Latency Insensitiveness in Adaptive Communication Channels: A Physical Design Perspective

FMGALS'07

Mario R. Casu

www.vlsilab.polito.it

www.polito.it

Before to start…

- Thanks to the FMGALS organizers!
- The research whose results are presented in this talk was a joint work with Prof. Luca Macchiarulo, formerly at Politecnico di Torino and now with the University of Hawaii.

Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

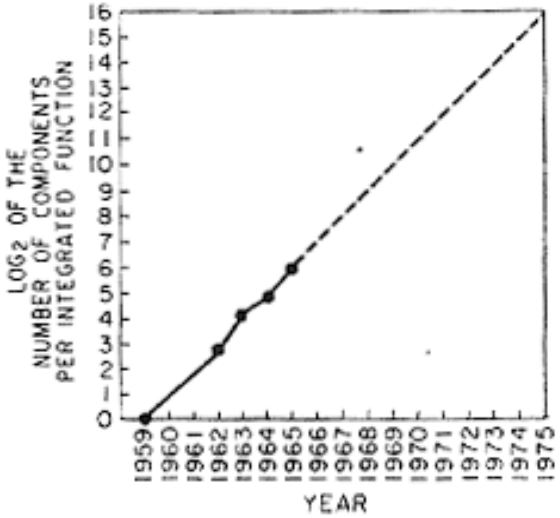
Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

It all started with a prophecy…

Transistors in IC will double every year! [G. Moore, 1965]

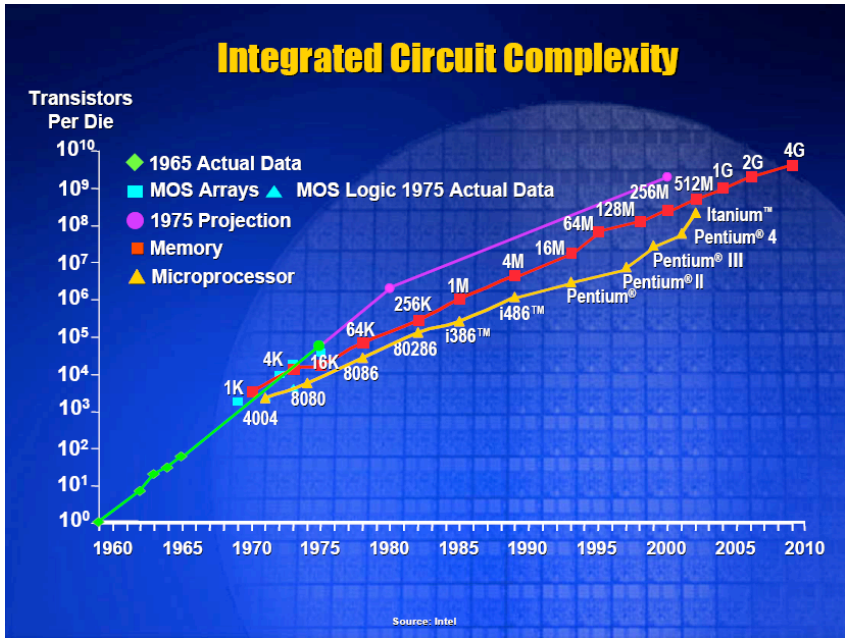
Prophet Isaiah
1509, Sistine Chapel, Michelangelo



’75 prophecy a.k.a. Moore’s Law

Prophet Zechariah
1509, Sistine Chapel, Michelangelo

Transistors in IC will double every 2 years! [G. Moore, 1975]



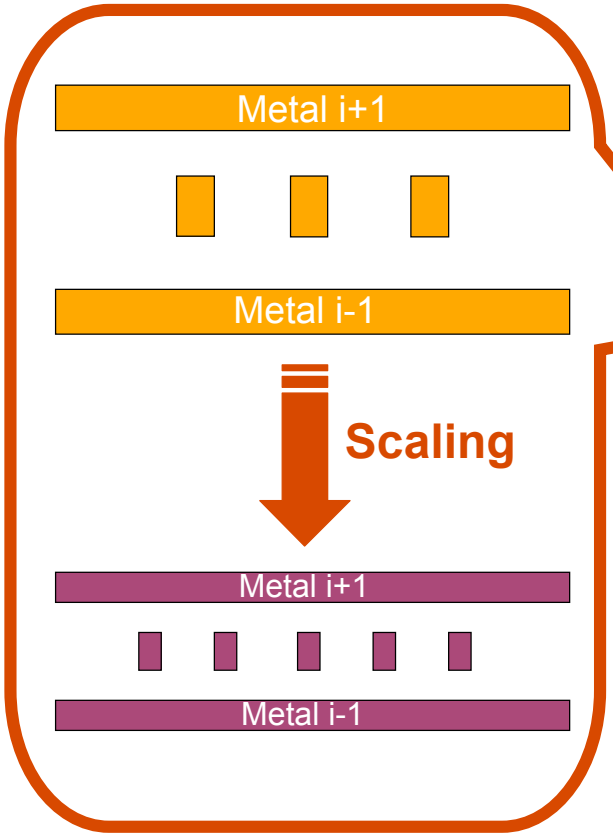
Source: INTEL

Performance implication

- Scaled transistors get faster and faster
  - ~ 17% / year
- Processor performance (f_{ck} x IPC) roughly doubled every 1.5-2 years (so far…)
- It seems we are now at an inflection point due to a combination of issues. Among the others:
  - distributing a low skew centralized clock is a nightmare
  - antinomy between faster transistors and slower wires
  - process parameters uncertainty
  - power management (dynamic + leakage)
  - …

The wise guy

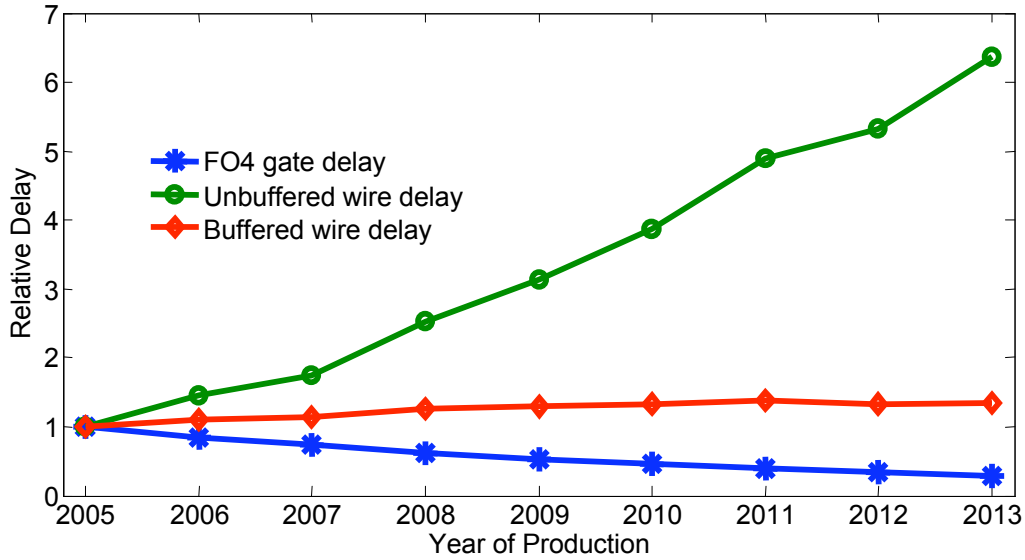
- *"If I make wires narrower and more crammed, resistance grows and capacitance remains constant…"*
- RC delay grows
- Buffered RC delay almost constant



Pithagoras
1509-1511, The School of Athens, Raffaello

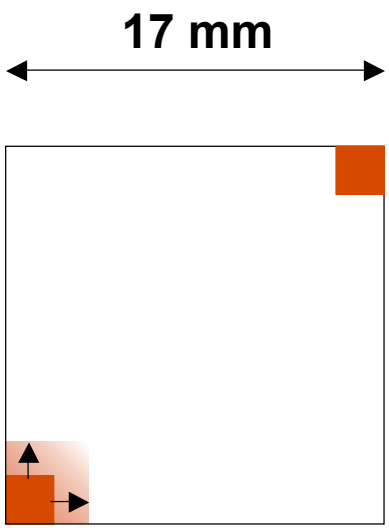
ITRS forecasts

- FO4 gate delays still follow historical -17%/year
- Starting 2007 Tck min flattens at 12 FO4
  - Diminishing returns of deep pipelines
- Bad news for wire delays…



- Constant die area:
  - "[…] power, cost and interconnect cycle latency are strong limiters of die size."
- No global wires in critical paths
  - "[...] buffered global interconnect does not contribute to the minimum clock period since long global interconnects are pipelined"

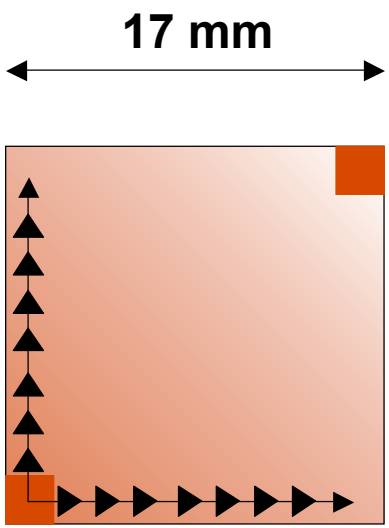
65 nm technology



- unbuffered wire delay
  - 2.6 FO4 $(L/1\ mm)^2$
- L(1 Tck) = 3 mm

- 65 nm shipping today
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 16 ps
- Tck 25 FO4 (min)
- 42 ps delay (2.6 FO4) of 1mm unbuffered global wire (min pitch)

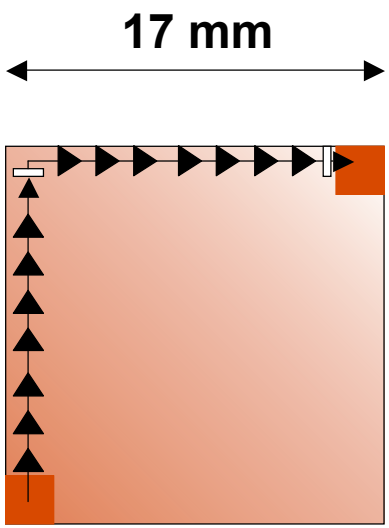
65 nm technology



- Buffered wire delay
  - 1.6 FO4 (L/1 mm)
- L(1 Tck) = 15 mm
  - ~ 24 repeaters

- 65 nm shipping today
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 16 ps
- Tck 25 FO4 (min)
- 42 ps delay (2.6 FO4) of 1mm unbuffered global wire (min pitch)
- 26 ps/mm delay of buffered global wire

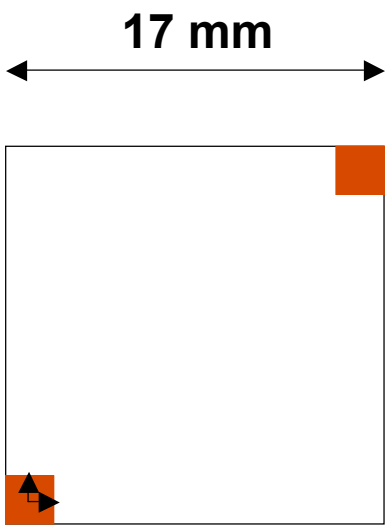
65 nm technology



- Buffered wire delay
  - 1.6 FO4 (L/1 mm)
- L(1 Tck) = 15 mm
  - Corner to corner: 2 ck latency

- 65 nm shipping today
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 16 ps
- Tck 25 FO4 (min)
- 42 ps delay (2.6 FO4) of 1mm unbuffered global wire (min pitch)
- 26 ps/mm delay of buffered global wire

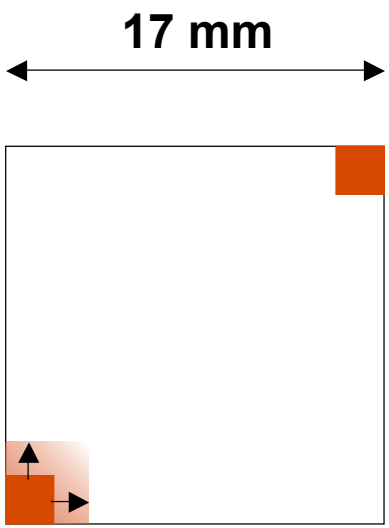
Near term roadmap



- Unbuffered wire delay
  - 19 FO4 $(L/1\ mm)^2$
- L(1 Tck) ~ 0.8 mm

- Year of production: 2007
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 9 ps
- Tck 12 FO4 (min)
- 170 ps delay (19 FO4) of 1mm unbuffered global wire (min pitch)

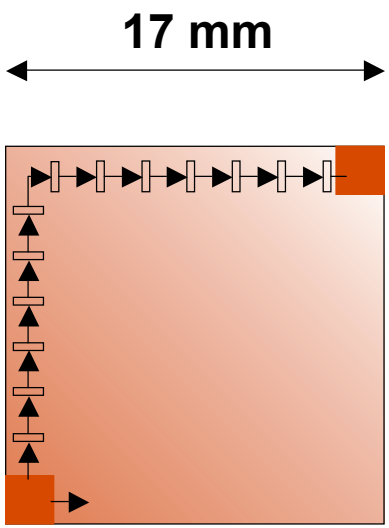
Near term roadmap



- Buffered wire delay
  - 4.5 FO4 (L/1 mm)
- L(1 Tck) ~ 3 mm

- Year of production: 2007
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 9 ps
- Tck 12 FO4 (min)
- 170 ps delay (19 FO4) of 1mm unbuffered global wire (min pitch)
- 40 ps/mm delay of buffered global wire

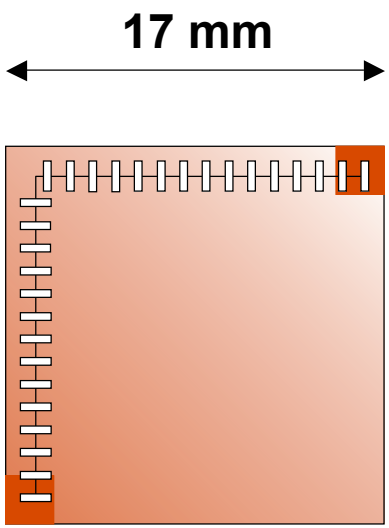
Near term roadmap



- Buffered wire delay
  - 4.5 FO4 (L/1 mm)
- L(1 Tck) ~ 3 mm
  - corner to corner: 13 ck latency

- Year of production: 2007
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 9 ps
- Tck 12 FO4 (min)
- 170 ps delay (19 FO4) of 1mm unbuffered global wire (min pitch)
- 40 ps/mm delay of buffered global wire

End of near term roadmap



- Buffered wire delay
  - 13 FO4 (L/1 mm)
- L(1 Tck) ~ 1 mm
  - corner to corner: **34 ck latency**

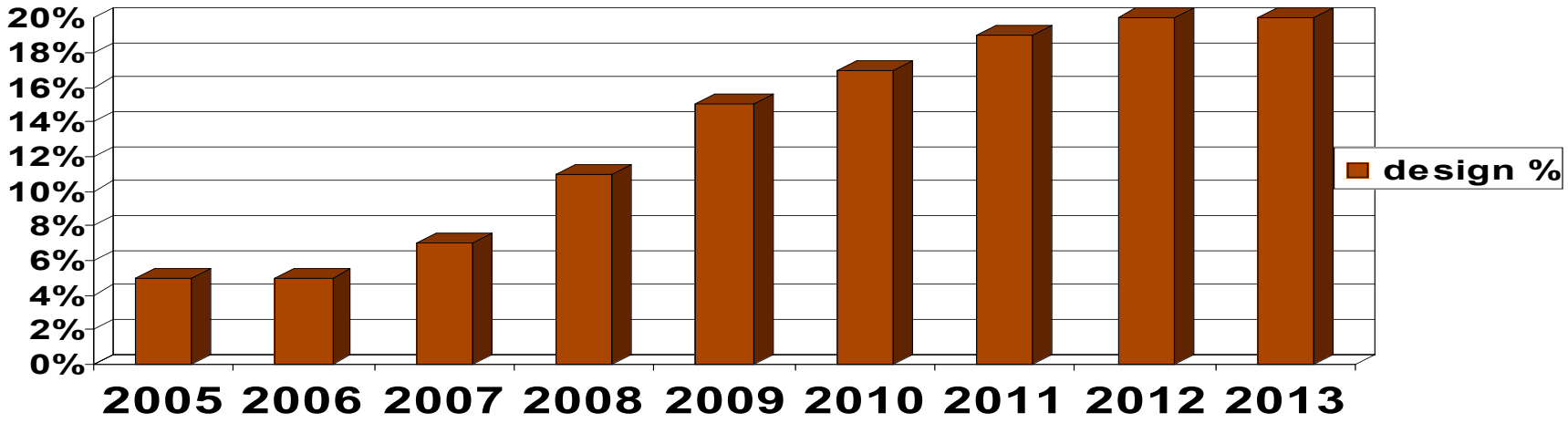
- Year of production: 2013
- Max chip size ~ 300 mm^2
- High performance process
  - FO4 delay 3.5 ps
- Tck 12 FO4 (min)
- 600 ps delay (170 FO4) of 1mm unbuffered global wire (min pitch)
- 45 ps/mm delay of buffered global wire

Interconnect summary

- Wire delay with repeaters: $\delta_{FO4} \cdot L$
- Clock period: $T_{ck} = n_{FO4}$
- Critical length: $L_{crit} = n_{FO4} / \delta_{FO4}$
  - L_{crit} is getting shorter and shorter
- T_{ck} can be expressed in terms of critical length:
  - $T_{ck} = n_{FO4} = \delta_{FO4} \cdot L_{crit}$
- For a given technology, we can normalize the proportionality coefficient:
  - $T_{ck} = L_{crit}$, $f_{ck} = 1 / L_{crit}$

GALS to the rescue

- Again from ITRS:
  - "One of the main challenges of modern IC is to distribute a centralized clock signal throughout the chip with an acceptable low skew."
- Asynchronous global signaling
  - % of a design driven by handshake clocking



Outline

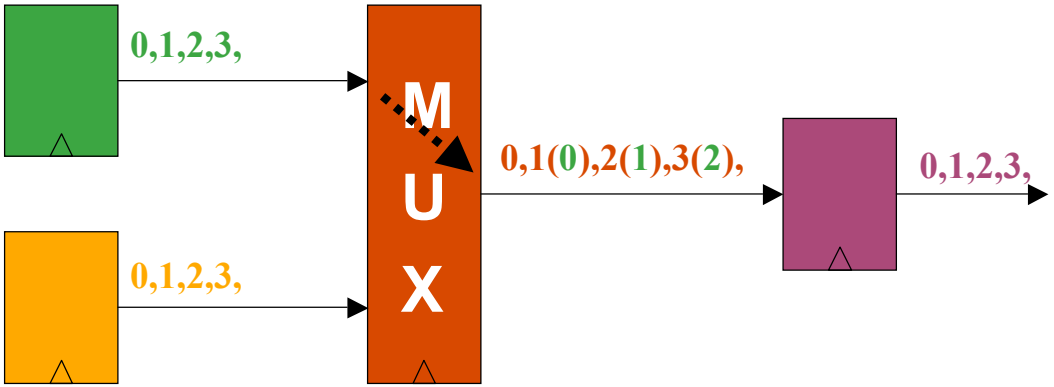
- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

Latency Insensitive Design

- Synchronous computational logic
  - No leap to fully asynchronous approach in mainstream design
- (A)synchronous global communication through “multi-cycle” channels
  - syn/meso/plesio/asyn-chronous
- Basic idea of Latency Insensitive Design
  - Gate/trigger local clock when data are absent/present
  - Use wire pipelines to sustain data rate (no global wires in critical paths) adding relay stations
  - Use a latency insensitive protocol (LIP) to enforce handshake (e.g. valid/stop)
- Two variants
  - Static LIP vs. Adaptive LIP

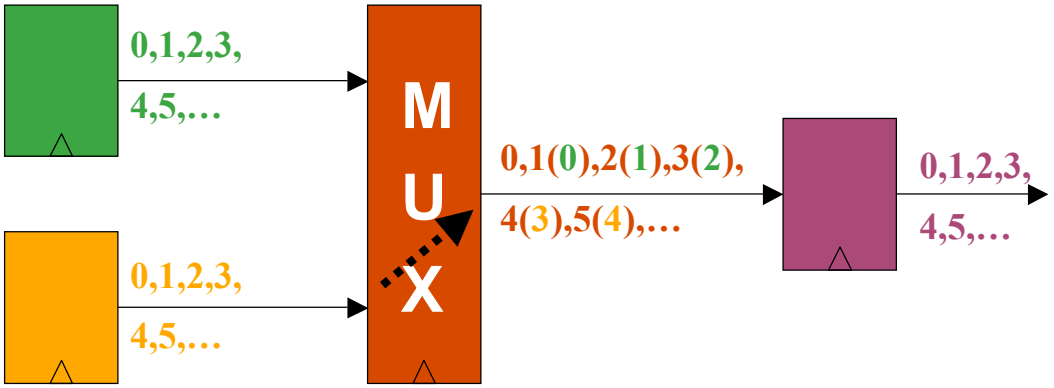
Static LIPs

- L. Carloni et al. [Carloni99]
- Original idea was fully synchronous
- Example: system prior to LIP modification



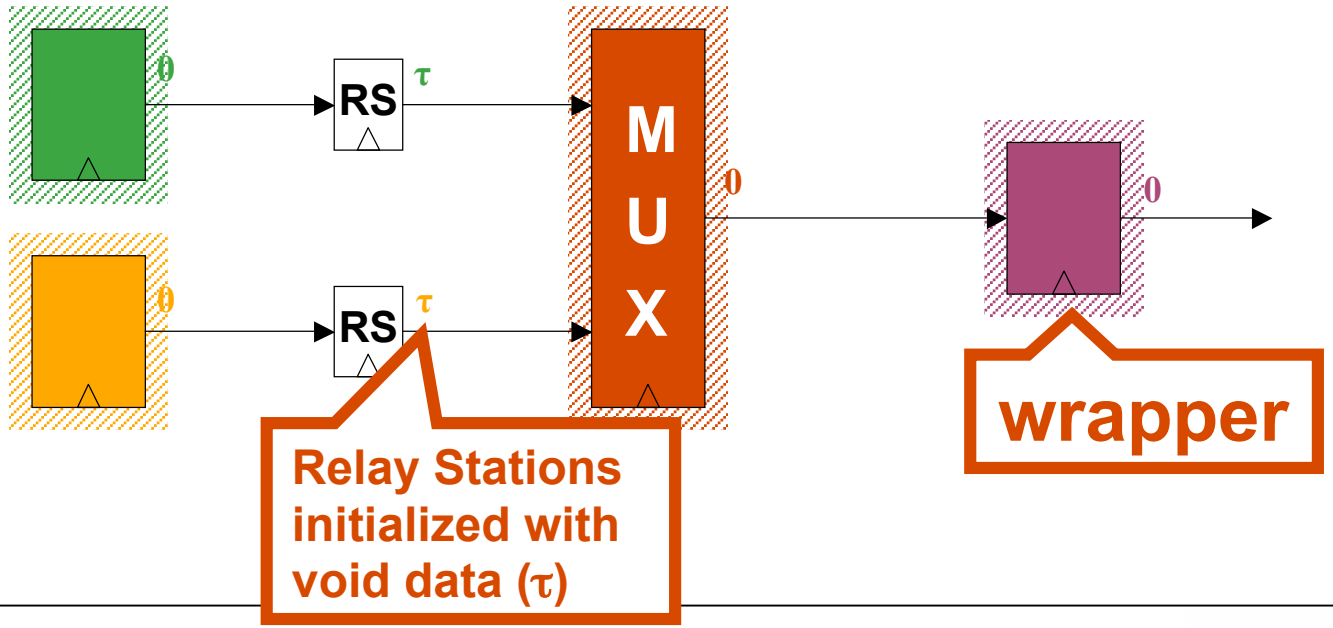
Static LIPs

- L. Carloni et al. [Carloni99]
- Original idea was fully synchronous
- Example: system prior to LIP modification



