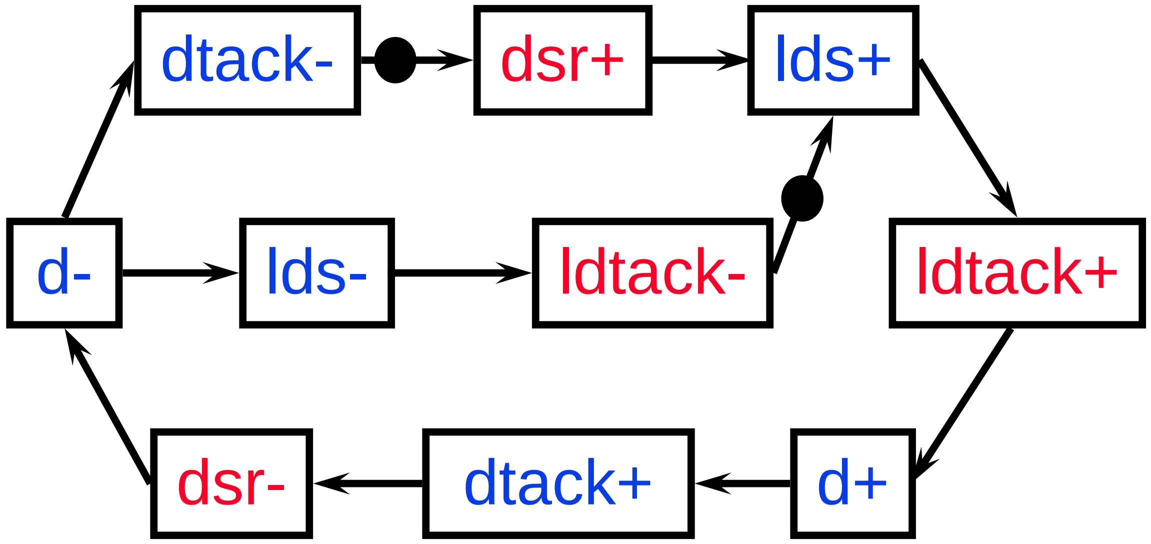
https://workcraft.org/tutorial/synthesis/technology_mapping/start