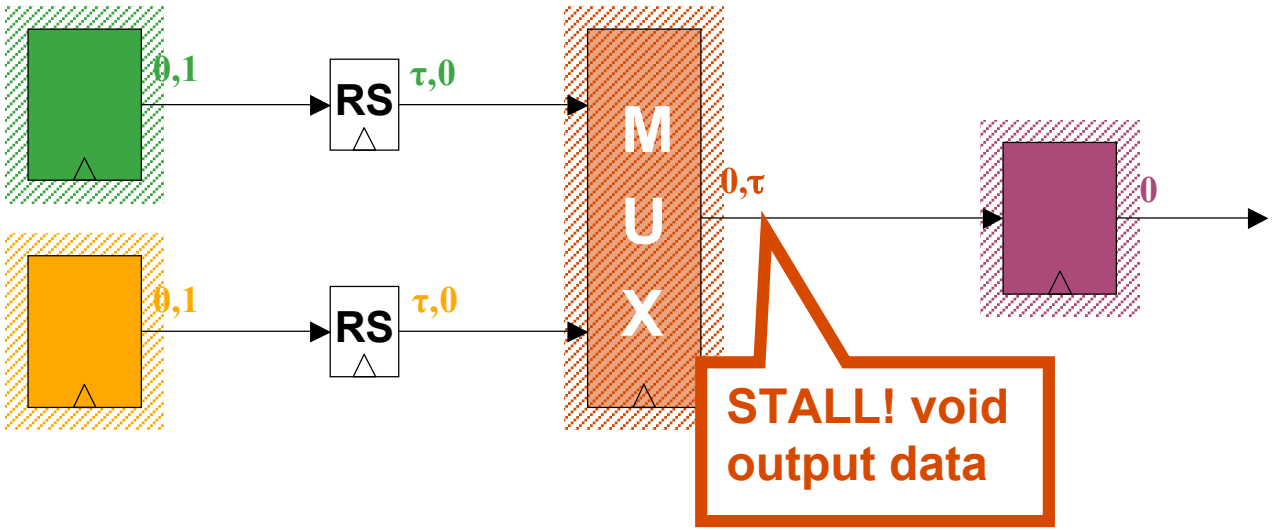
Static LIPs

- L. Carloni et al. [Carloni99]
- Original idea was fully synchronous
- Example: system *after static LIP modification*



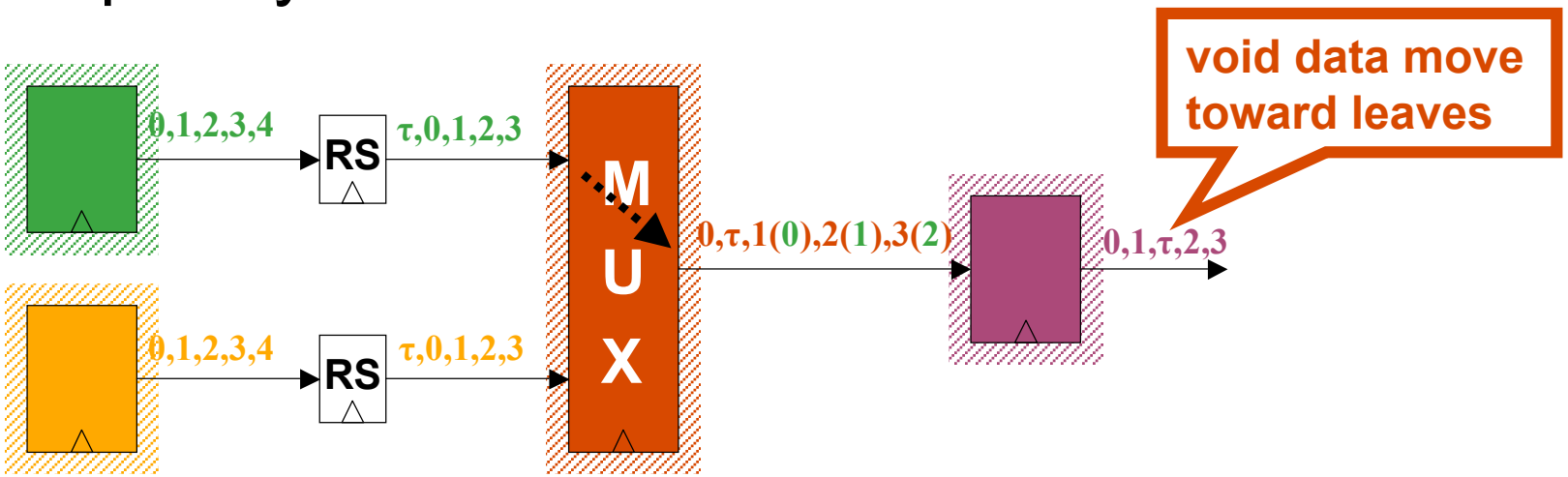
Static LIPs

- L. Carloni et al. [Carloni99]
- Original idea was fully synchronous
- Example: system *after static LIP modification*



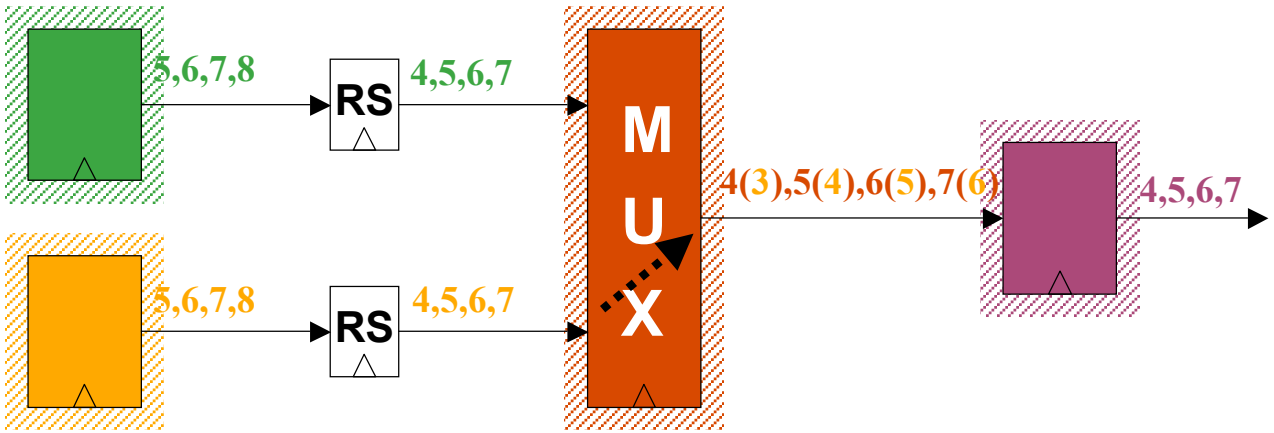
Static LIPs

- L. Carloni et al. [Carloni99]
- Original idea was fully synchronous
- Example: system *after static LIP modification*



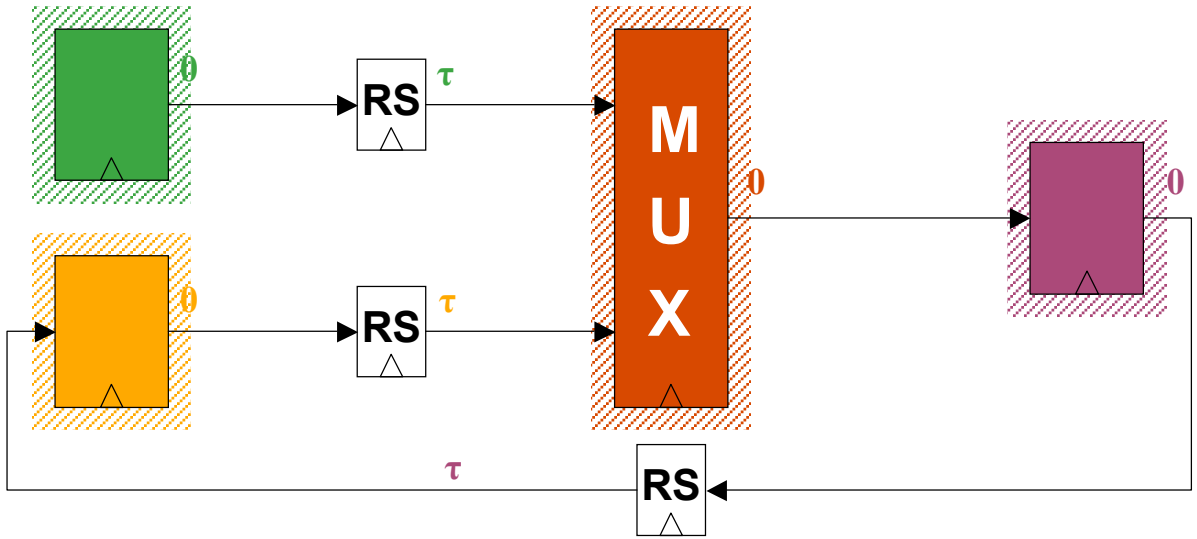
Static LIPs

- Feed-forward topology
- Void data *removed* after a transient
- *Throughput*: 1 data/1 ck (synch hypothesis)



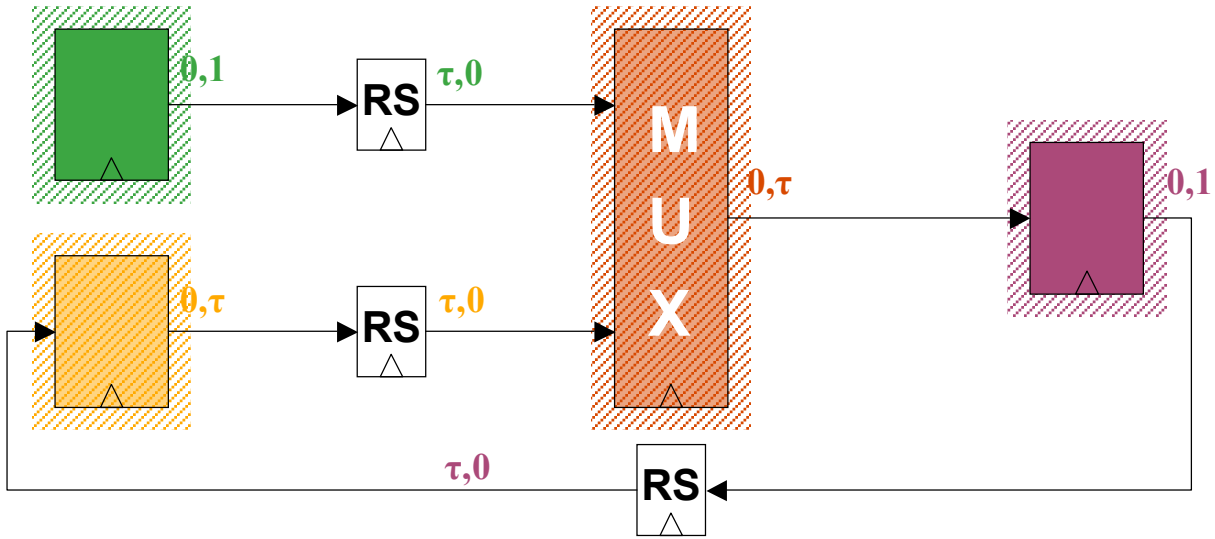
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*



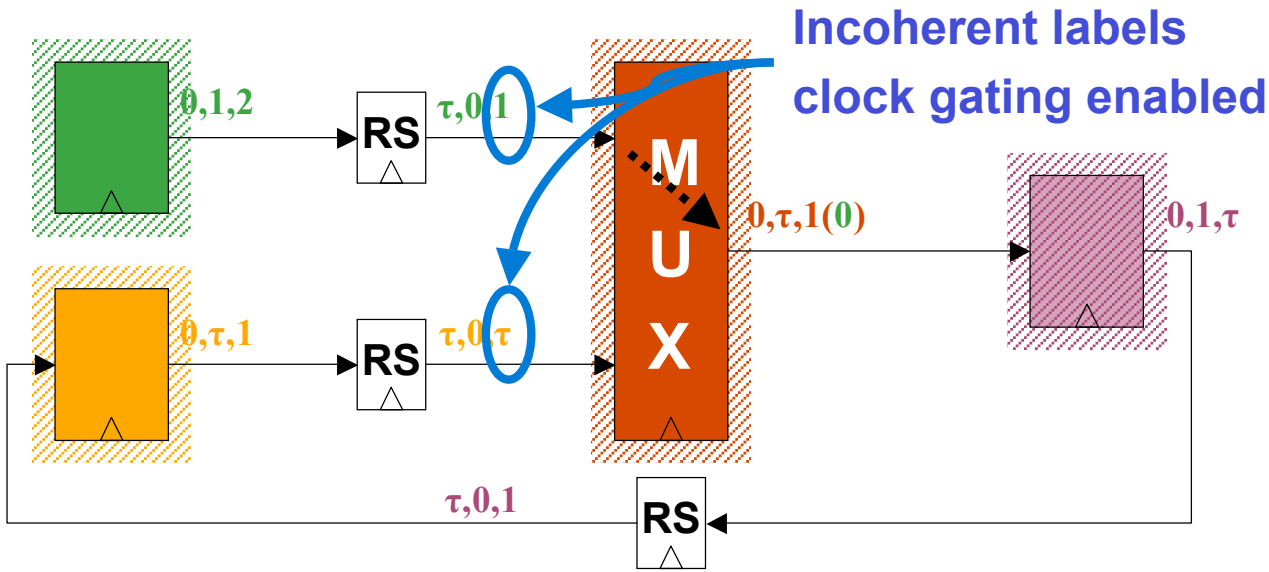
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*



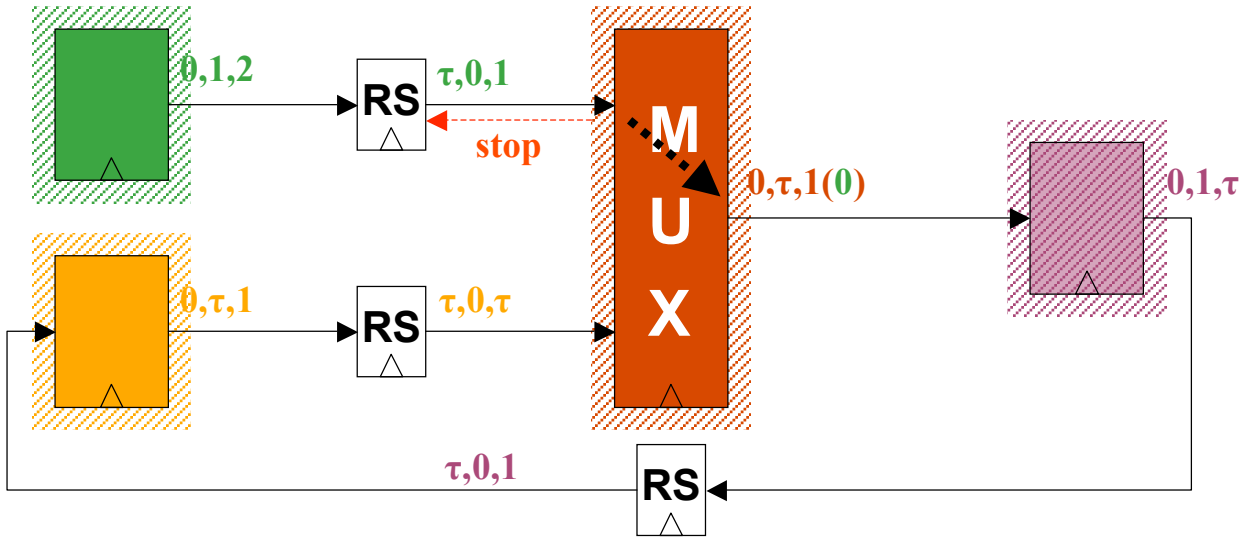
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*
- Back-pressure exerted by wrappers on *fast* links



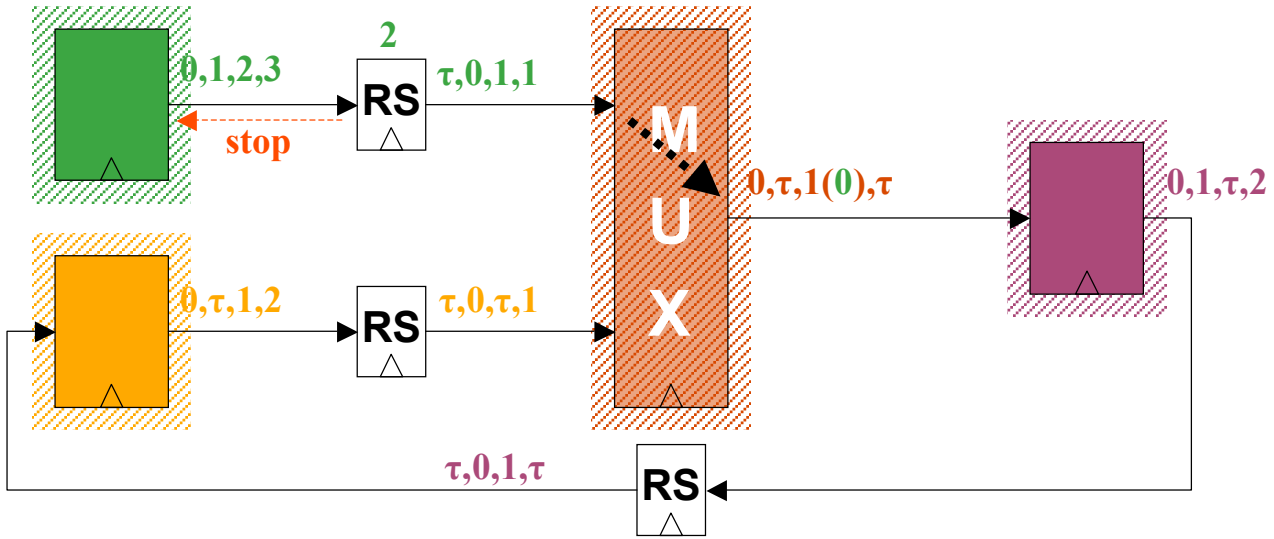
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*
- Back-pressure exerted by wrappers on *fast* links



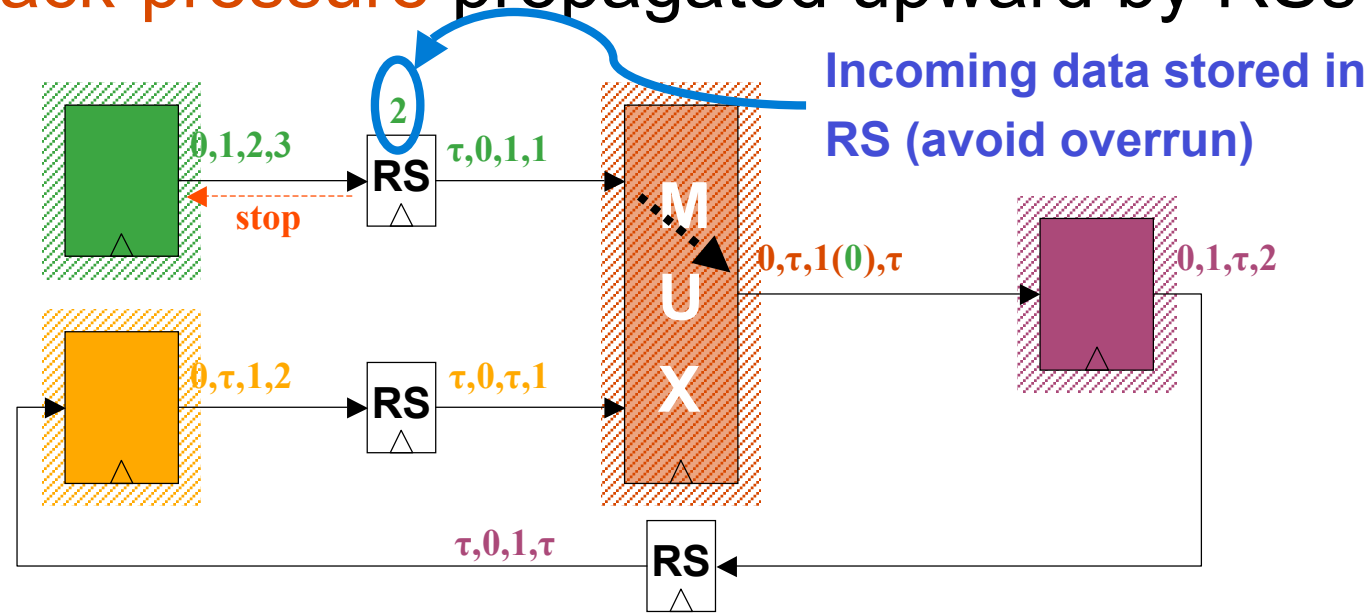
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*
- Back-pressure propagated upward by RSs



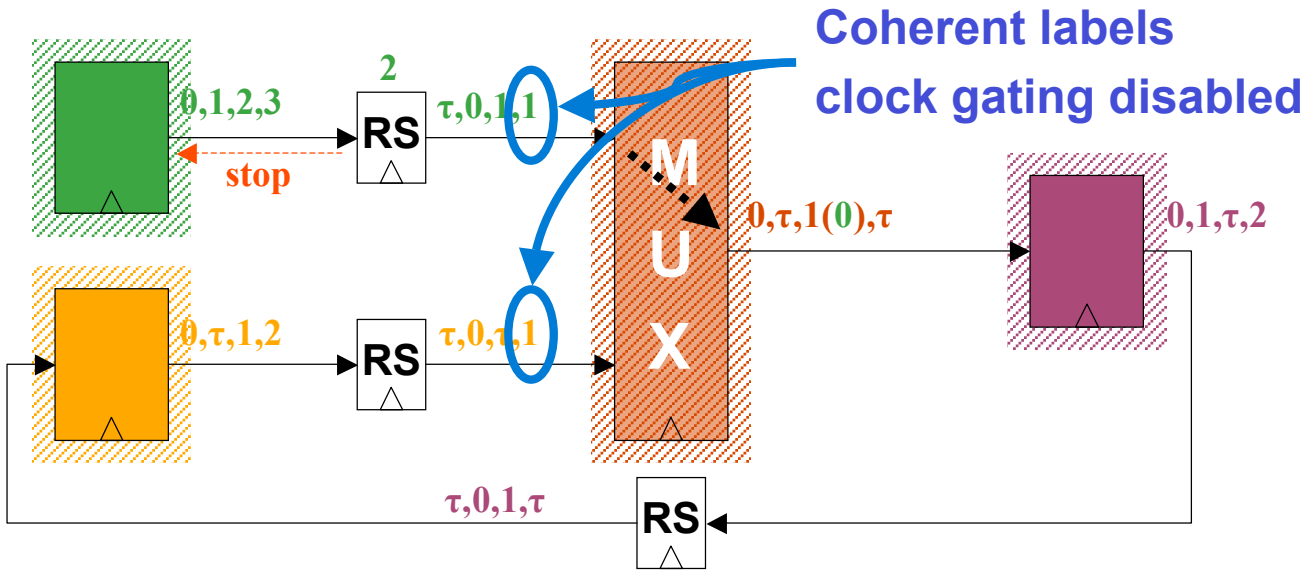
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*
- Back-pressure propagated upward by RSs



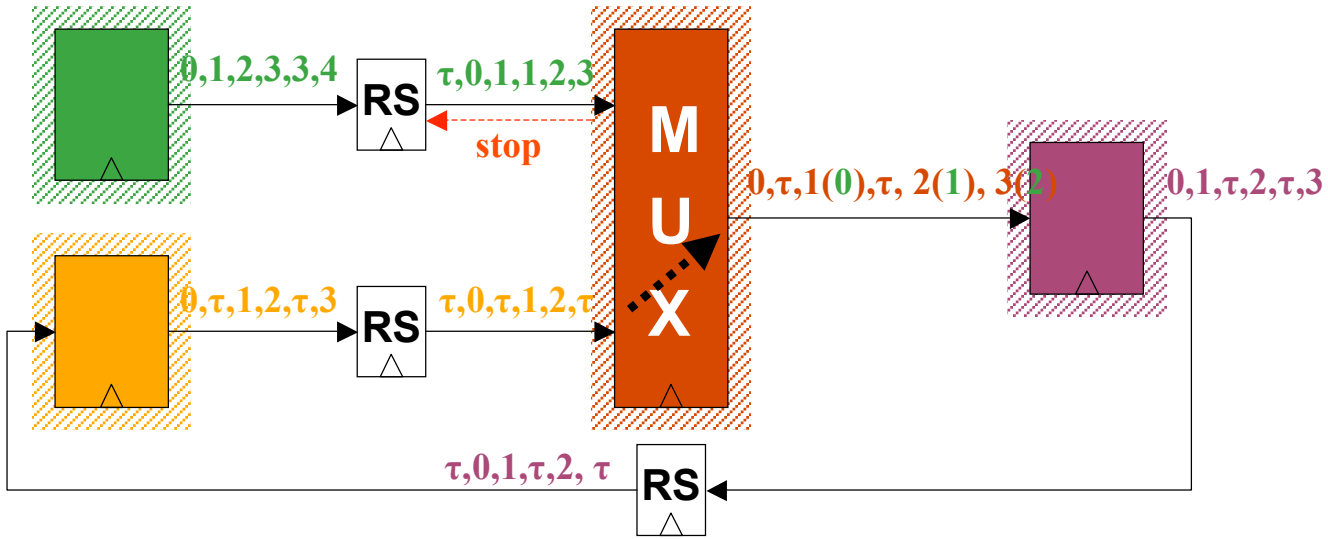
Loops in static LIPs

- Feed-back (loop) topology
- Void data *circulate*
- Back-pressure propagated upward by RSs



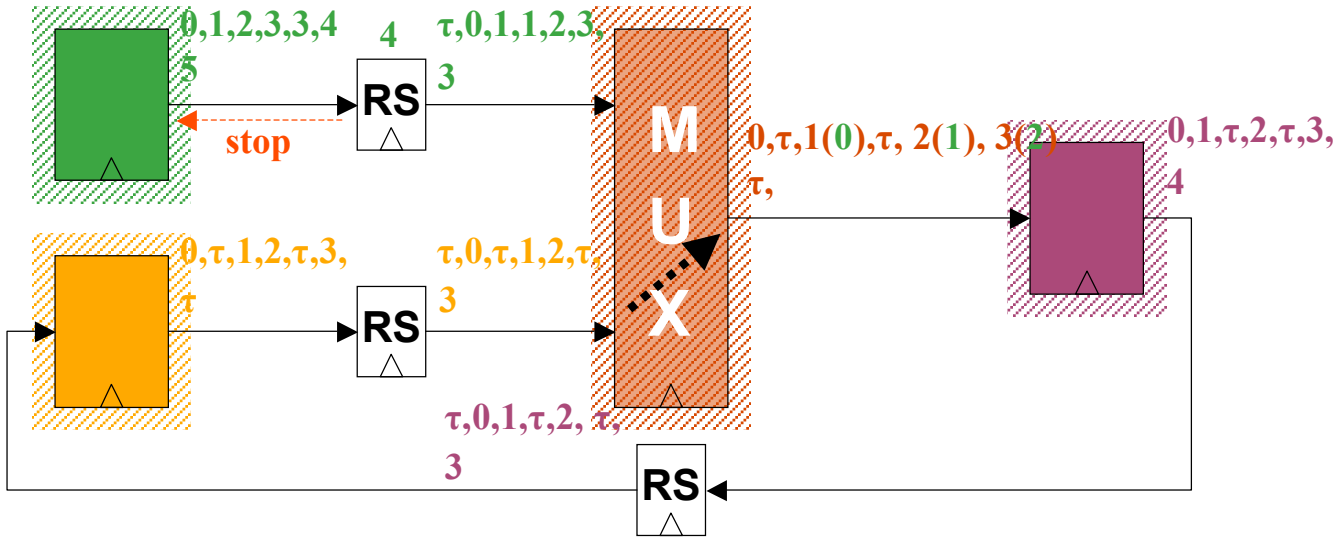
Loops in static LIPs

- Moving two clock ticks forward…
- Yet another stall for the mux
- Back-pressure again on fast link



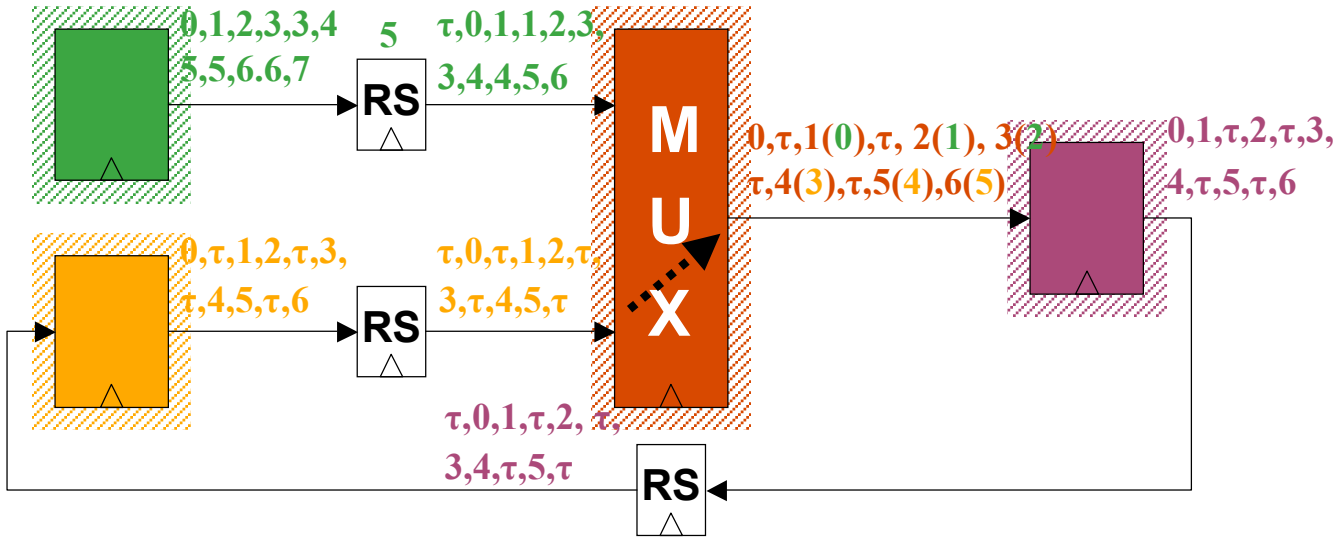
Loops in static LIPs

- Another clock tick forward…
- Back-pressure propagated upward
- Valid and void data alternate periodically



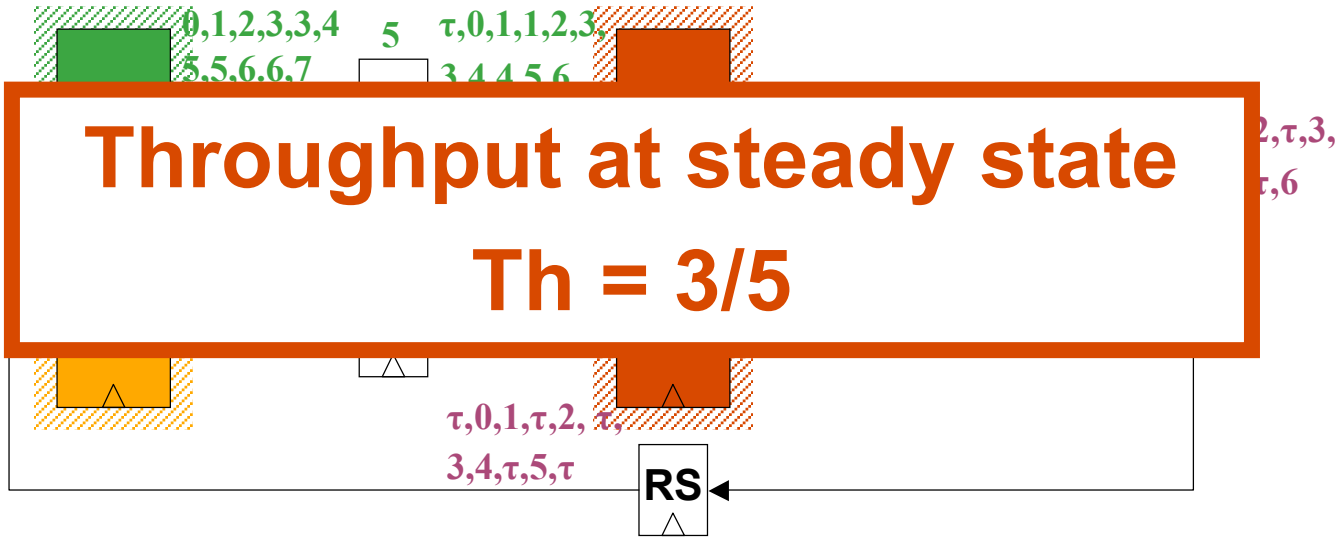
Loops in static LIPs

- Looking at the valid/void sequence
- |v,v,τ,v,τ| modulus repeats indefinitely
- 3 valid data out of 5 “tokens”



Loops in static LIPs

- Looking at the valid/void sequence
- |v,v,τ,v,τ| modulus repeats indefinitely
- 3 valid data out of 5 “tokens”

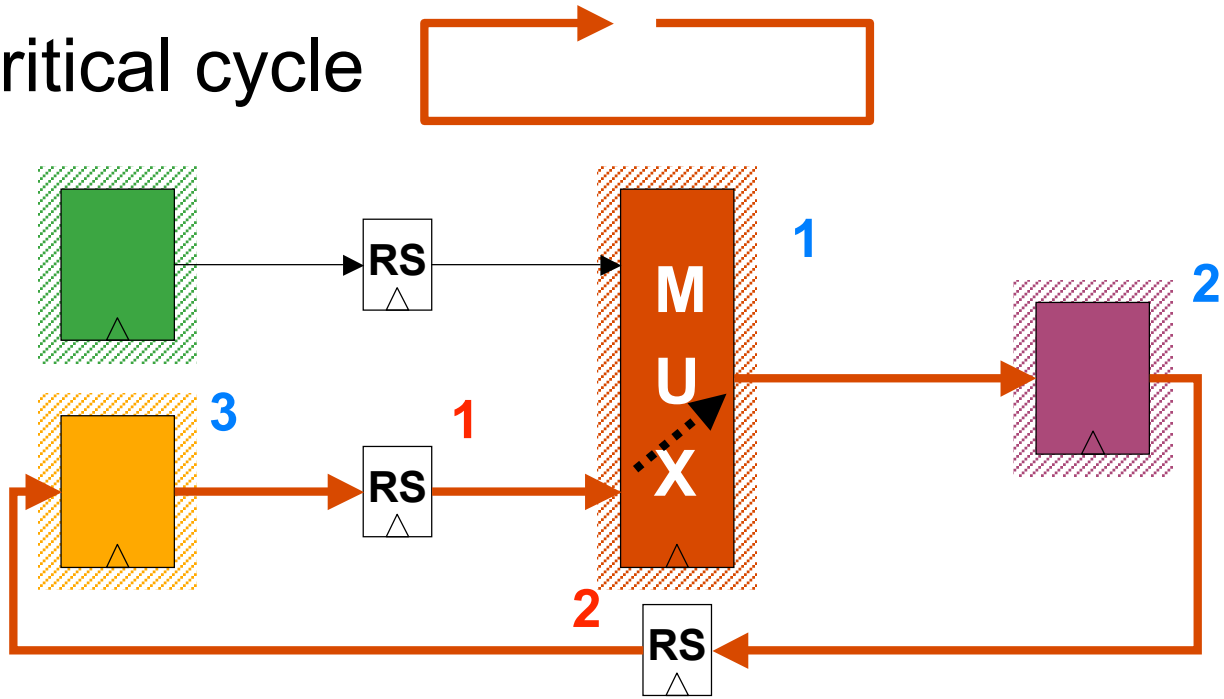


Throughput at steady state

Th = 3/5

Loops in static LIPs

- Cycle time [Carloni00]: $\ddot{e}(C) = \frac{w(C) + |C|}{|C|} = \frac{1}{Th(C)}$
- Critical cycle



Throughput at steady state

Th = 3/5=3/(3+2)

Static LIPs: PROS/CONS

- PROS
  - Complete orthogonalization of computation and communication
  - Simple wrapper
  - Performance known upfront from netlist only: no need to know the exact behavior of the system
  - Simpler protocol allowed [DAC04]
  - Can be adapted to GALS systems (e.g. modifying valid/stop protocol to account for FIFO empty/full semantics and using mixed-clock FIFOs [Nowick01])

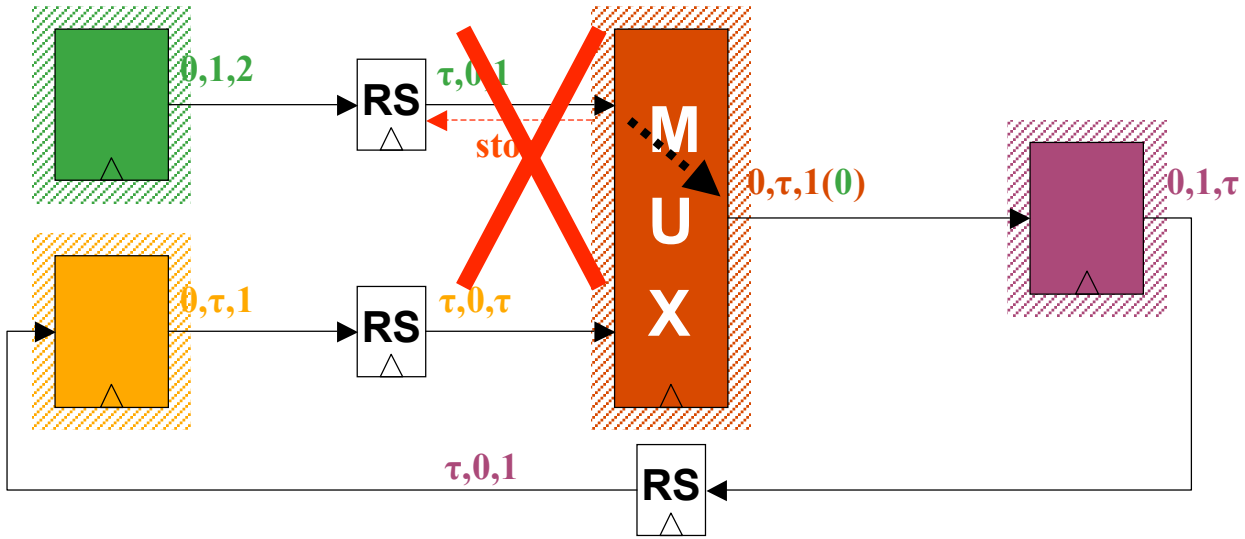
- CONS
  - Area overhead (wrappers & RS)
  - Routing overhead (extra signals)
  - No guarantee of better data rate (DR) than clock frequency slow-down due to wire delay:
  - $DR_{no\ LIP} = f_{no\ LIP} \cdot 1$
  - $DR_{LIP} = f_{LIP} \cdot Th$ where Th is the throughput of the *worst loop*
  - Th always ≤ 1

Generalized LIPs [Singh03]

- Static LIPs:
  - unavailability of input forces stall
- Basic idea of Generalized LIPs (Singh and Theobald, FMGALS'03):
  - *Stalls can be avoided if unavailable inputs aren't needed for next computation* (see previous MUX)
- Throughput is no more statically determined by the worst loop. Throughput behavior is *adaptive*
  - Need for synchronization? Overrun avoidance?
- In the following "Adaptive LIPs"

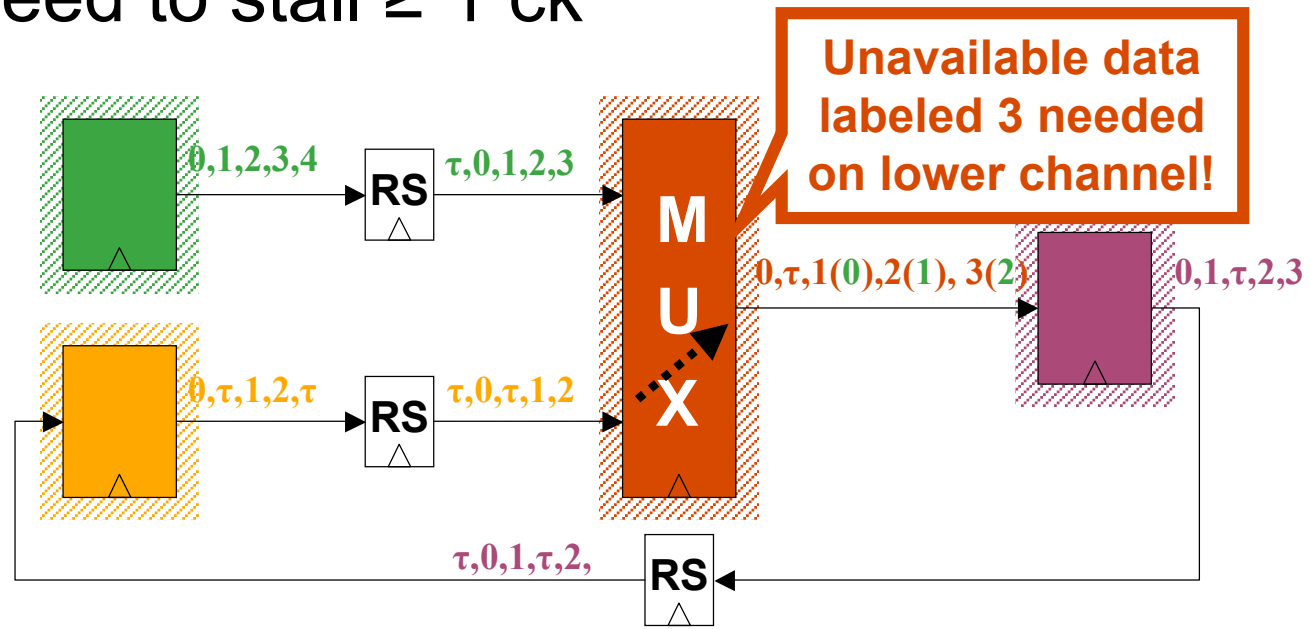
Adaptive LIPs

- Previous example: void data ignored on lower input because not needed for next computation
- Back-pressure and stall avoided



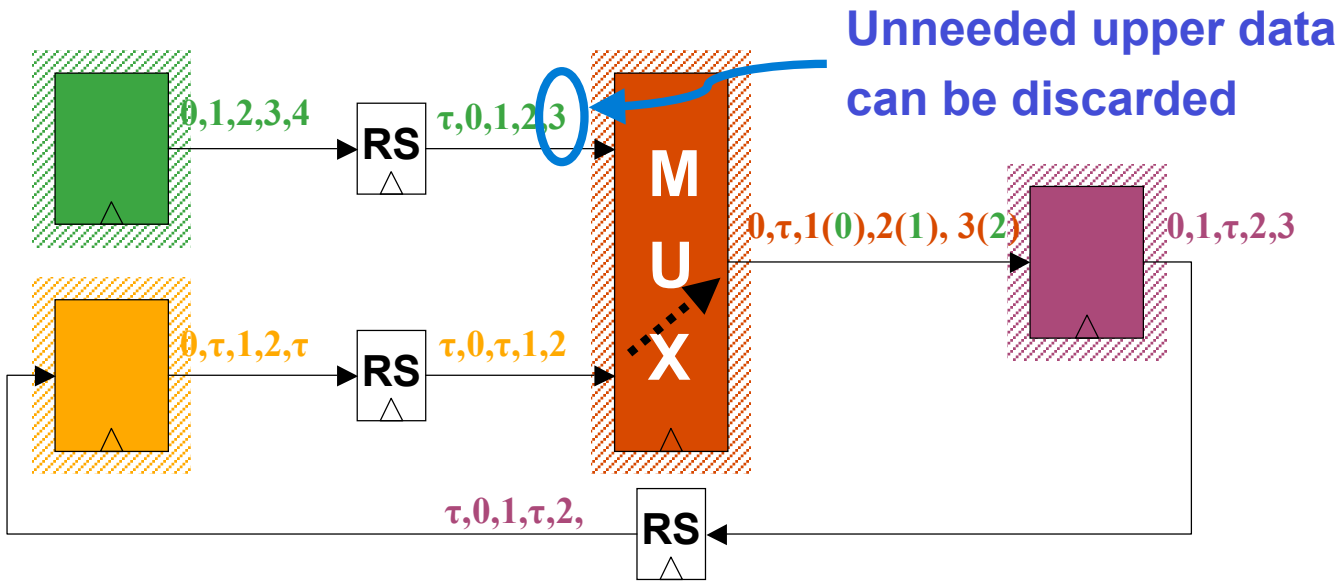
Adaptive LIPs

- Moving 2 ticks ahead. Lower input now needed…
- Problem: *old* data (label 2) w.r.t. *local time* (3)
- Need to stall ≥ 1 ck



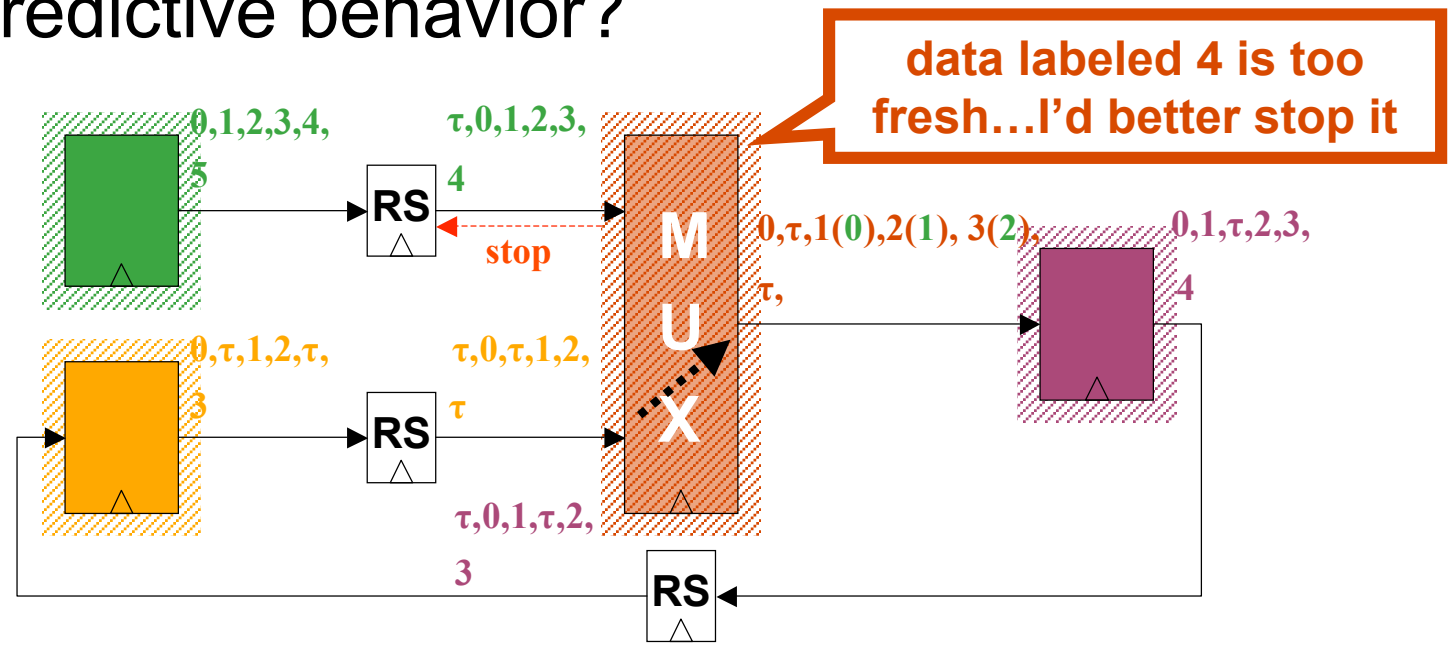
Adaptive LIPs

- Moving 2 ticks ahead. Lower input now needed…
- Problem: *old* data (label 2) w.r.t. *local time* (3)
- Need to stall ≥ 1 ck



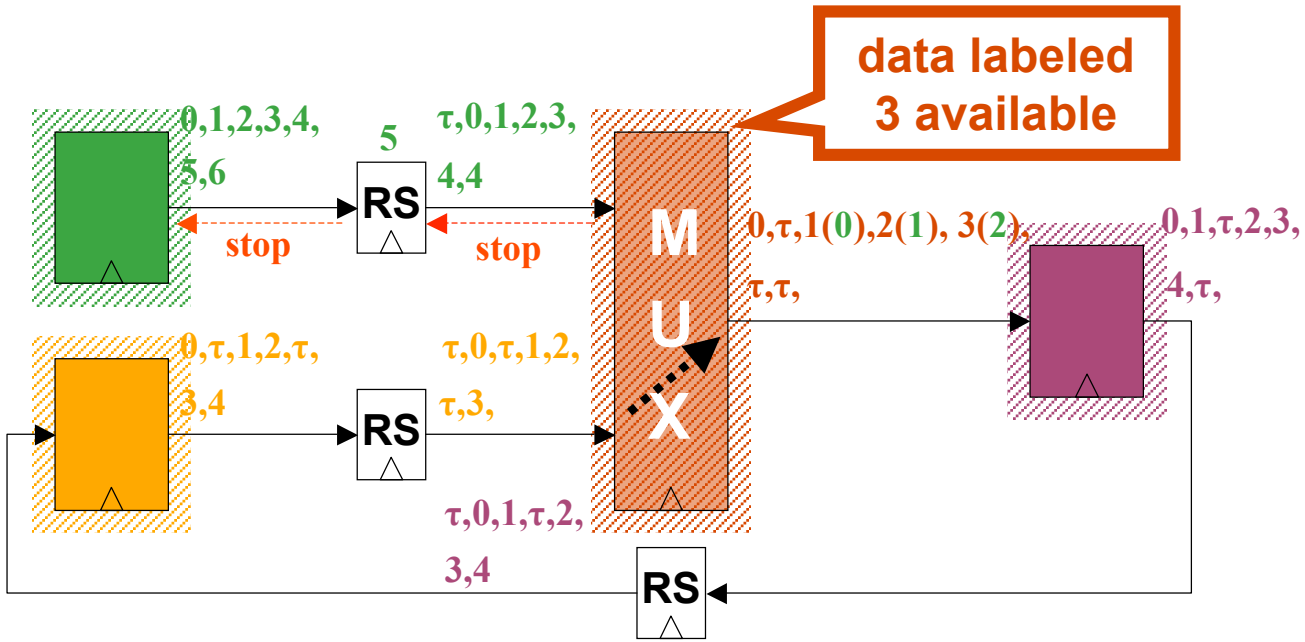
Adaptive LIPs

- Upper input at risk of overrun. Stop or not?
- Avoid back-pressure if you have a *crystal ball…*
- Predictive behavior?