CSC Conflict Resolution

Victor.Khomenko@ncl.ac.uk

Online tutorial available from workcraft.org

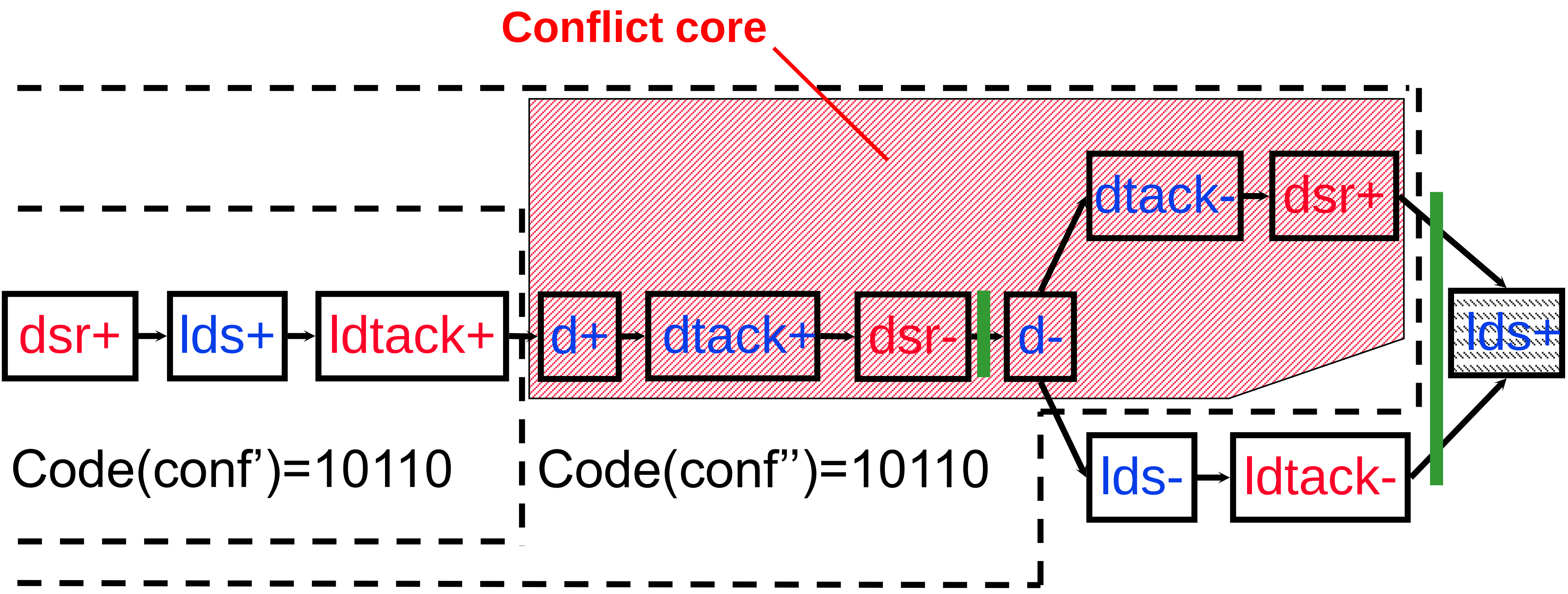
Example: VME Bus Controller



Example: CSC conflict

$$\begin{tikzcd}
00100 \arrow[r, "dtack-"] & 00000 \arrow[r, "dsr+"] & 10000 \arrow[dr, "lds+"] & \\
01100 \arrow[u, "ldtack-"] \arrow[r, "dtack-"] & 01000 \arrow[u, "ldtack-"] \arrow[r, "dsr+"] & 11000 \arrow[u, "ldtack-"] & 10010 \arrow[d, "ldtack+"] \\