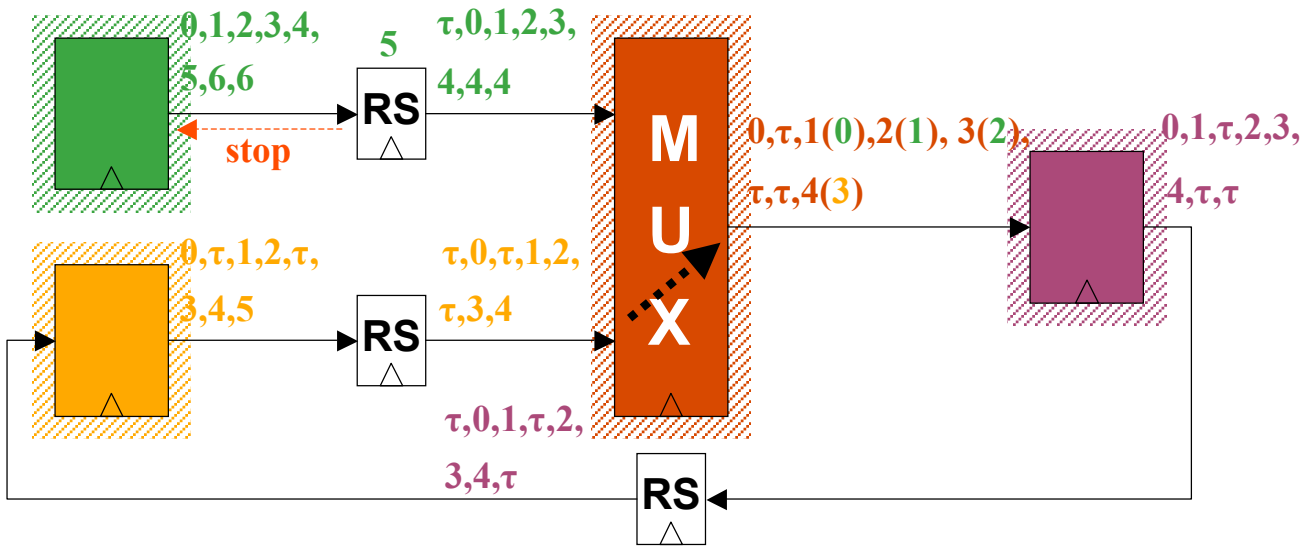
Adaptive LIPs

- Two cycles stall (ττ) due to late data number 3
- Data 4 on upper input still stopped



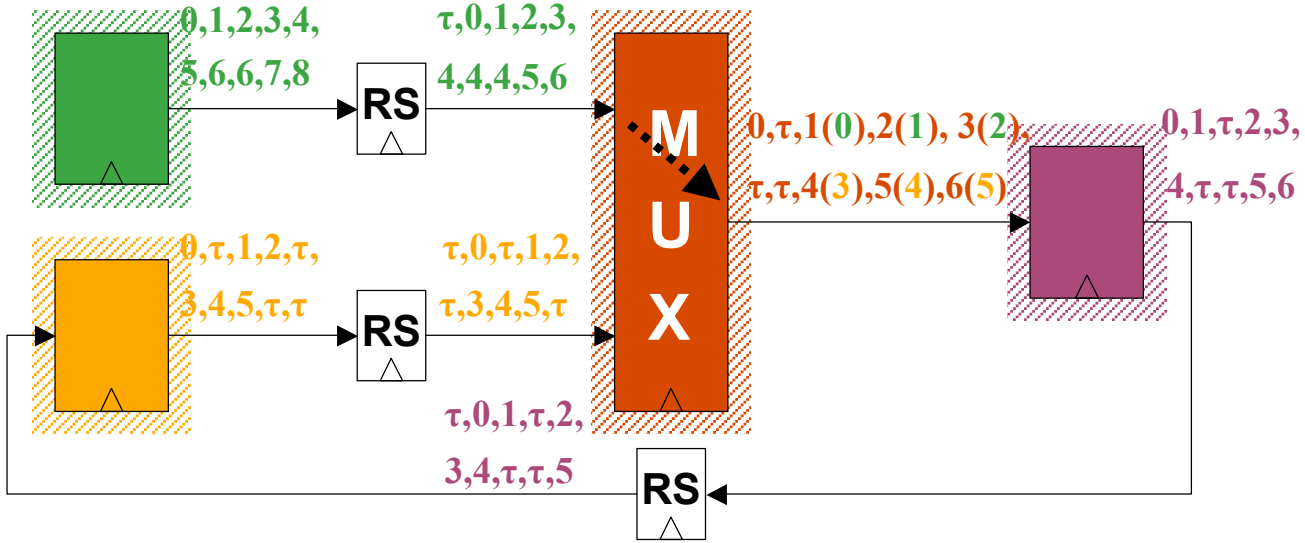
Adaptive LIPs

- One computation step later…
- Data 4 on upper input can now be discarded



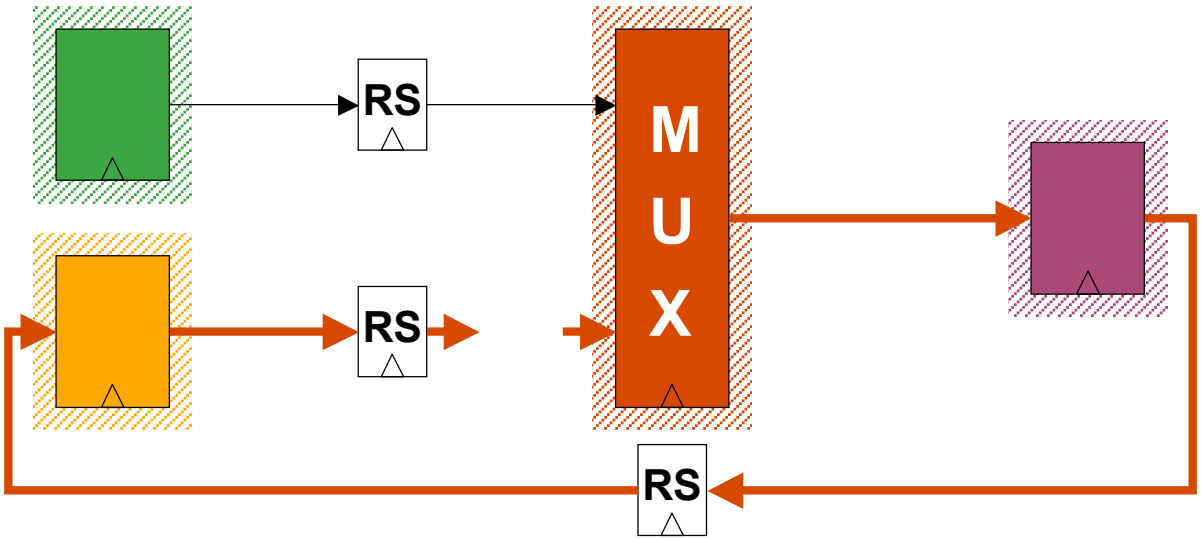
Adaptive LIPs

- Two steps later, the MUX switches on upper input
- Data label 6 already available
- Void data on lower channel ignored. Go ahead!



Adaptive LIPs

- Loops *open from time to time*
- Chance for higher throughput
- Critical loop? Behavior dependent



Throughput at steady state?

Adaptive LIPs: PROS/CONS

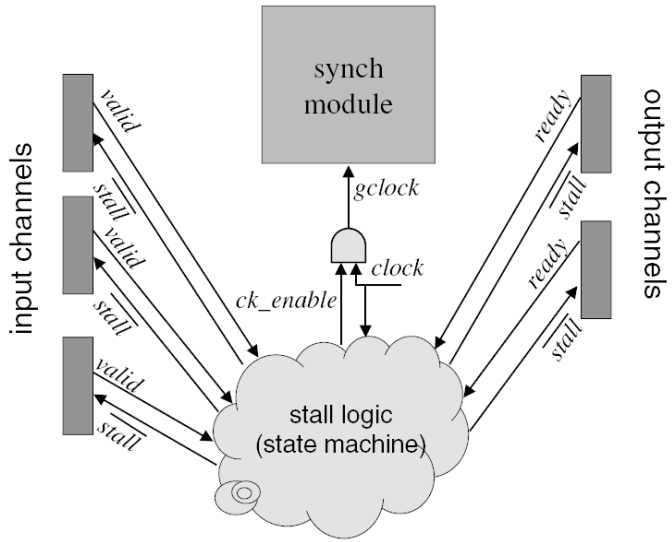
- PROS
  - Less restrictive conditions of applications will hopefully lead to higher average throughput than static LIPs
  - As a consequence, higher Data Rate at the same clock frequency
  - If input channel usage is unknown (for a part or even for the entire system) adaptive LIPs behavior converges to static LIPs
  - Can be adapted to GALS systems [Singh05]
- CONS
  - No pure orthogonalization of computation and communication
  - Adaptive wrapper more complex than static
  - Performance predictable only from statistics of channel access or from in-depth knowledge of computational behavior and not in closed form
  - Worst loop approach fails in capturing performance behavior

Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

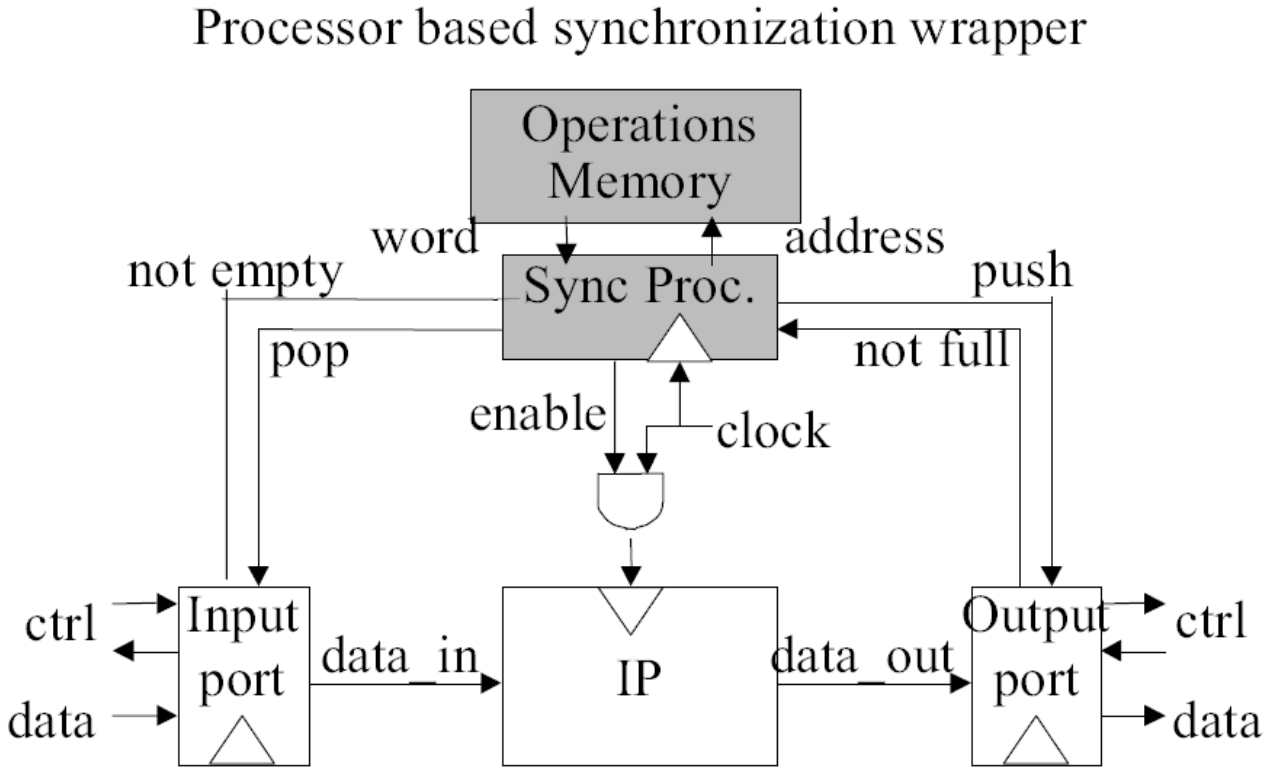
Practical issues

- [Singh03] and [Singh05]: companion FSM



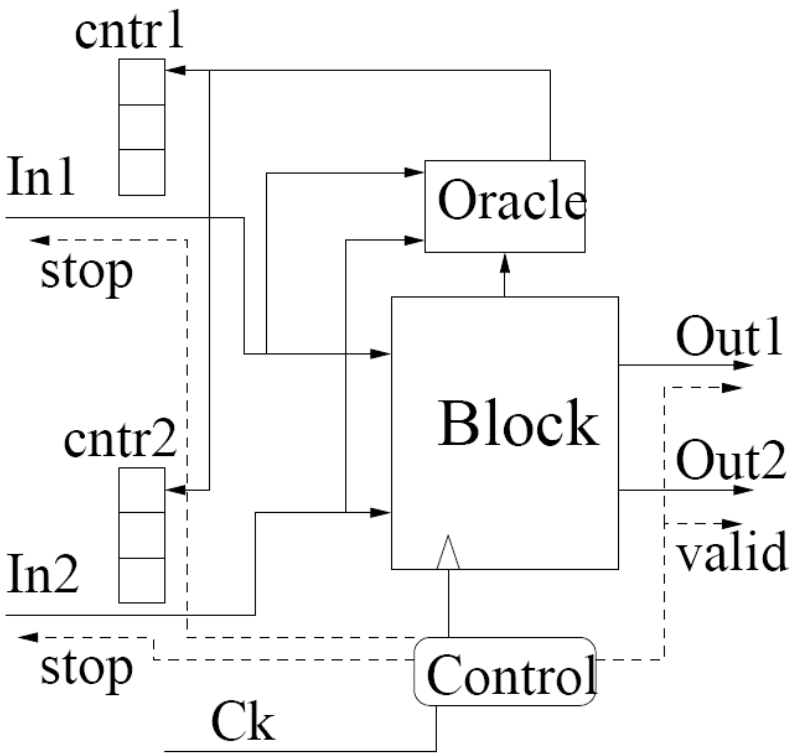
Practical issues

- [Bomel05]: synchronization processor



YAW: Yet Another Wrapper!

- Counters keep track of “virtual tags”…[DATE05]

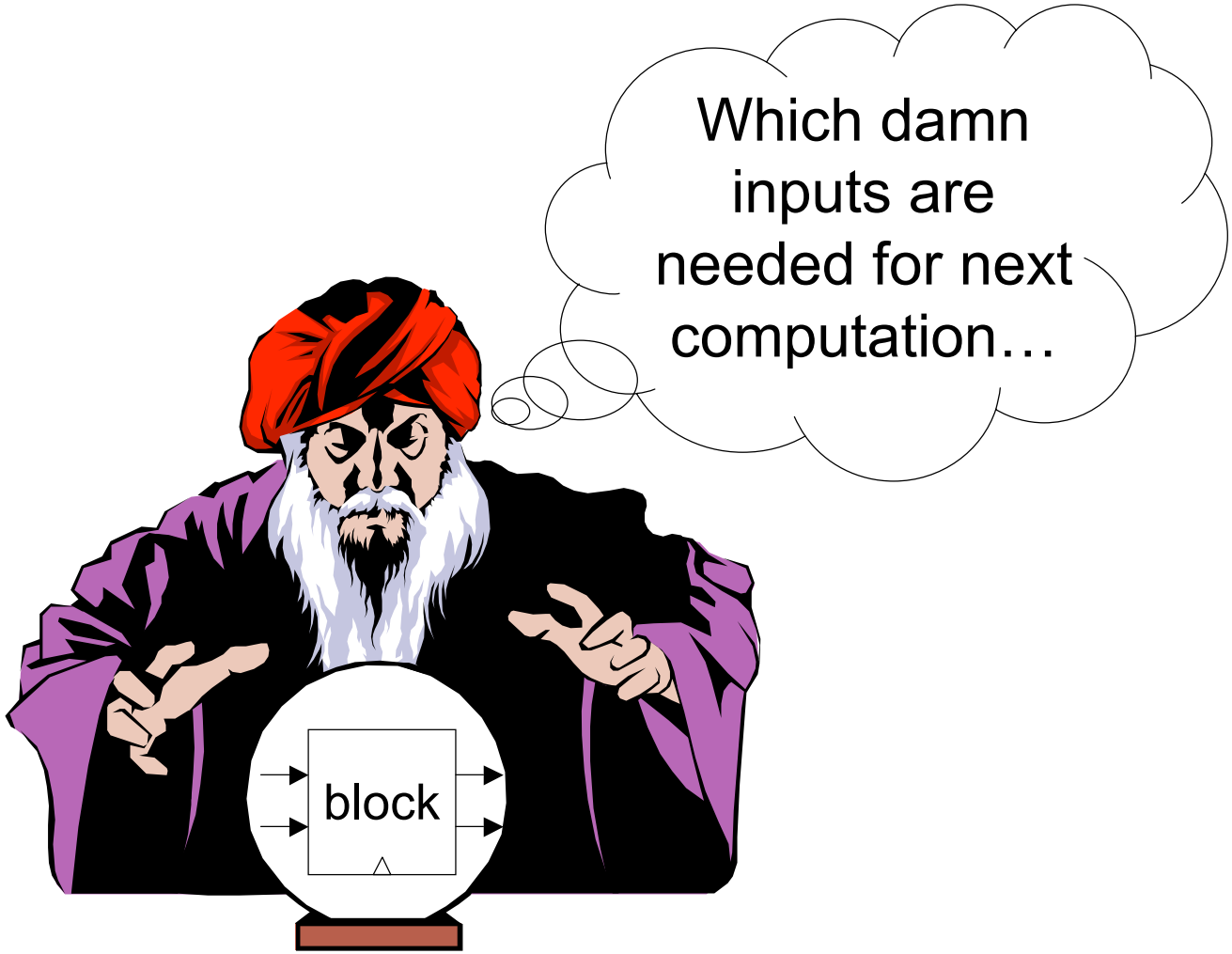


- INC on invalid or “old” valid on non-processed inputs
- DEC if input is valid, block is gated, and either counter is positive (waiting for old discarded signals) or non-processed input has a zero count (input can be discarded, but not next one)
- Min value = -1: in case of early non processed inputs we cannot predict if will be used in future…
- Max value? Back-pressure signal emitted to avoid overflow
- How about the oracle?