01110 \arrow[u, "lds-"] \arrow[r, "dtack-"] & 01010 \arrow[u, "lds-"] \arrow[r, "dsr+"] & \boxed{11010}\ M'' \arrow[u, "lds-"] & M'\ \boxed{11010} \arrow[d, "d+"] \\
 & 01111 \arrow[ul, "d-"] & 11111 \arrow[l, "dsr-"] & 11011 \arrow[l, "dtack+"]
\end{tikzcd}$$

Example: Resolving the conflict



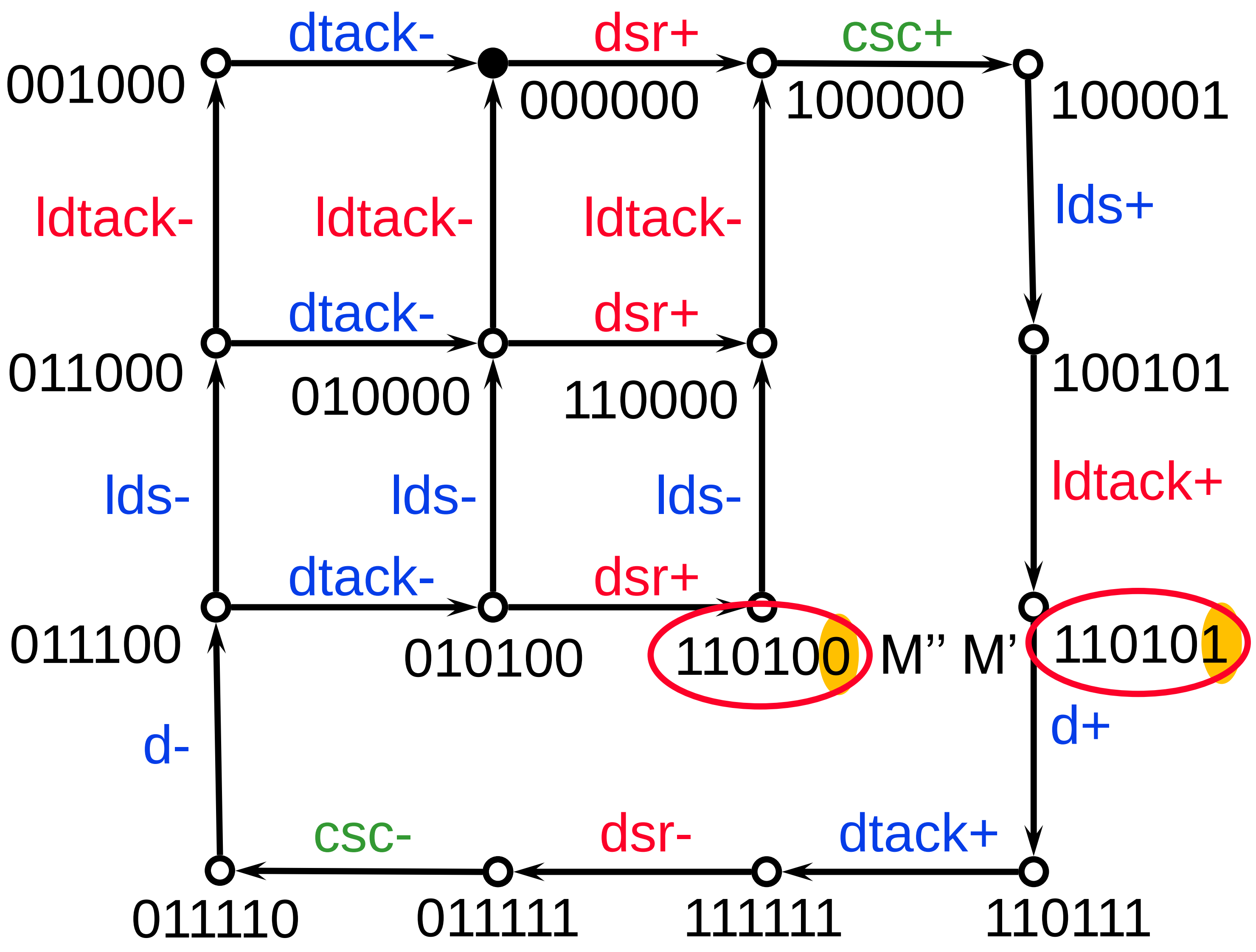
Idea: Insert csc+ into the core and csc- outside the core to break the balance

Note: Cannot delay inputs!

Example: Resolving the conflict

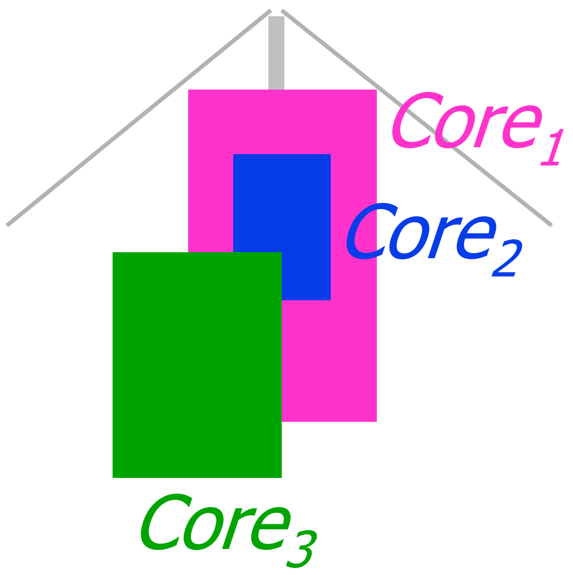
$$\begin{tikzcd}
\boxed{dtack-} \arrow[r, "\bullet"] & \boxed{dsr+} \arrow[r] & \boxed{csc+} \arrow[r] & \boxed{lds+} \arrow[d] \\
\boxed{d-} \arrow[u] \arrow[r] & \boxed{lds-} \arrow[r] & \boxed{ldtack-} \arrow[u, "\bullet"] & \boxed{ldtack+} \arrow[d] \\
\boxed{csc-} \arrow[u] & \boxed{dsr-} \arrow[l] & \boxed{dtack+} \arrow[l] & \boxed{d+} \arrow[l]
\end{tikzcd}$$