The oracle

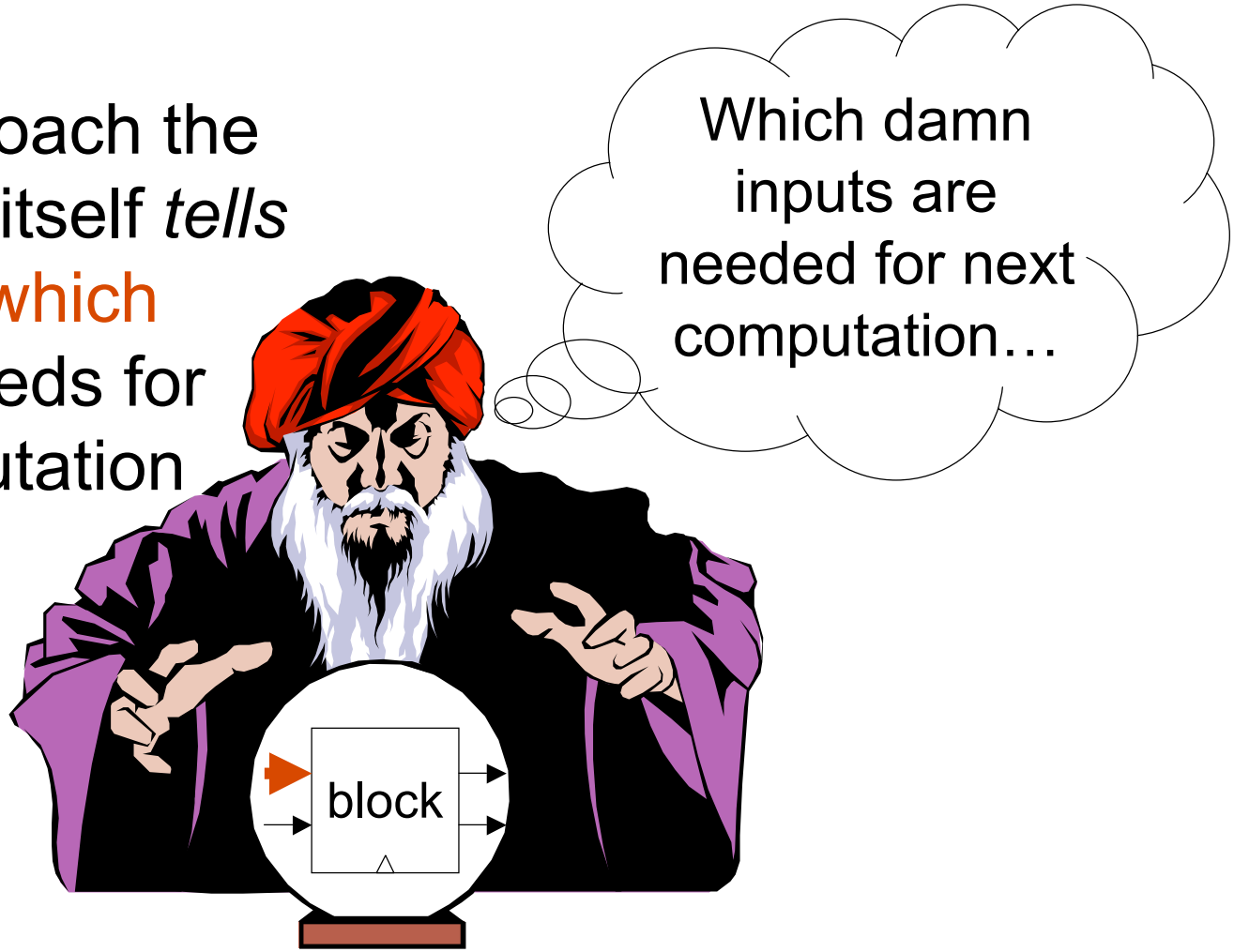
The Delphic Sybil (Pythia),
1509, Sistine Chapel, Michelangelo

The oracle



The oracle

- In our approach the logic block itself *tells* the oracle which inputs it needs for next computation



The oracle

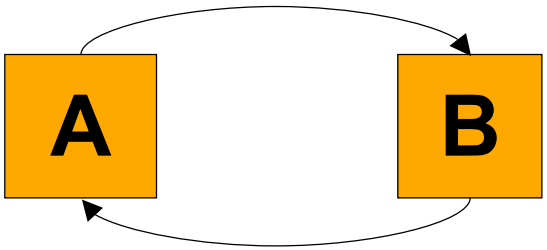
- The logic block tells the oracle which inputs it needs for next computation (no black magic…)
- Instead of being precharacterized (e.g. through simulations), some blocks can be slightly modified to emit a "processing signal" for all or a subset of inputs
  - Modifications are not strictly needed to make the wrapper works. If the block does not use processing signals, the wrapper behaves in a static fashion
  - Modifications are not always necessary, example: cpu/memory interaction through *explicit* wr/rd requests

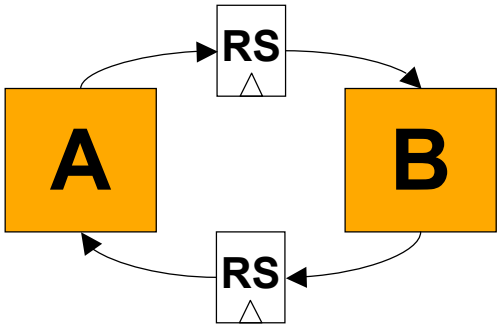
Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

How to get real speedup

- Static LIPs really endangered. Example
  - Data rate of 2 tightly interacting blocks. DR = f · Th





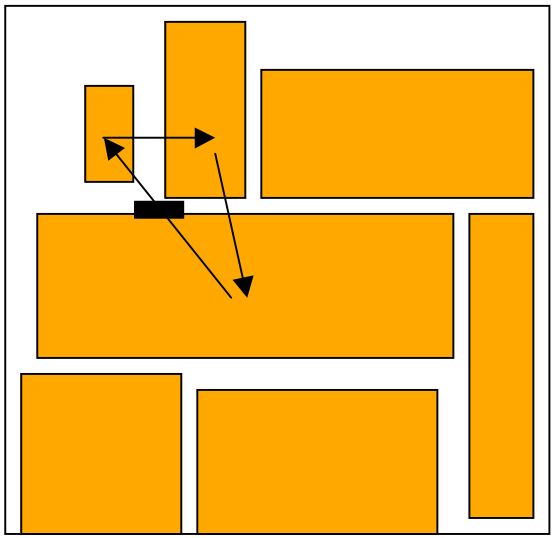
$DR_{no\ LIP} = f_{no\ LIP} \cdot 1$

$DR_{LIP} = f_{LIP} \cdot 1/2$

- Hard to get $f_{LIP} > 2 \cdot f_{no\ LIP}$. Better avoid RS in tight loops through proper ***physical design***
- Adaptive LIP may help increase DR (no guarantee!)

Floorplanning for Throughput

- Standard floorplan problem:
  - *find a placement of blocks that minimizes whitespace, overall wirelength, critical path, or a combination*
- **Static LIP case:**
  - *floorplan maximizes throughput (possibly multi-objective)*
  - Maximum throughput equivalent to worst cost-to-time ratio loop
  - No need to enumerate loops (exponential): cost evaluation algorithm $O(EV^2)$ [TCAD05]



Floorplanning for Throughput

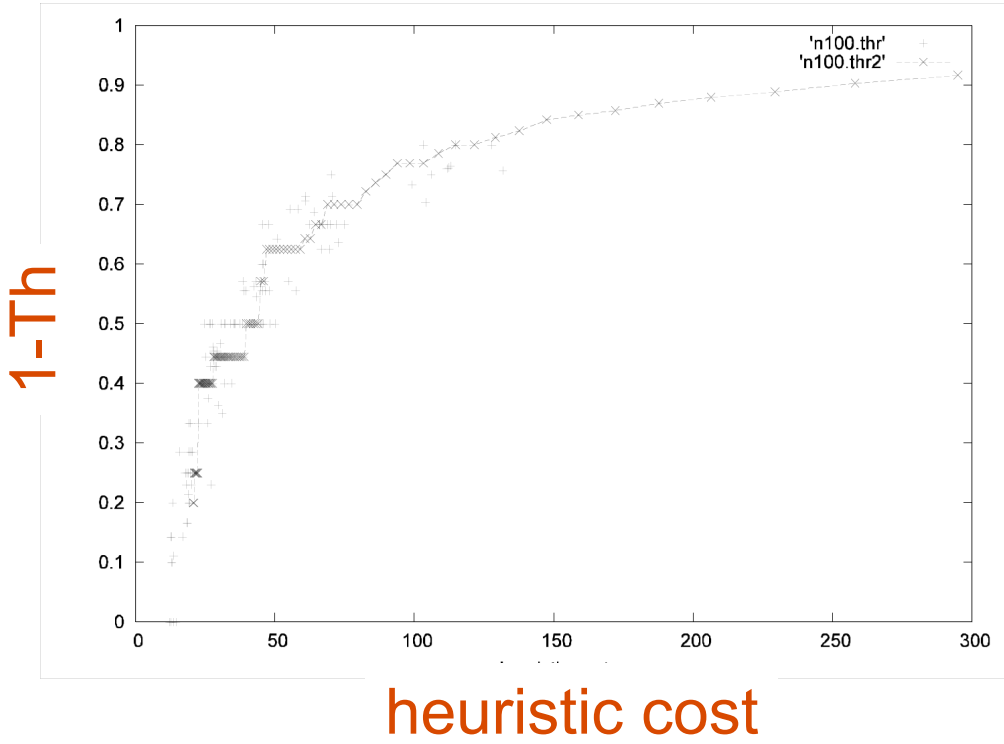
- Simulated annealing main features
  - System is cooled from a high initial temperature T_0
  - If cooling is slow enough a minimum of energy is reached
  - Moves accepted with probability exp(- δ/T) if reduce energy of δ
- Our work builds on *Parquet* [Markov03]
  - Energy becomes a cost function (Th, WL, A, HPWL, or a combo)
- Problems with exact cost evaluation
  - CPU time too high inside the optimization loop:
  - Avg/Max CPU time: 0.2/1.1 s on MCNC and GSRC benchmarks
  - Exact cost function not that smooth ("max" evaluation), especially when close to the solution

Floorplanning for Throughput

- Heuristic should be smooth and easy to compute and follow monotonically the real cost. A good one is
  - Statically compute the shortest loop *l(e)* in which every edge *e* appears (outside the iteration loop)
  - For every optimization iteration:
  1. *∀e, cost(e)=1/l(e)·latency(e)*
  2. *TotCost=Σcost(e)*
- *latency(e)*
  - *floor of the edge's Manhattan length divided by the max length between clocked elements (e.g. previously defined critical length, l_{crit} in the following)*

Floorplanning for Throughput

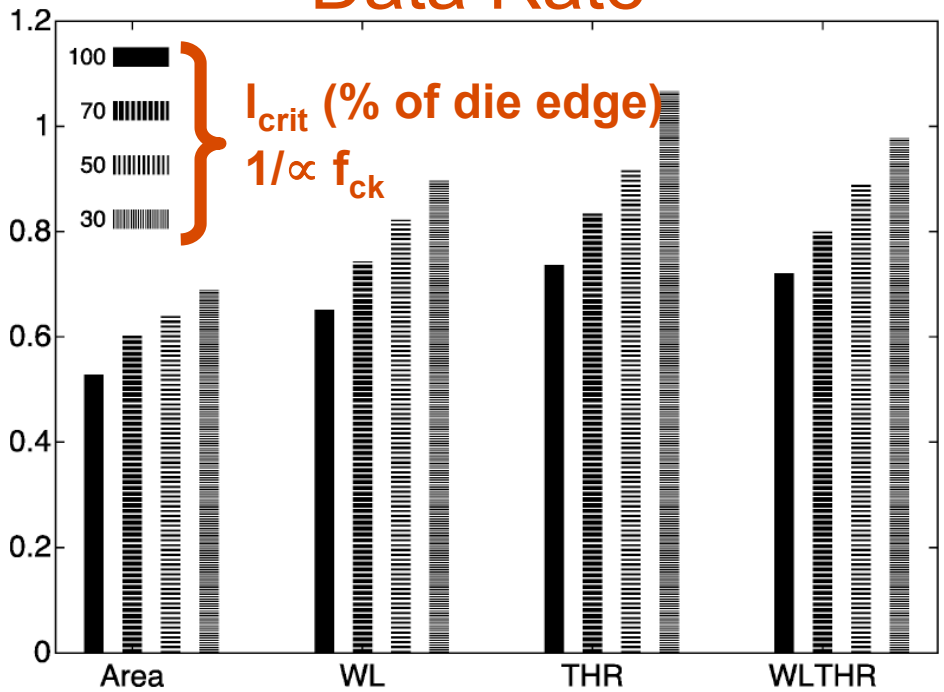
- Heuristic properties
  - Considers only relevant nets
  - Long nets not in short loops discarded
  - Computationally light
  - Smooth (function of the whole circuit rather than a max value)



Floorplanning for Throughput

- Th and DR results
  - GSRC and MCNC benchmarks
  - floorplans obtained varying l_{crit}
  - On avg: 25% better than area and 11% better than wirelength cost functions
  - Better gain at long l_{crit}: 64% and 24% if l_{crit}= die edge
- Data Rate increases at shorter l_{crit}
  - higher clock frequency overcompensates throughput degradation.

Data Rate



Caveat: clock overhead not considered (skew, ...)

Did we get real speedup?

- OK, but how does it compare with no wire pipelining at all
  - i.e. clock frequency slow-down
- Speedup SU = DR/DR_0: upper & lower bounds [TCAD06]
  - $L/(l_{crit} + \langle l_{e,loop} \rangle) \leq SU \leq L/\langle l_{e,loop} \rangle$
- $L \geq l_{crit}$ is the interconnect length which sets the clock frequency limit in a no LIP system
- $\langle l_{e,loop} \rangle$ is the average length of the edge of the worst loop
  - Best floorplan minimizes the average length of the worst loop
- No matter how fast is clock (possibly infinite, i.e. $l_{crit} \rightarrow 0$), the maximum speedup is upper bounded!
  - unless the netlist is devoid of loops!

Did we get real speedup?

- Results obtained letting the tool seek for the optimal floorplan varying l_{crit}. It always turned out that $l_{crit} \rightarrow 0$, confirming math formulation

bench.	#blocks	*DR*	DR_0	*L*(%)	*SU*(%)	$l_{e,loop}$(%)
n10	10	0.961	0.852	117	+13	104
n30	30	0.979	0.727	138	+35	102
n50	50	0.793	0.617	162	+29	126
n100	100	1.114	0.555	180	+100	90
apte	9	0.705	0.699	143	+1	142
xerox	10	0.613	0.565	177	+9	163
hp	11	0.660	0.511	196	+29	151
ami33	33	1.106	1.039	96	+6	90
ami49	49	1.047	0.774	129	+35	96

Floorplanning in Adaptive LIPs

- When a block in a loop ignores a subset or all inputs, is actually *breaking the loop*
- Performance modeling: a given block's task needs *N* computations of which
  - αN done with "closed" loop and $(1 - \alpha)N$ with "open" loop ($\alpha \leq 1$)
- *α is called channel activation ratio*
- Each computation takes one clock cycle when the loop is open and *1/Th* clock cycles when closed.
- The number of ck cycles required to finish is
  - $M = (1 - \alpha)N + \alpha N/Th.$
- The *effective throughput* of the loop is $Th_e = \frac{N}{M} = \frac{1}{1 - \acute{a} + \frac{\acute{a}}{Th}}$
  - $Th_e > Th$ if $\alpha < 1$

Floorplanning in Adaptive LIPs

- Modified floorplan cost function [TCAD06]
  - Statically compute the shortest loop *l(e)* in which every edge *e* appears (outside the iteration loop)
  - For every optimization iteration:
  1. $\forall e,\ cost(e)=1/l(e)\cdot latency(e)\cdot w(e)$
  2. $TotCost=\Sigma cost(e)$
- The only change consists in the inclusion of a weight *w(e)* that depends on the channel activation ratio *α(e)*
- Several strategies possible
  - $w = \alpha,\ w = max_{loop}(\alpha_i),\ w = 1/(2-\alpha)\ldots$

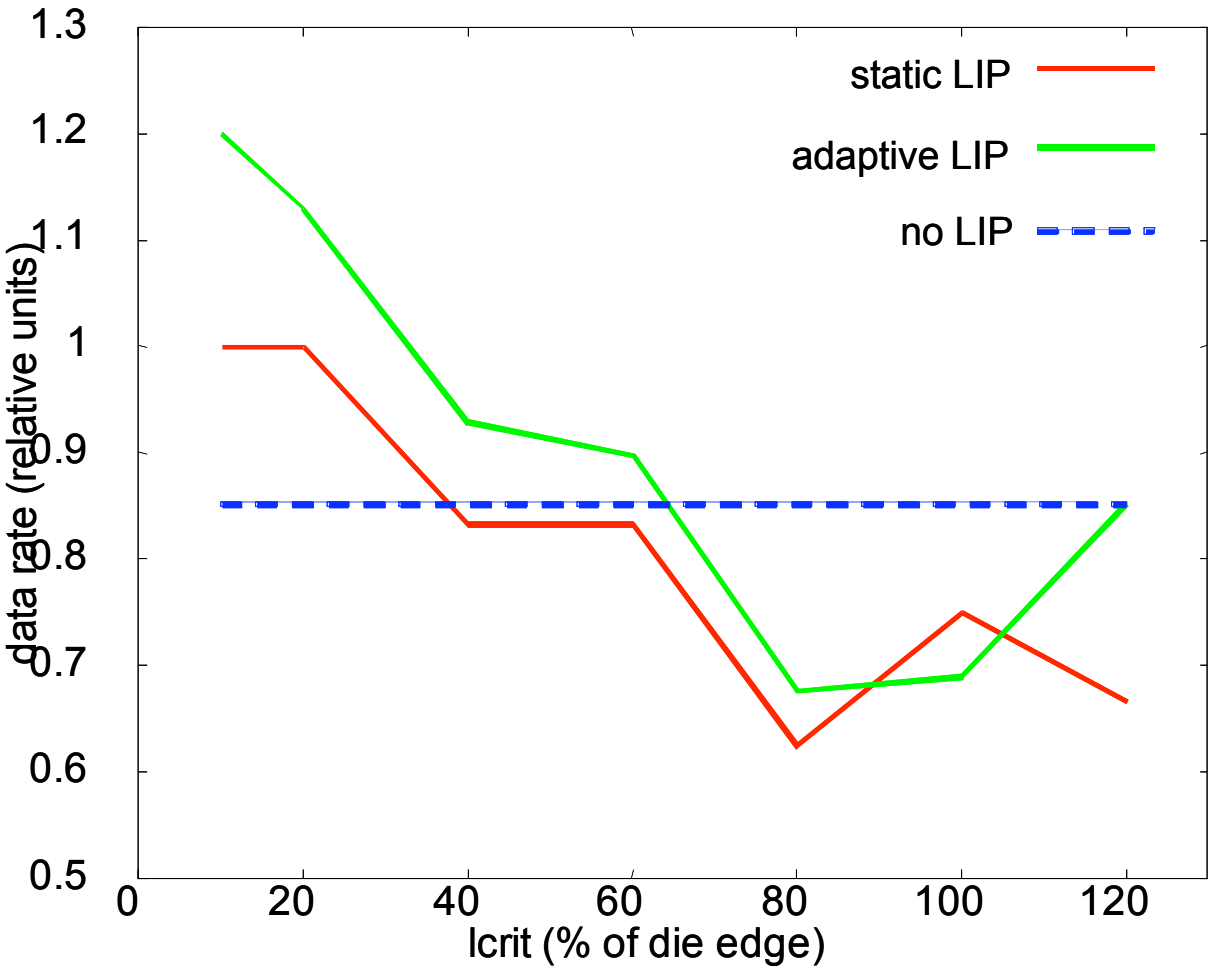
Floorplanning in Adaptive LIPs

- Problem with floorplan benchmarks:
  - *how to assign channel activation ratios α's?*
- GSRC and MCNC benchmarks random assignment…
  - Hypothesis: channels used in *burst* mode
- MPEG encoder and small CPU measured *α's*
  - Need for post-layout verification (cannot evaluate Th a priori)
- Floorplanner output gives also a performance estimate (to be compared with actual simulations)
  - Calculated with worst effective throughput Th_e

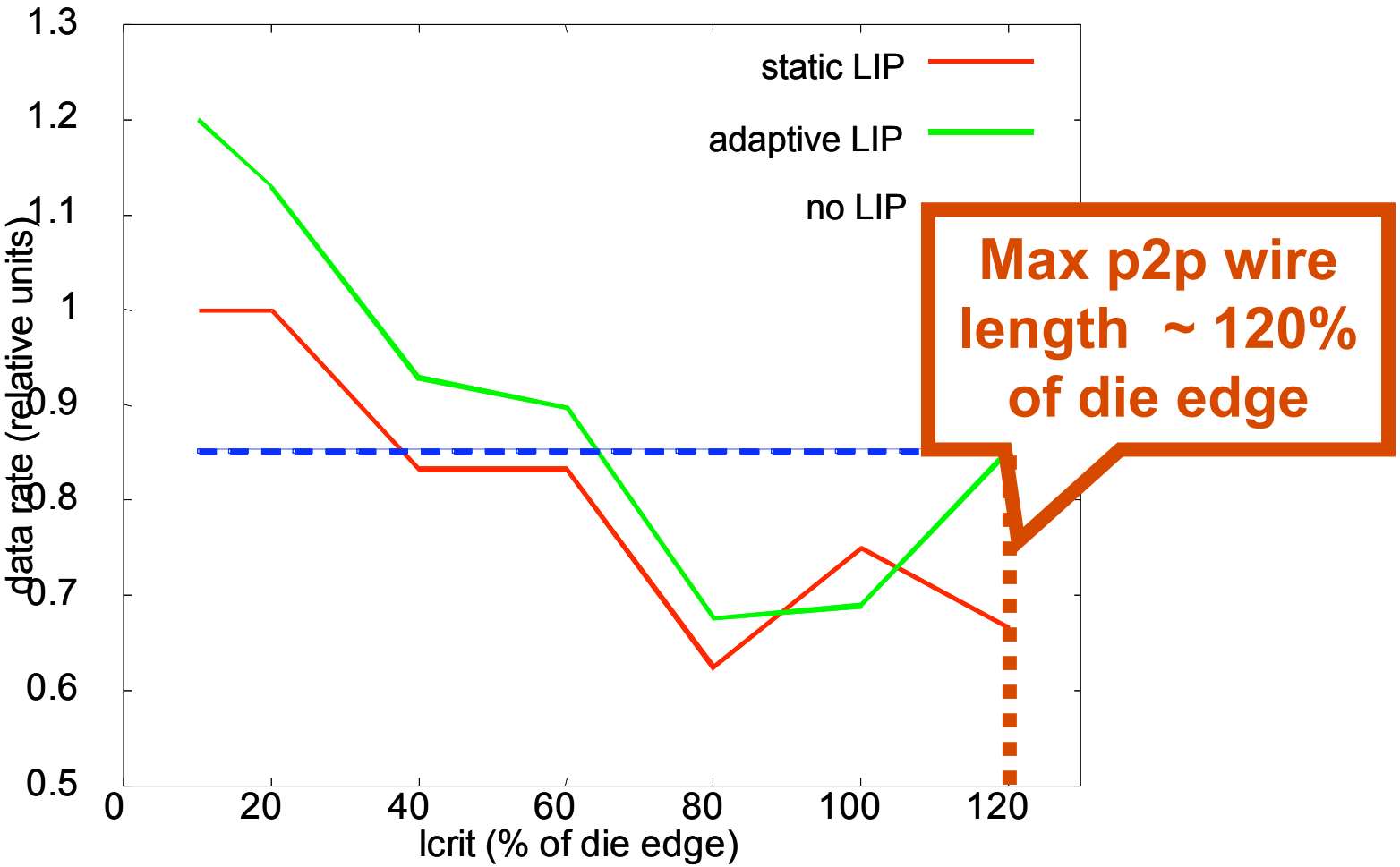
Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