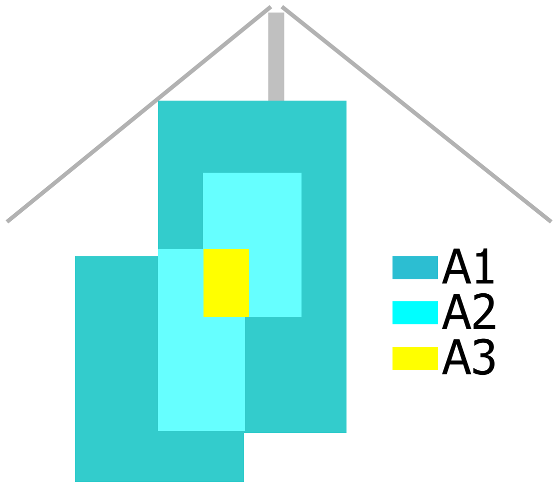
Example: Resolving the conflict

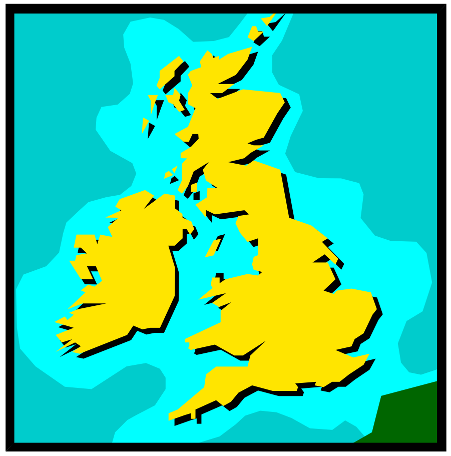


Core map

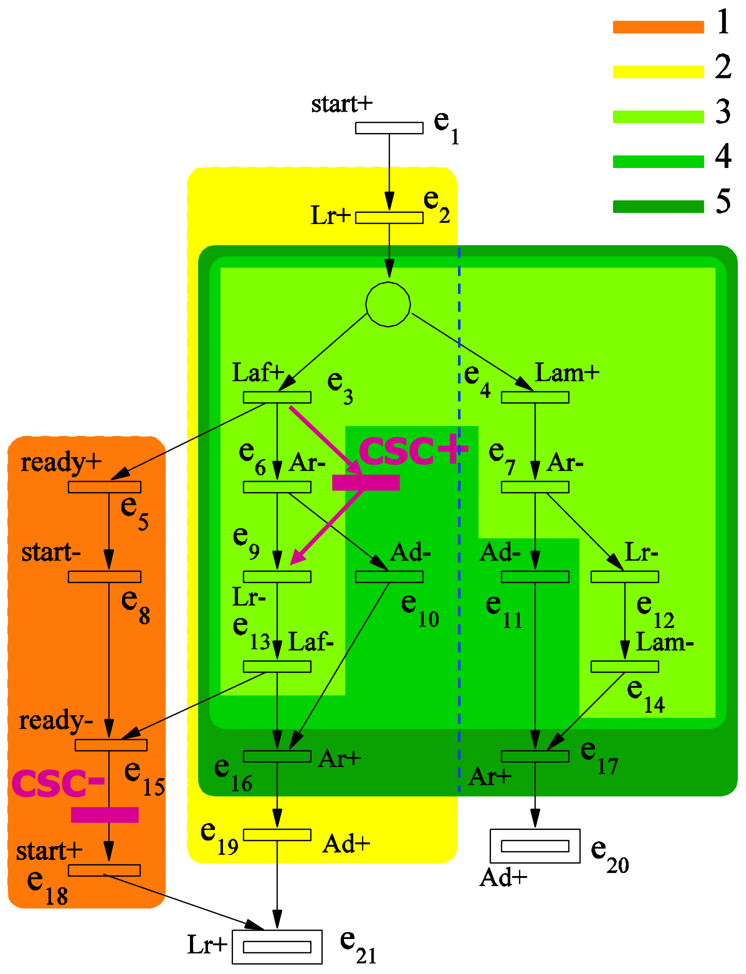
- Cores often overlap
- High-density areas are good candidates for signal insertion
- Analogy with topographic maps







Example: core map



Concurrency reduction

Introduces a new arc in the STG: **a → b**

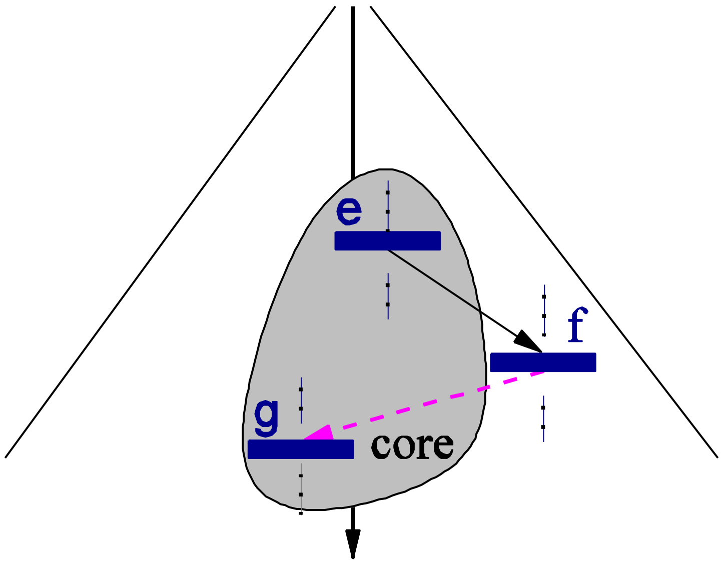
Note: Must not delay inputs, i.e. **b** cannot be an input!

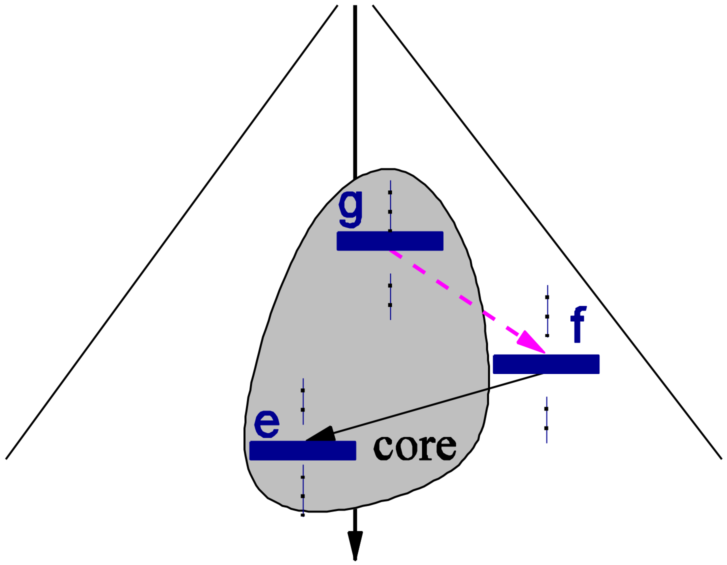
Note: Changes the behaviour, impacts the environment!

Heuristic: Try not to introduce new triggers of **b**, e.g. if there is an arc **a+ → b+** then **a- → b-** is preferred

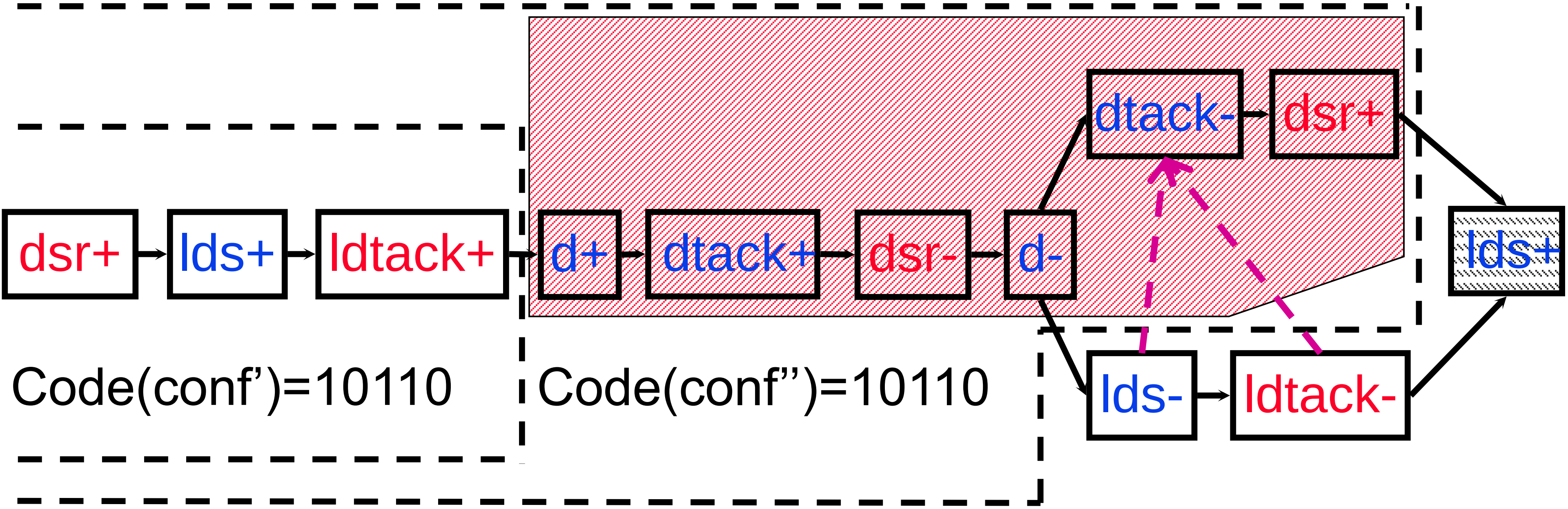
Used for resolving CSC conflicts and circuit simplification