Example: GSRC n10

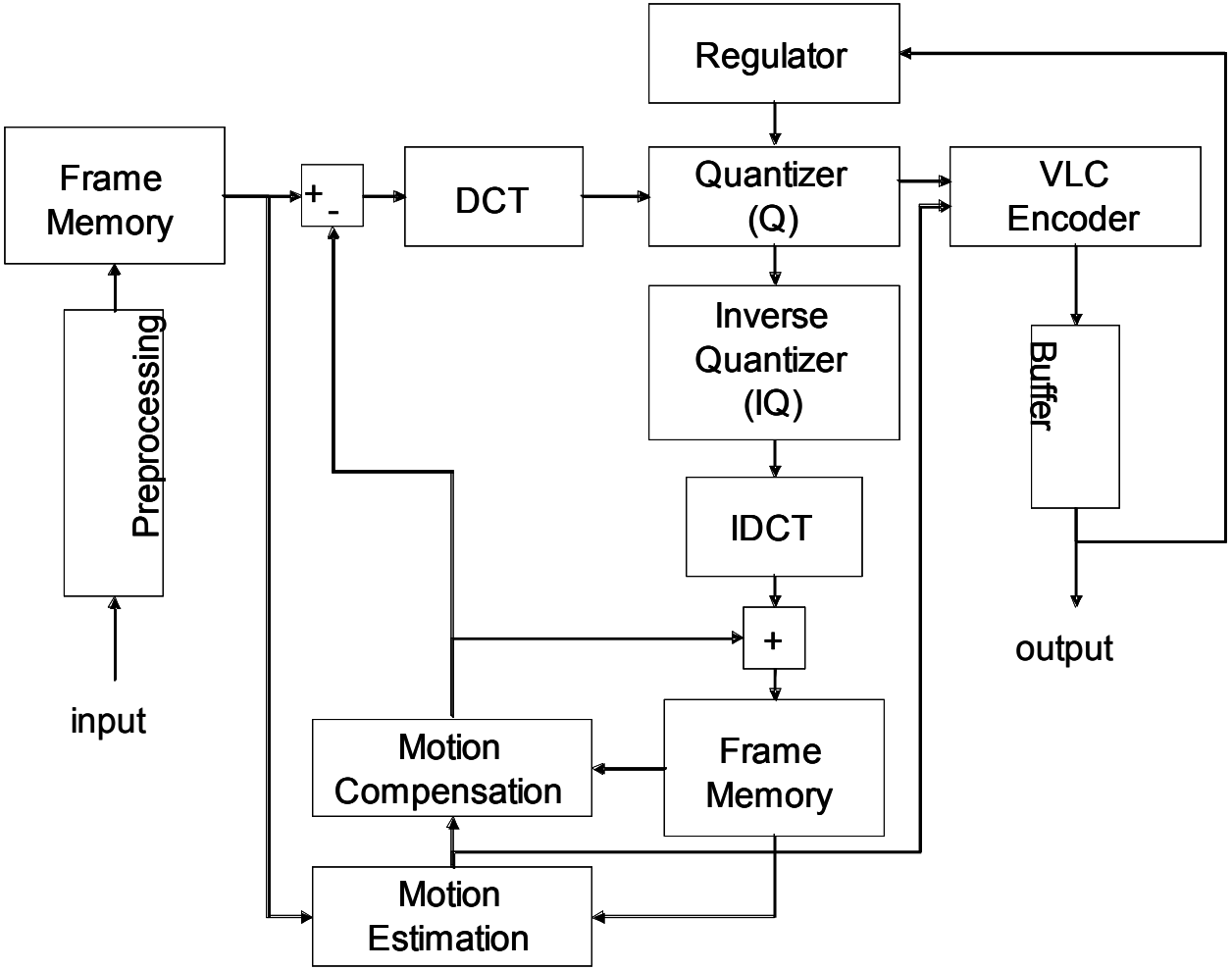


Example: GSRC n10



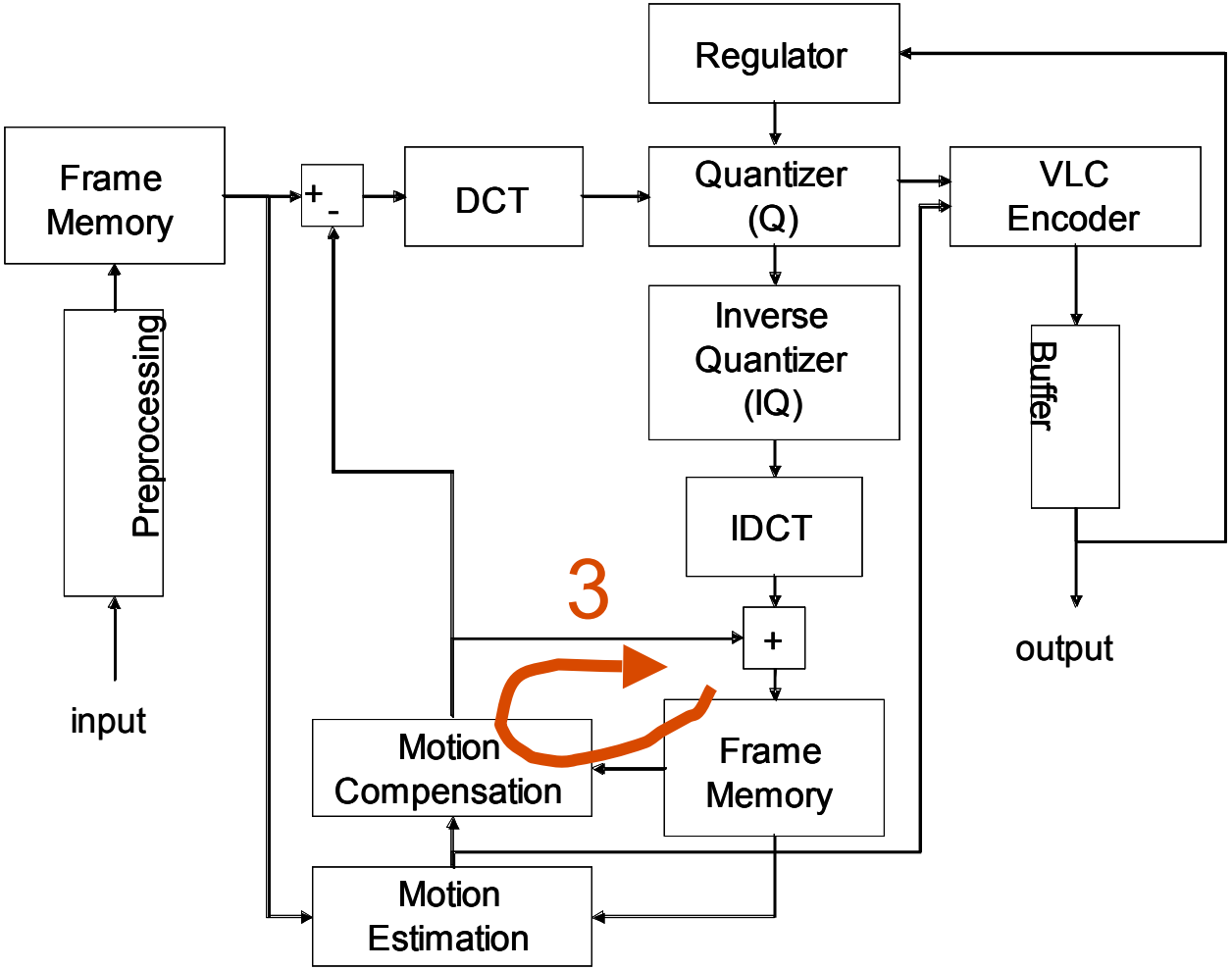
Example: MPEG [NTT96],[NTT99]

- Case of study in [Carloni00]



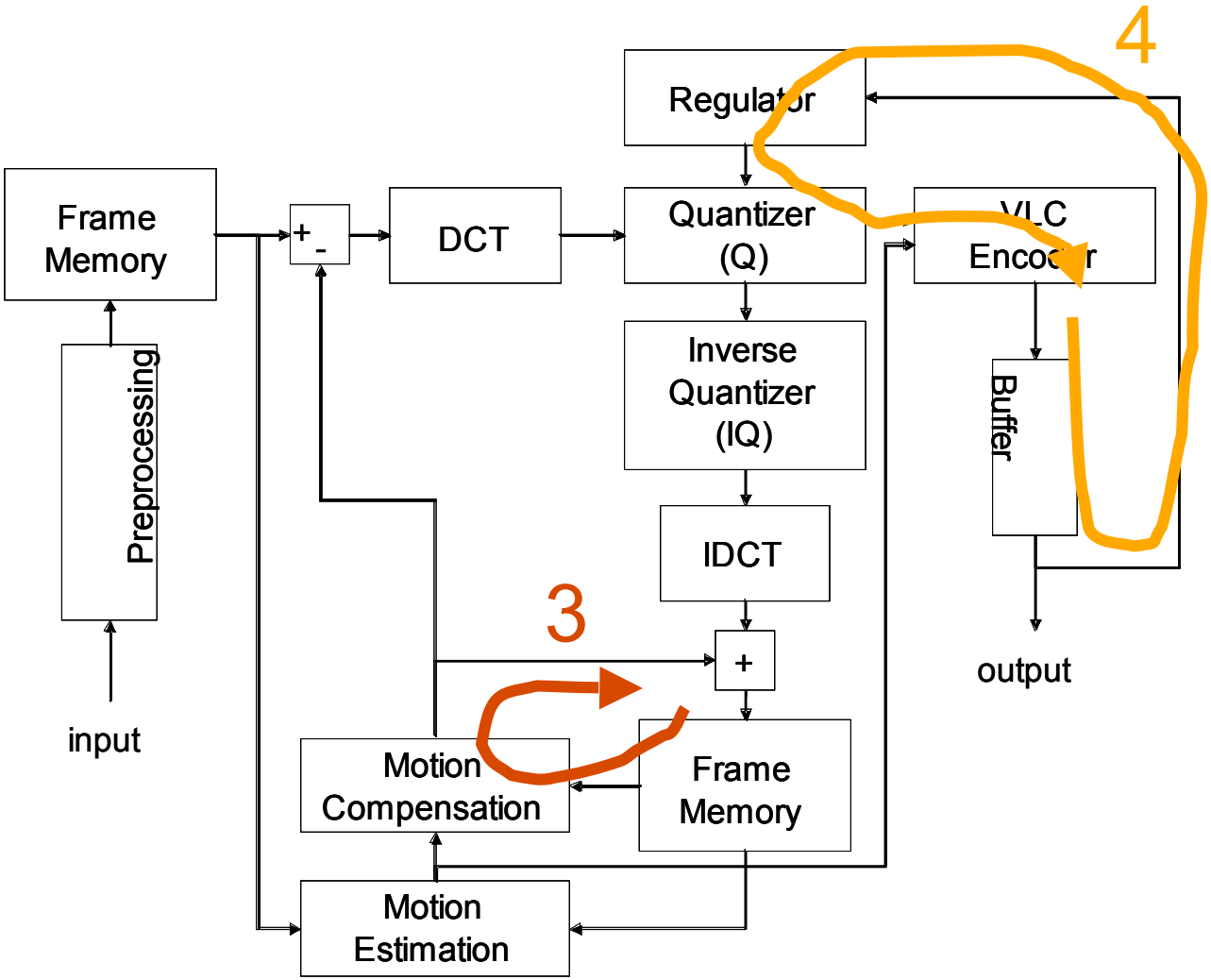
Example: MPEG [NTT96],[NTT99]

- Case of study in [Carloni00]



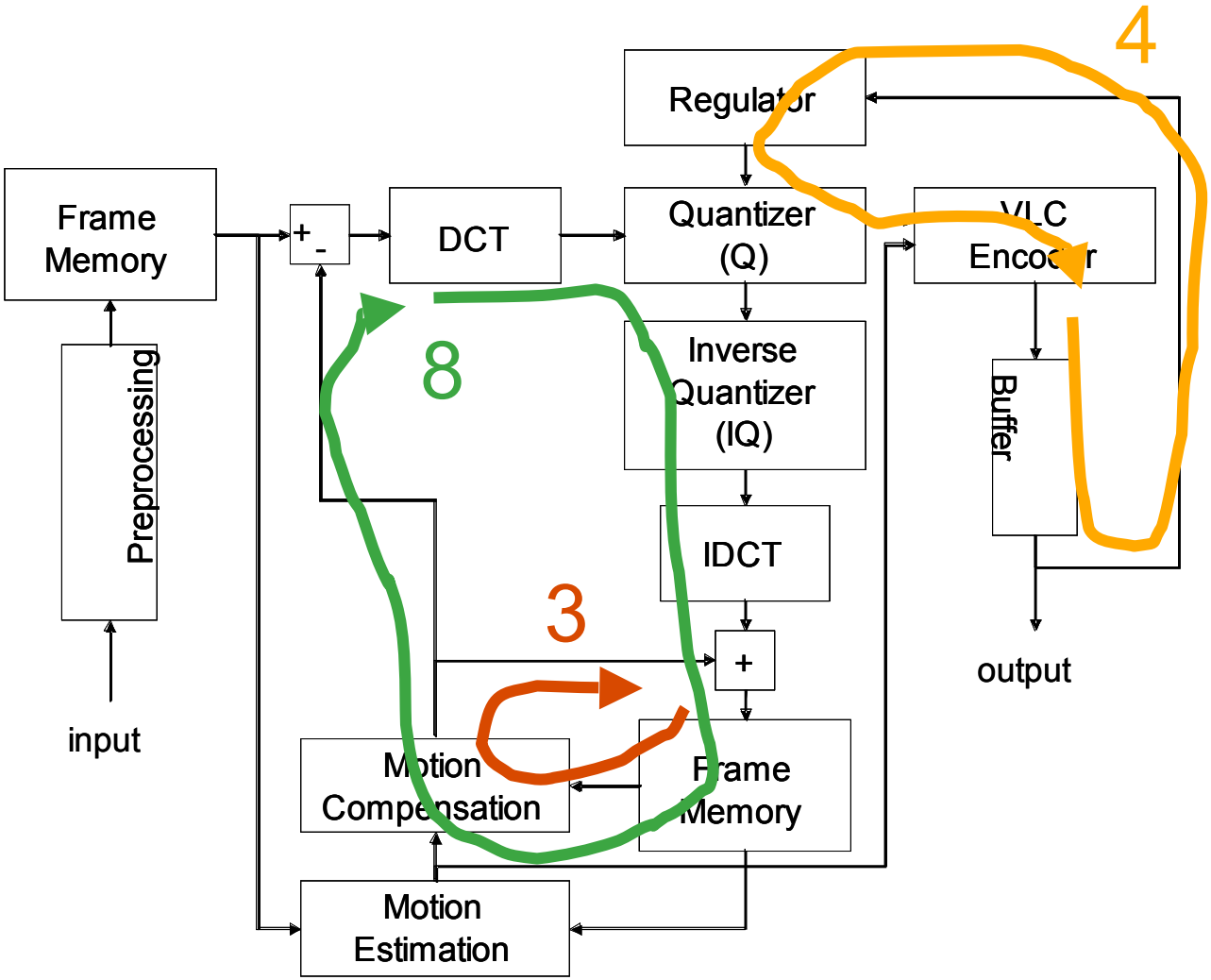
Example: MPEG [NTT96],[NTT99]

- Case of study in [Carloni00]



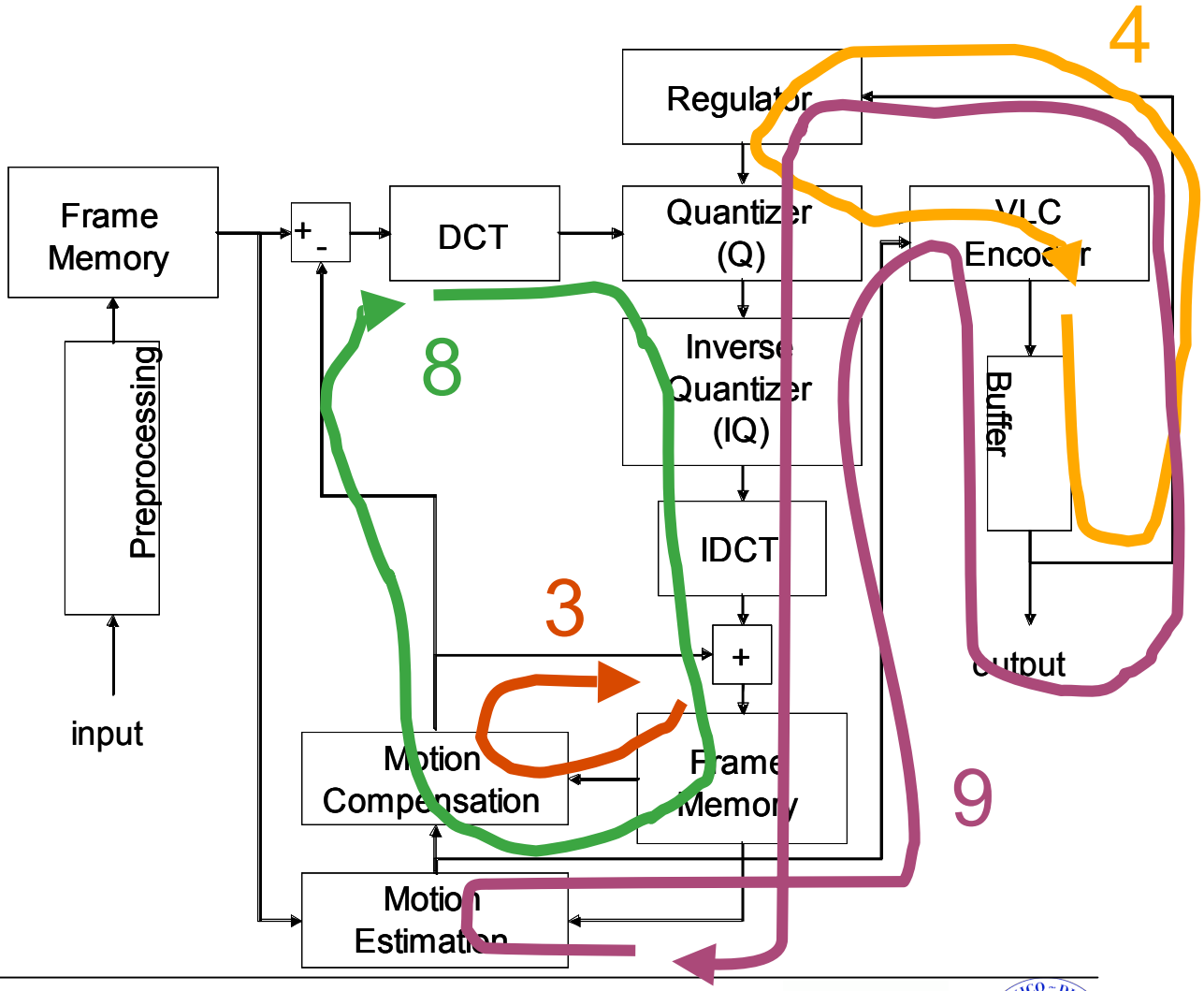
Example: MPEG [NTT96],[NTT99]

- Case of study in [Carloni00]

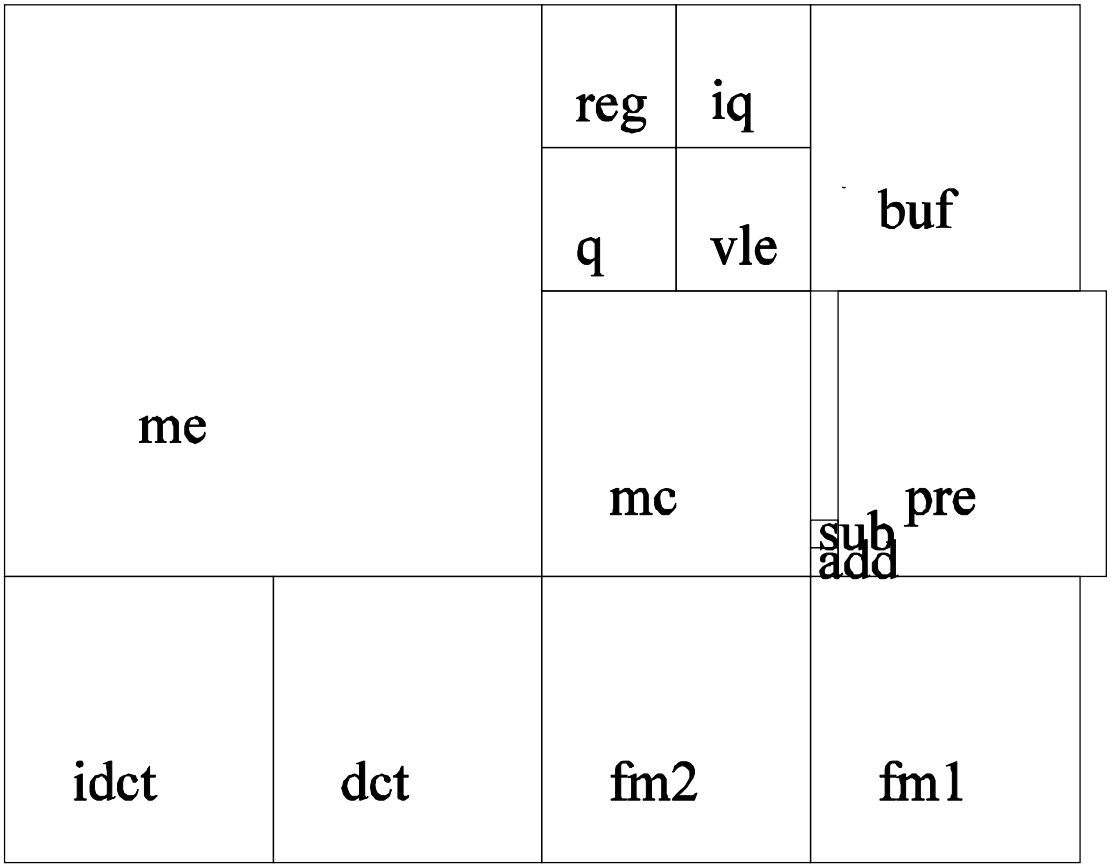


Example: MPEG [NTT96],[NTT99]

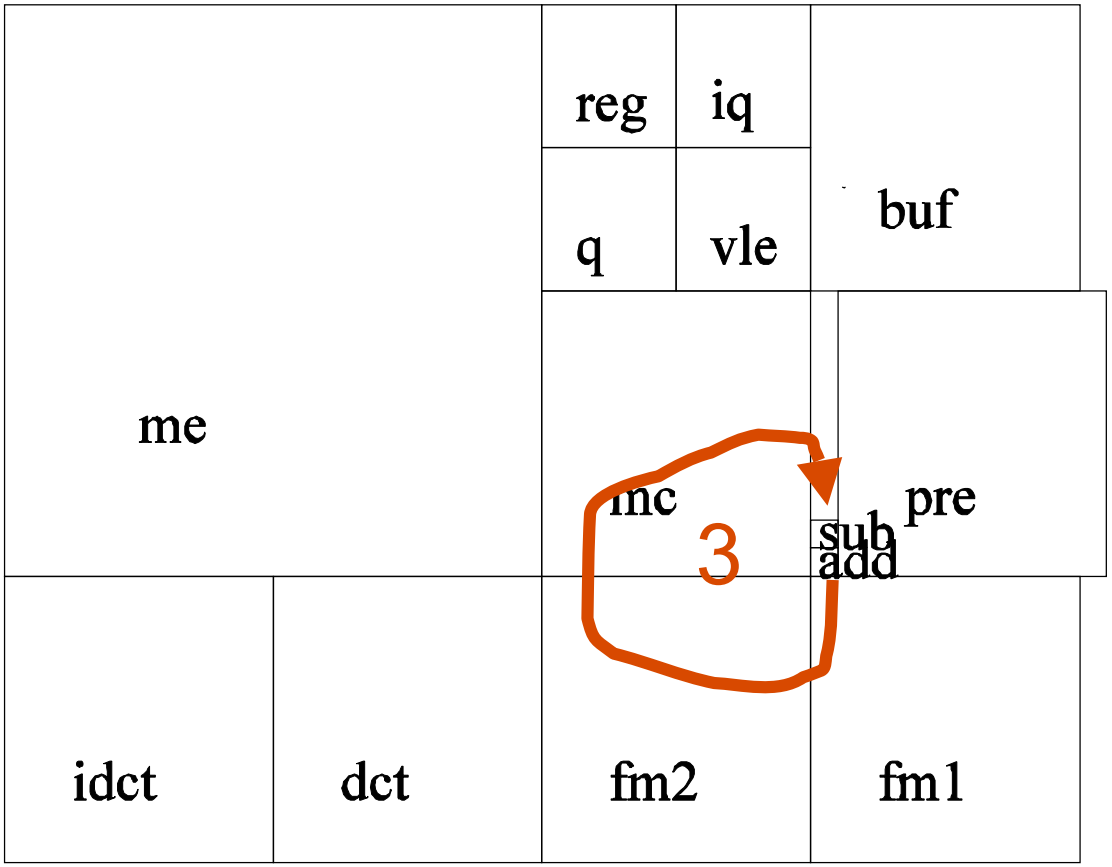
- Case of study in [Carloni00]