‘Drag’ some events into the core to break the balance:



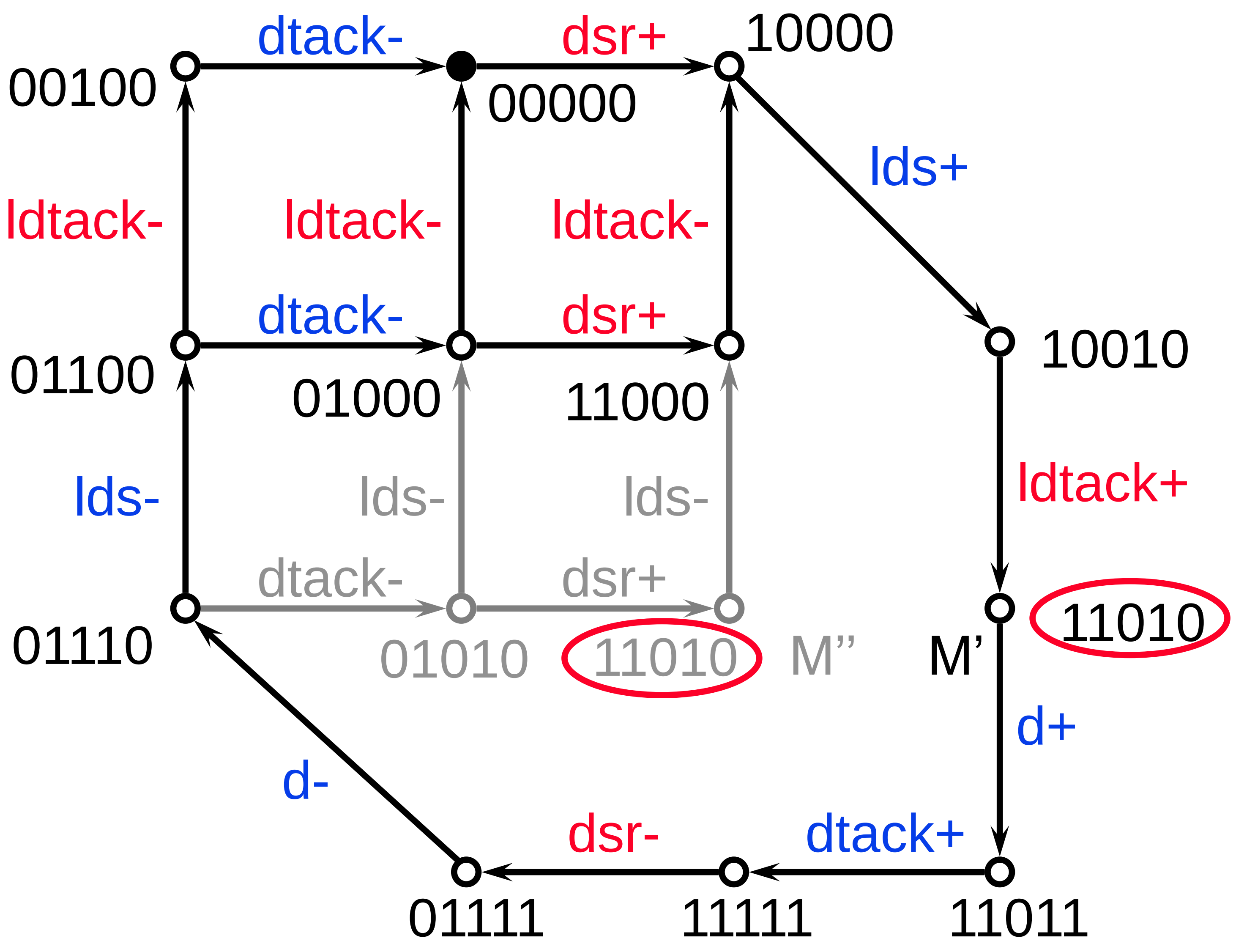


Example: Resolving the conflict



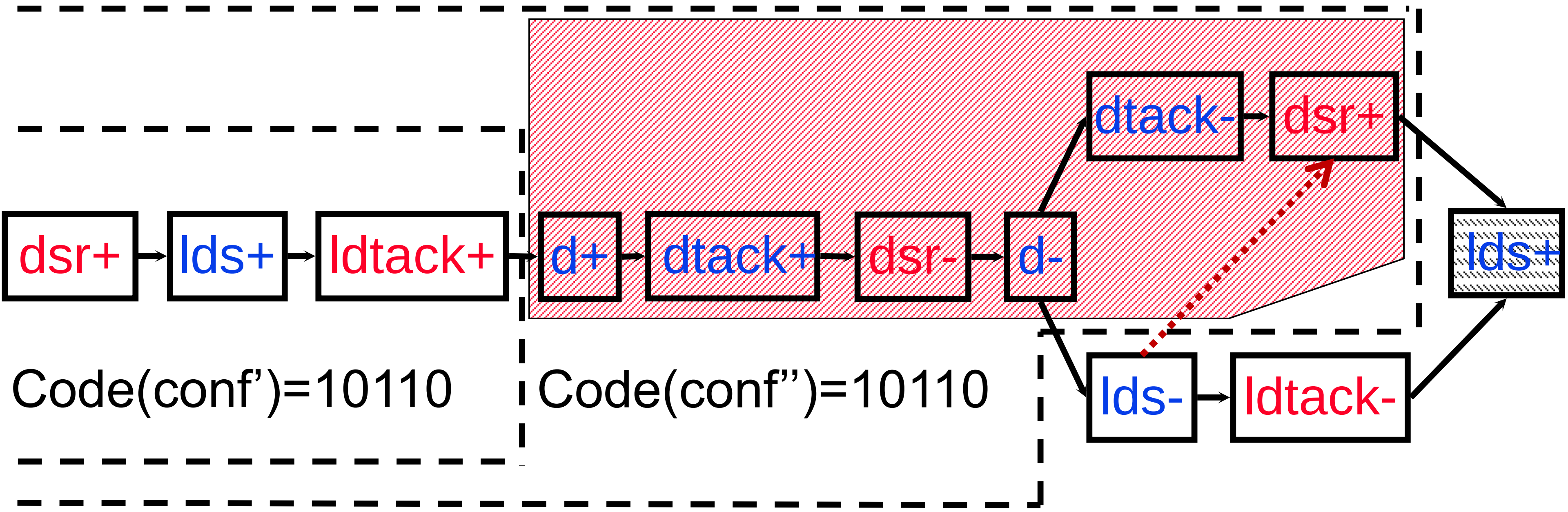
May be problematic!