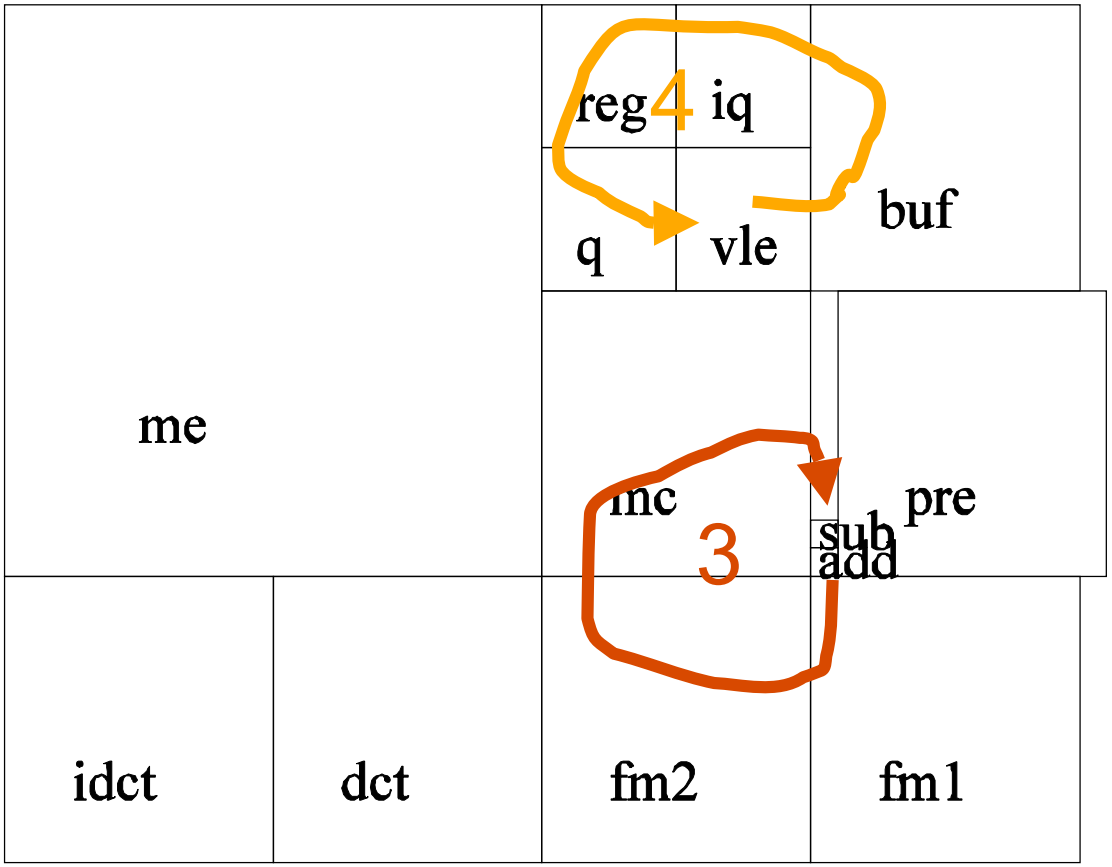
Example: MPEG [NTT96],[NTT99]



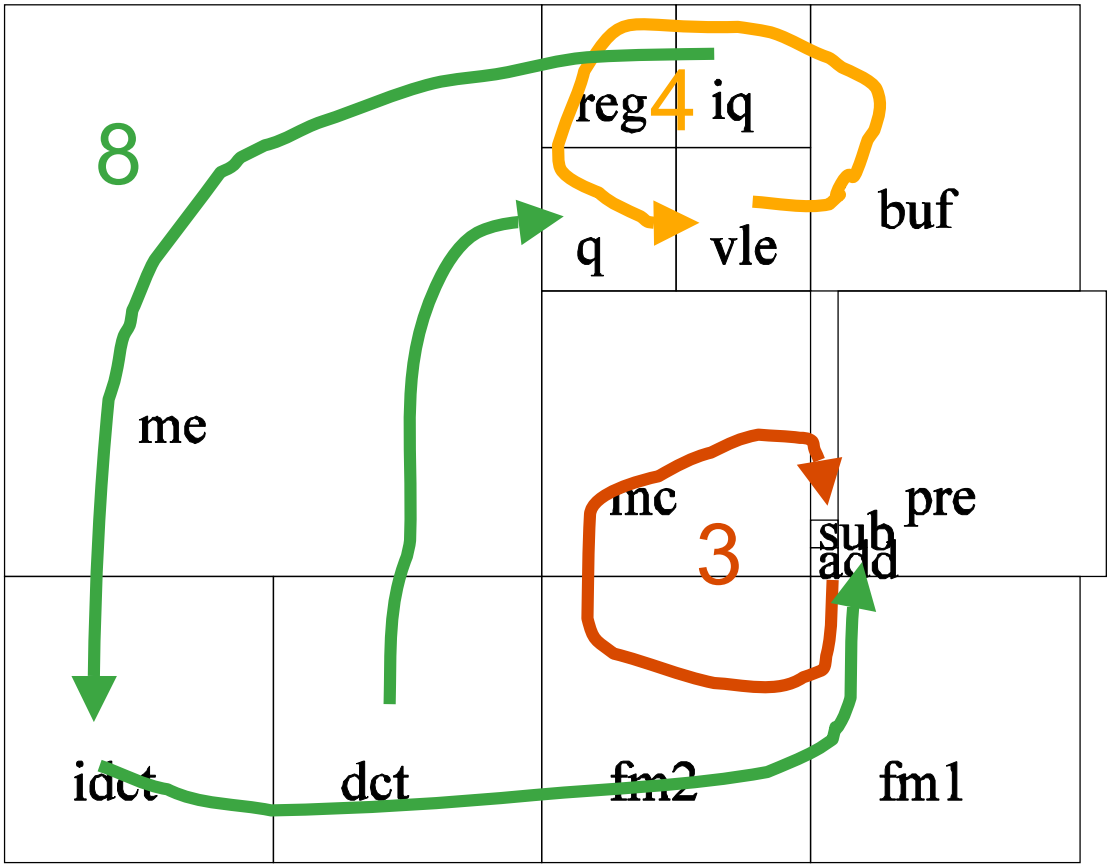
Example: MPEG [NTT96],[NTT99]



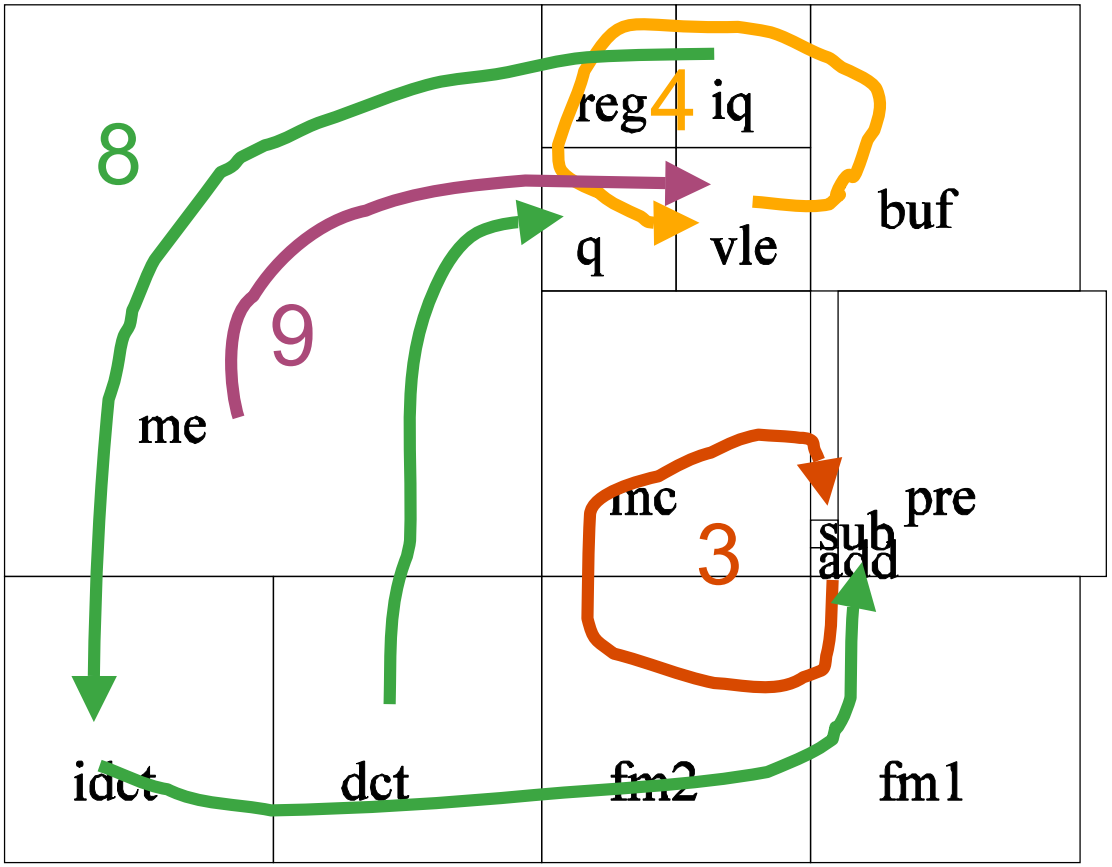
Example: MPEG [NTT96],[NTT99]



Example: MPEG [NTT96],[NTT99]

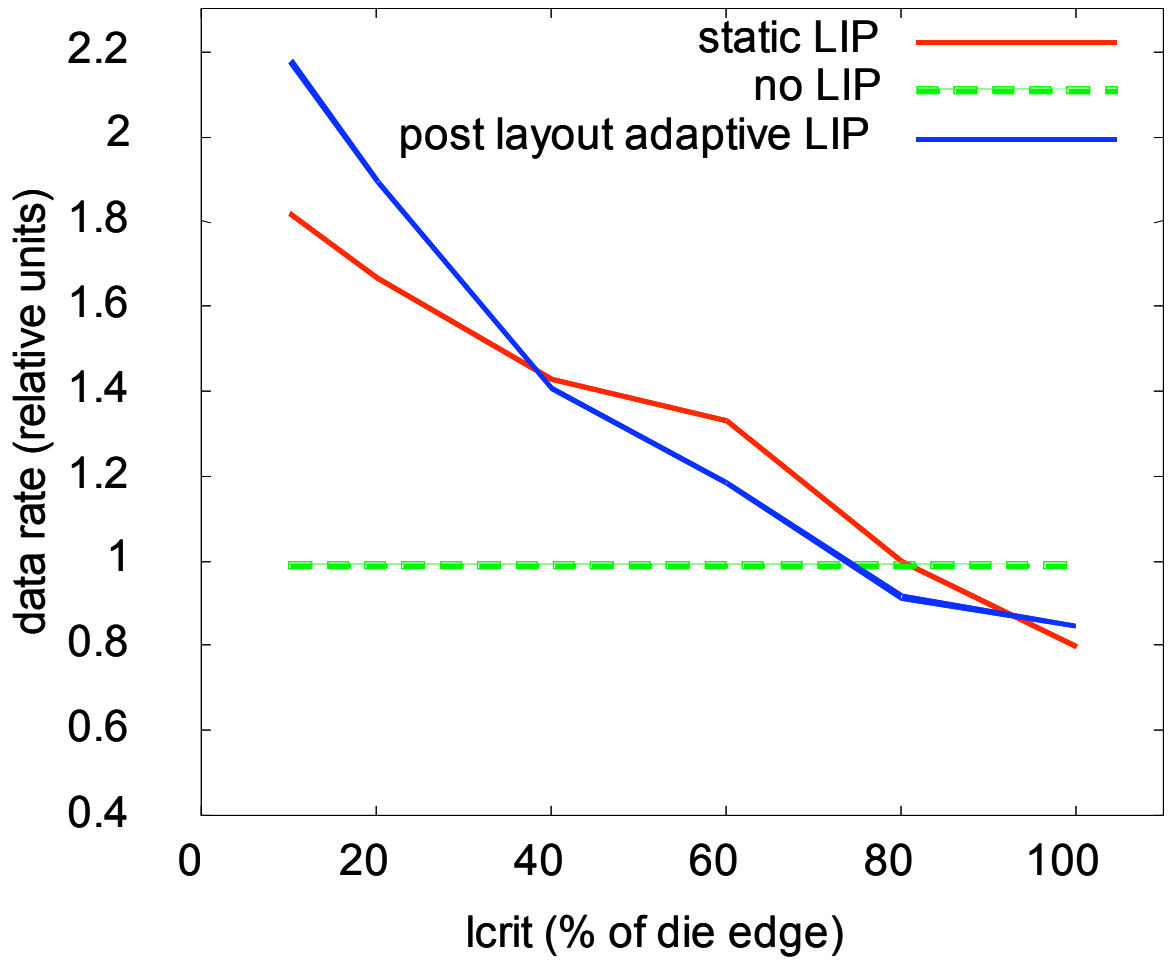


Example: MPEG [NTT96],[NTT99]

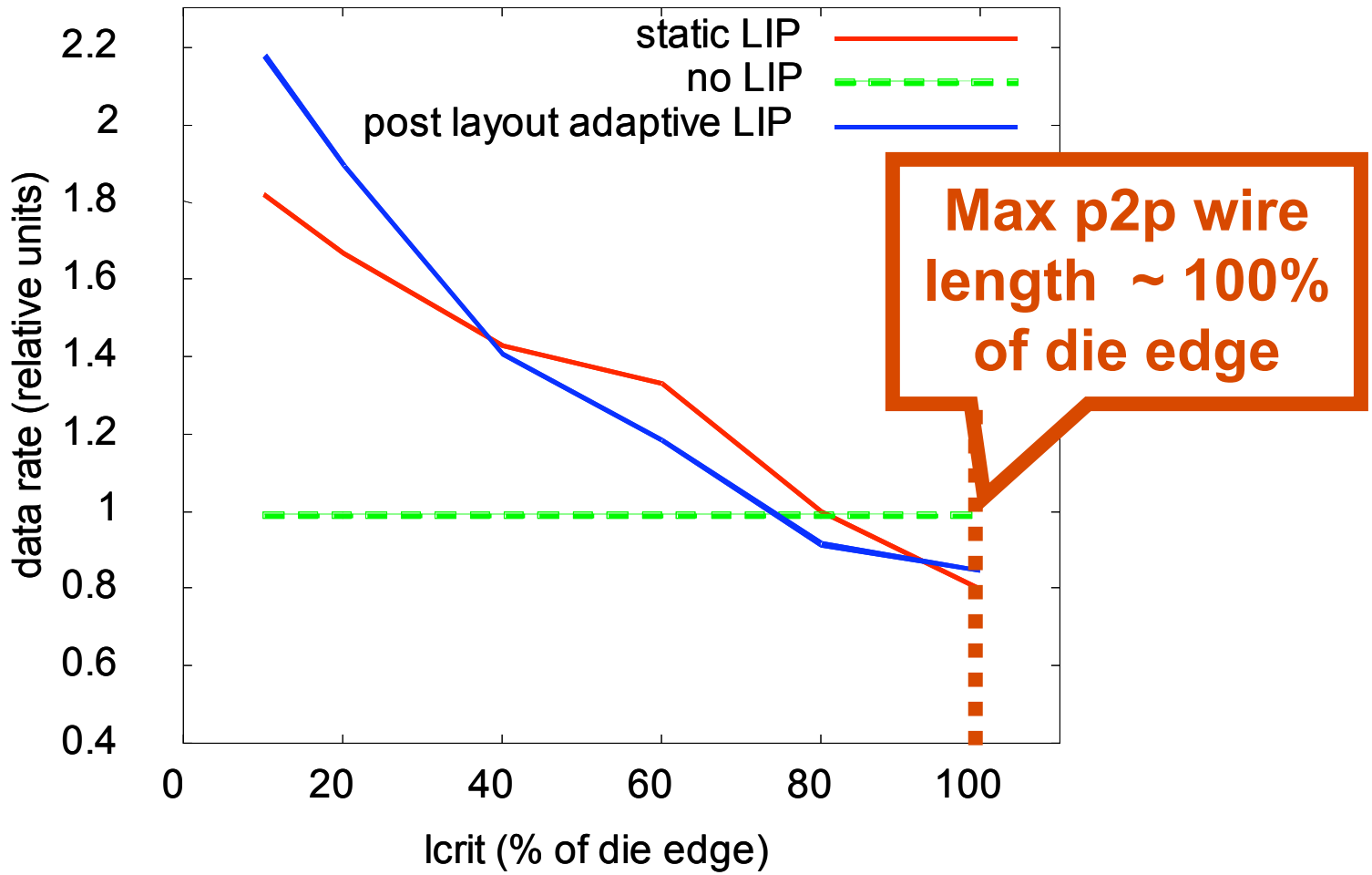


Example: MPEG [NTT96],[NTT99]

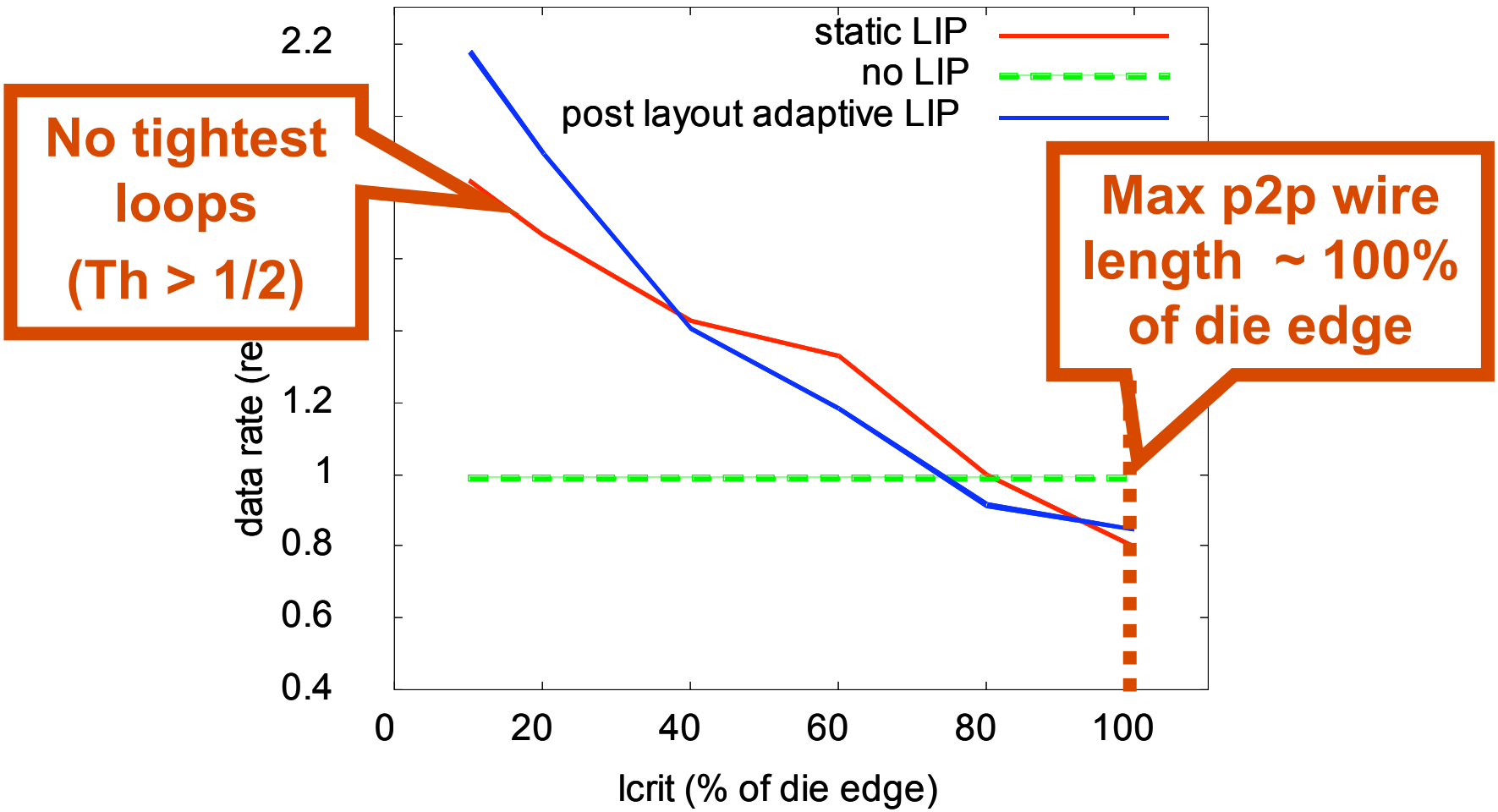
Example: MPEG [NTT96],[NTT99]



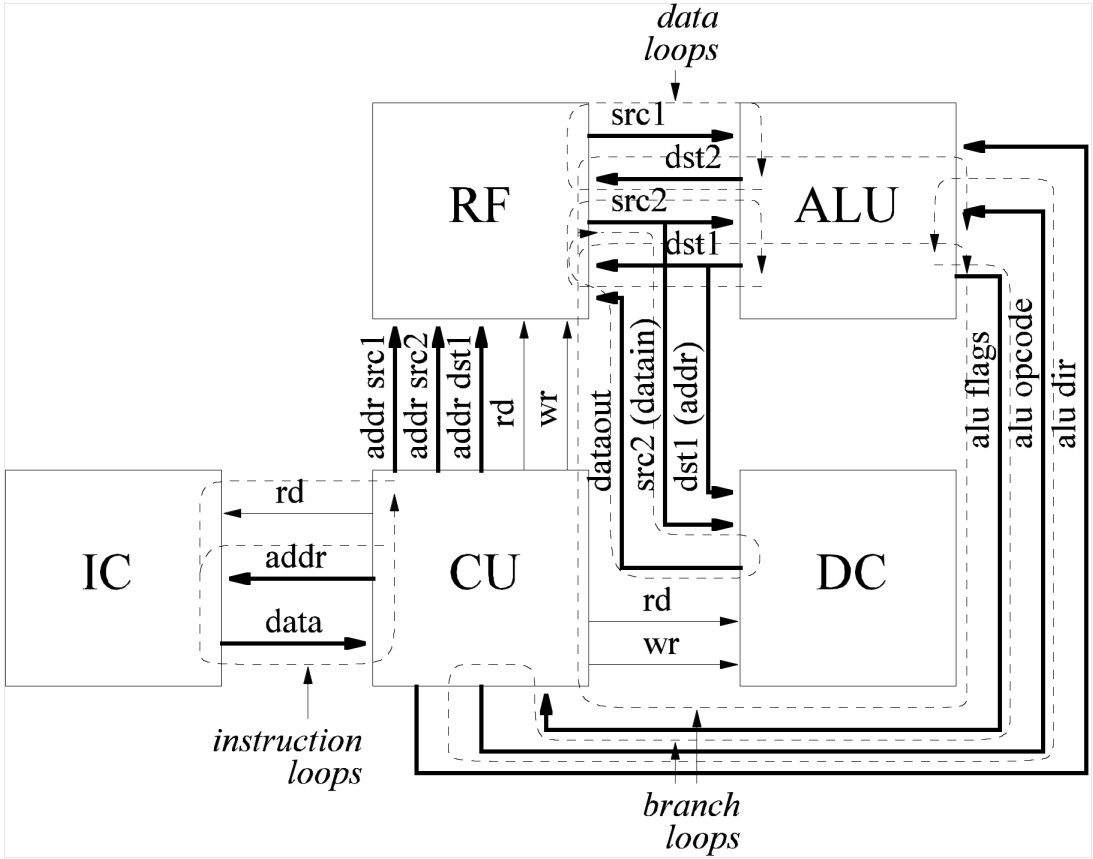
Example: MPEG [NTT96],[NTT99]



Example: MPEG [NTT96],[NTT99]