Example: Resolving the conflict

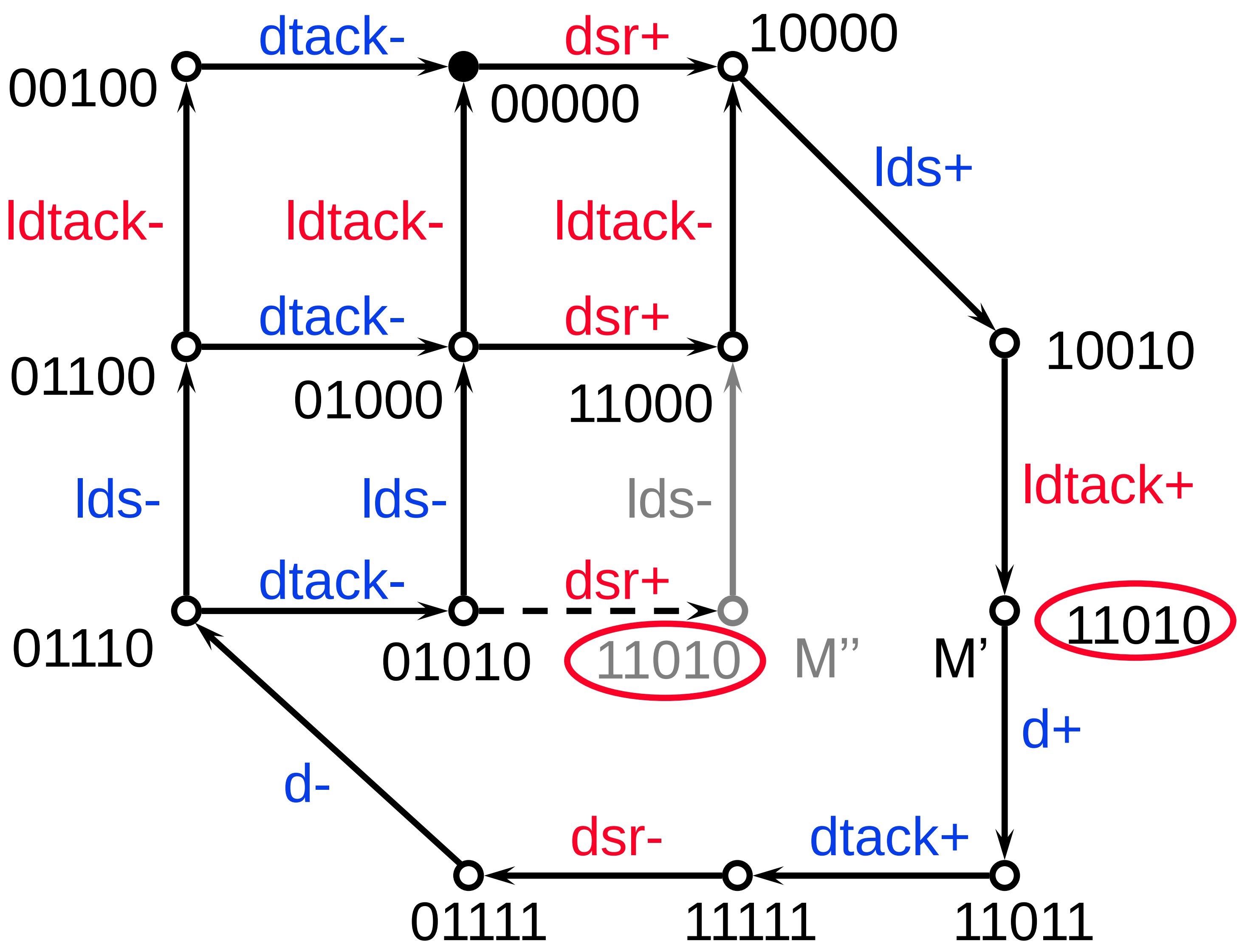


Relative timing assumptions

- "This event will happen faster than that one"
- Break speed-independence, require care & tool support
- Similar to concurrency reductions, but the introduced arcs are special, in particular they don't trigger signals
- Can "delay" inputs



Example: Resolving the conflict



Comparison of the methods

Online tutorial:
workcraft.org

- Signal insertions – paracetamol
  - ☺ behaviour is preserved
  - ☹ inserted signals have to be implemented
- Concurrency reductions – antibiotic
  - ☺ no new signals
  - ☺ reduced state graph and so more don't-cares in minimisation tables
  - ☹ change the behaviour: need to be careful if input → output (even indirectly) – this puts a new assumption on the environment!
  - ☹ can introduce deadlocks: Circuit: a → b & Environment: b → a
- Timing assumptions – surgery
  - ☺ no new signals
  - ☺ reduced state graph and so more don't-cares in minimisation tables
  - ☹ break speed-independence
  - ☹ require deep understanding of theory and the circuit's behaviour
  - ☹ introduce layout constraints, and need care & tool support
  - ☹ fragile due to variability (manufacturing, temperature, voltage, etc.)