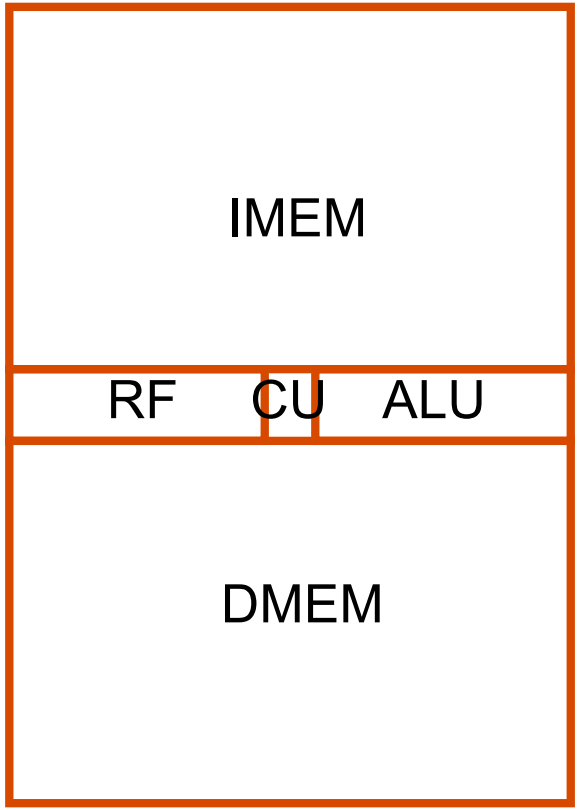


Example: small CPU



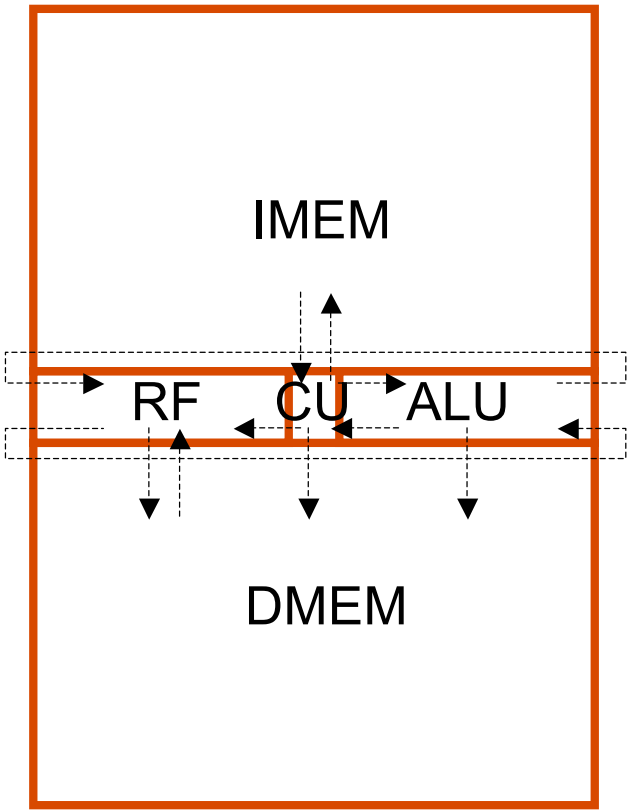
- Many “tight” loops
- Easy to derive channel activation ratios and input “processing” signals (for the oracle…)
- Post layout code exec. Two programs:
  - Matrix multiply exercises mostly RF-DMEM loops
  - Extraction Sort activates mainly CU-RF-ALU branch loops

Example: small CPU



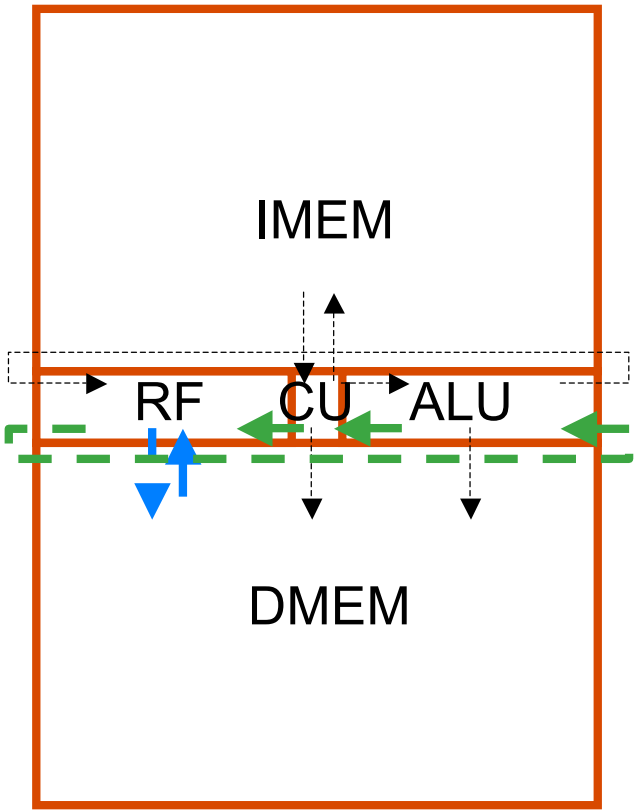
- Many “tight” loops
- Easy to derive channel activation ratios and input “processing” signals (for the oracle…)
- Post layout code exec. Two programs:
  - Matrix multiply exercises mostly RF-DMEM loops
  - Extraction Sort activates mainly CU-RF-ALU branch loops

Example: small CPU



- Many “tight” loops
- Easy to derive channel activation ratios and input “processing” signals (for the oracle…)
- Post layout code exec. Two programs:
  - Matrix multiply exercises mostly RF-DMEM loops
  - Extraction Sort activates mainly CU-RF-ALU branch loops

Example: small CPU



- Many “tight” loops
- Easy to derive channel activation ratios and input “processing” signals (for the oracle…)
- Post layout code exec. Two programs:
  - Matrix multiply exercises mostly RF-DMEM loops
  - Extraction Sort activates mainly CU-RF-ALU branch loops

Example: small CPU

- Example VHDL code: input “processing” companion signals in **Register File**

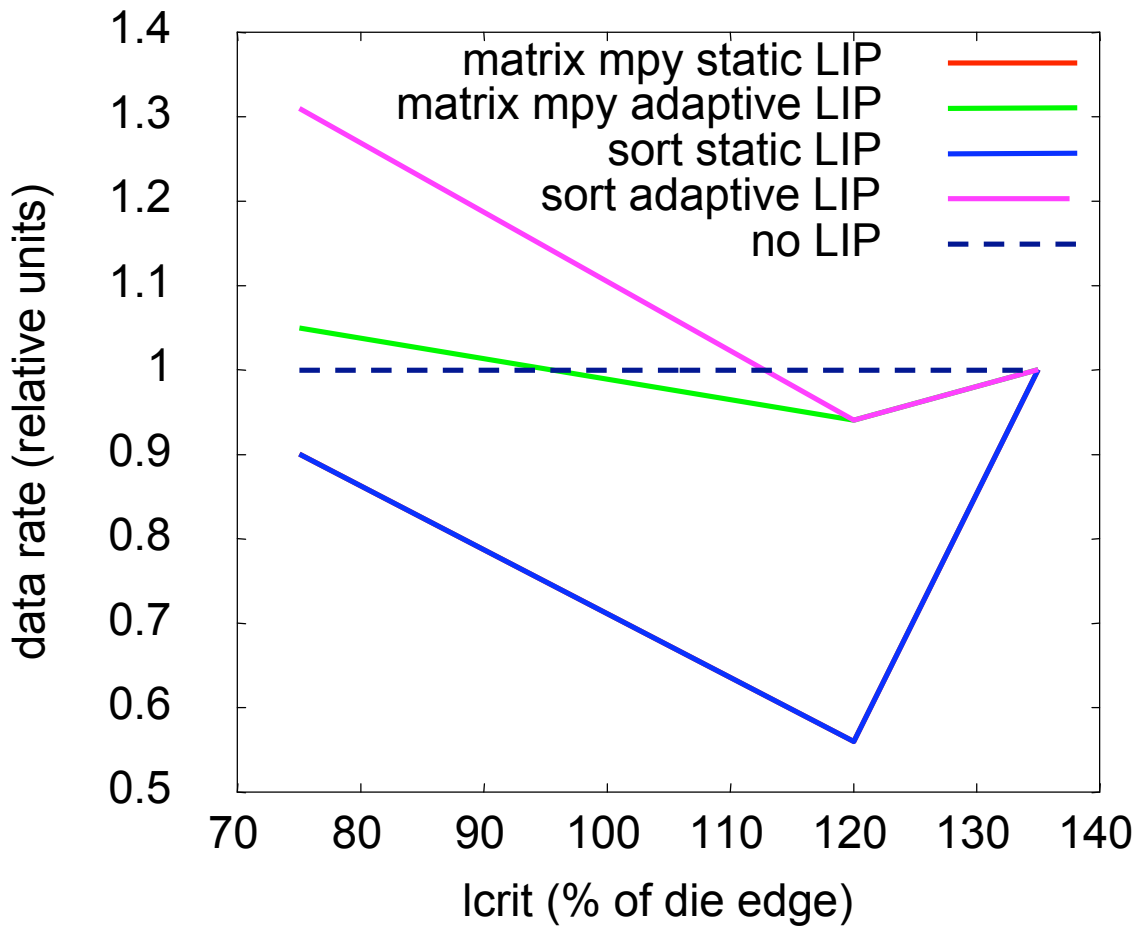
```
entity RF is
   ...
   rf_src1     : in   UNSIGNED (4 downto 0);  -- source reg 1 address
   p_rf_src1   : out STD_LOGIC;               -- source reg 1 PROCESSING bit
   rf_src2     : in   UNSIGNED (4 downto 0);  -- source reg 2 address
   p_rf_src2   : out STD_LOGIC;               -- source reg 2 PROCESSING bit
   rf_des1     : in   UNSIGNED (4 downto 0);  -- dest reg 1 address
   p_rf_des1   : out STD_LOGIC;               -- dest reg 1 PROCESSING bit
   ...
   process(rd, wr, from_mem)
   begin
   if( rd = '1' ) then
        p_rf_src1 <='1'; -- read cycle: addresses of source
        p_rf_src2 <='1'; -- registers have to be processed!

   if( wr = '1' ) then
        p_rf_des1 <='1'; -- write cycle: address of dest
                         -- register has to be processed!
   ...
```

Example: small CPU

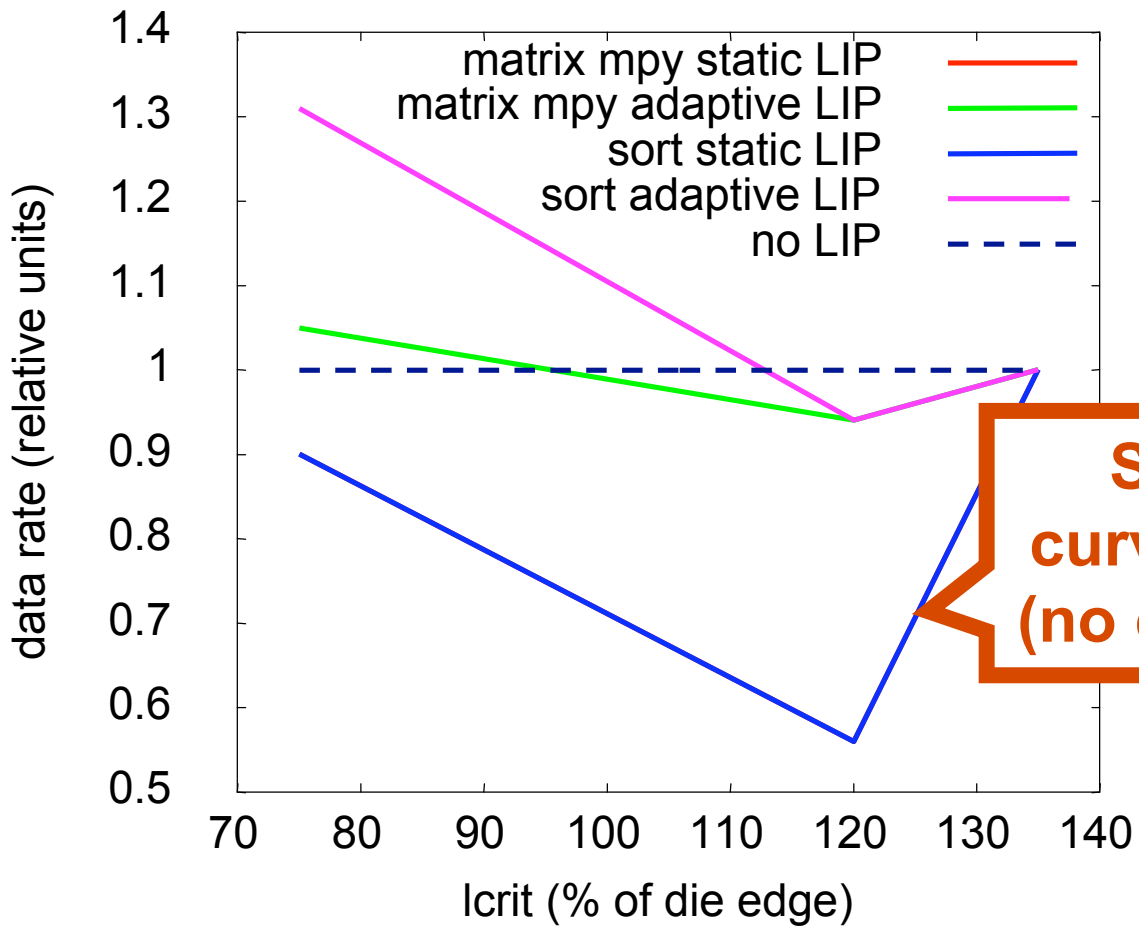
- Example VHDL code: input “processing” companion signals in **ALU**

```
entity ALU is
   ...
   op_code : in  UNSIGNED (3 downto 0);
   src_1   : in  UNSIGNED (15 downto 0);   -- src_1 input
   p_src_1 : out STD_LOGIC;                -- src_1 PROCESSING bit
   src_2   : in  UNSIGNED (15 downto 0);   -- src_2 input
   p_src_2 : out STD_LOGIC;                -- src_2 PROCESSING bit
   ...
   process(op_code)
   begin
   case op_code is -- switch based on opcode
        when OP_IS_ADD =>          -- when ADDITION
                p_src_1 <= '1';    -- process both input src_1 and
                p_src_2 <= '1';    -- input src_2
        when OP_IS_OR =>           -- when logic OR
                p_src_1 <= '1';    -- process both input src_1 and
                p_src_2 <= '1';    -- input src_2
        when OP_IS_RL =>           -- when ROTATE LEFT
                p_src_1 <= '1';    -- process only input src_1
   ...
```

Example: small CPU

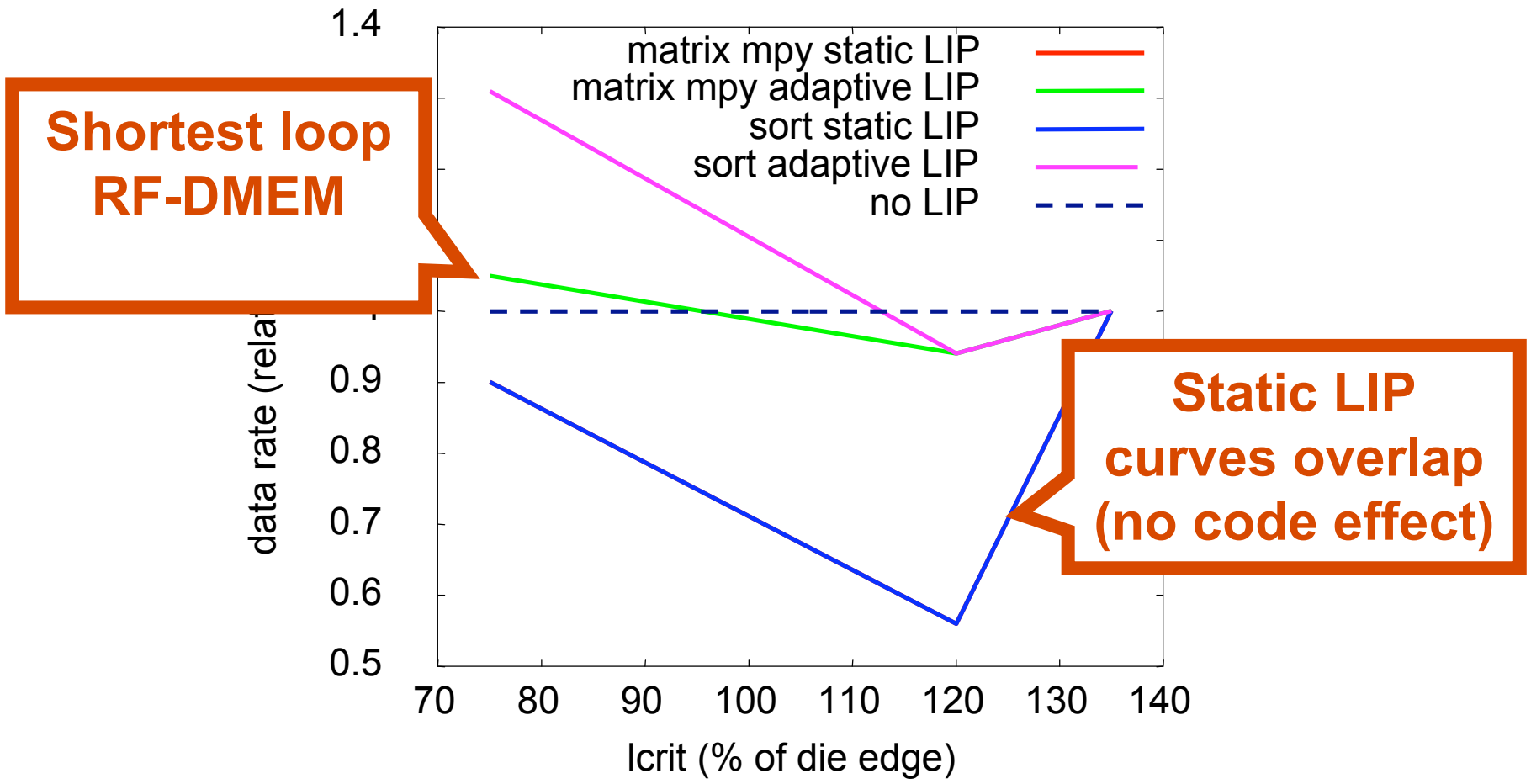


Example: small CPU



Static LIP curves overlap (no code effect)

Example: small CPU



Discussion

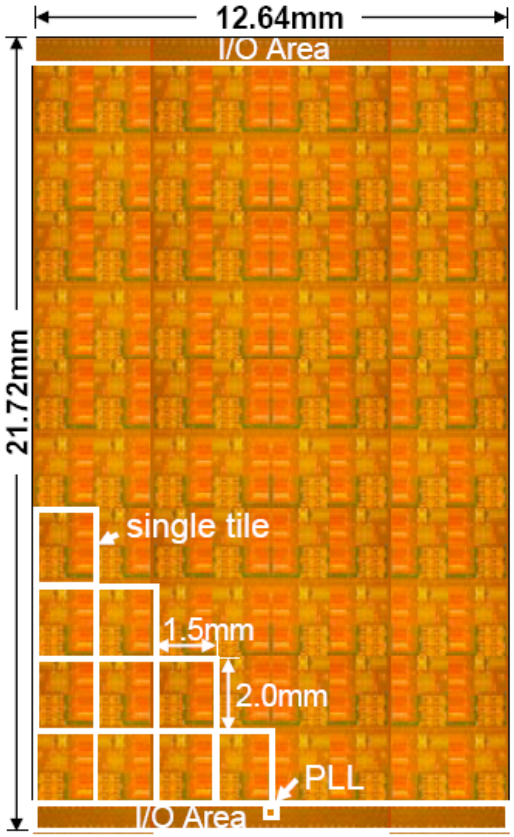
- Floorplan results confirm that static LIPs advantage emerges only in few cases
  - Loose loops with latencies≠0 only in few edges
  - Tight loops must be zero latency
  - Otherwise, slowing down computation to meet wire delay is a better option
- Adaptive LIPs alleviate these limitations
  - As always, there's no such a thing as a free lunch…
  - wrapper area cost and engineering cost of building processing signals or wrapper's FSM)
  - In any case advantages are benchmark dependent
- Problem: *are we benchmarking the right way?*

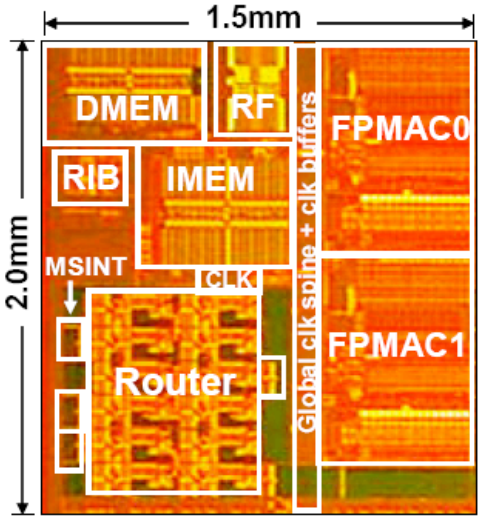
Discussion

- Q: What type and what size for the elementary logic block ("Carloni's pearl")
  - Q': what is the size of a clock domain
- A: Prospectively looking, SoC will look more as array of regular fabrics
  - e.g. many simple processor cores paired with memories and few specialized hw accelerators
  - A'. the clock domain is the "tile"
- Communication between cores will be *explicit*
  - Latency insensitive protocols will be *natively* adaptive

“Tile based” design

- 80 cores connected though NoC [Intel07]
- Global mesochronous 4GHz clocking
- Cores communicate only with tile routers
- Tile routers are connected through p2p links
- Making links latency insensitive is easy!





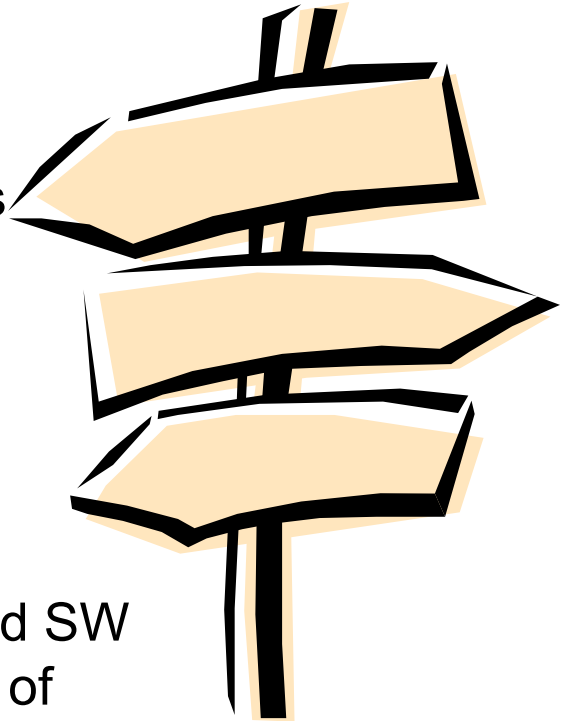
Technology	65nm CMOS Process
Interconnect	1 poly, 8 metal (Cu)
Transistors	100 Million
Die Area	275mm²
Tile area	3mm²
Package	1248 pin LGA, 14 layers, 343 signal pins

Outline

- ITRS roadmap calls for innovative design
- Static vs. Adaptive Latency Insensitive Protocols
- Practical issues
- Latency & throughput-aware floorplanning
- Results and discussion
- Future directions and conclusions

Future directions

- Exploring the relation between “new” models of computation and the GALS paradigm



- New benchmarks
  - Right mix of HW and SW
  - Global assessment of various design choices through accepted metrics (and their sensitivity)

- GALS physical design
  - Performance modeling and inclusion in floorplan tool
  - Simultaneous P&R of repeaters and mixed-clock RS

Conclusions

References

[Carloni99] L. P. Carloni et al., A methodology for correct-by-construction latency insensitive design. In *Proc. ICCAD'99*

[Carloni00] L.P. Carloni and A. Sangiovanni-Vincentelli, Performance Analysis and Optimization of Latency Insensitive Systems, *Proc. DAC'00*

[Carloni01] L.P. Carloni *et al.* Theory of Latency-Insensitive Design, IEEE TCAD, vol. 20, No. 9, Sept. 2001, pp. 1059-1076.

[DAC04] M.R. Casu and L. Macchiarulo, A New Approach to Latency Insensitive Design, *Proc. DAC'04*

[Nowick01] T. Chelcea and S.M. Nowick, Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols, *Proc. DAC'01*

[Singh03] M. Singh and M. Theobald, Generalized Latency Insensitive Systems for Single-Clock and Multi-Clock Architectures, *Proc. FMGALS'03*

[Singh05] An Architecture and a Wrapper Synthesis Approach for Multi-Clock Latency-Insensitive Systems, *Proc. ICCAD'05*

[Bomel05] P. Bomel *et al.*, High-Level Synthesis in High-level synthesis in latency insensitive system methodology, *Proc. DSD'05*

[DATE05] M.R. Casu and L. Macchiarulo, A New System Design Methodology forWire Pipelined SoC, Proc. DATE'05

References

[TCAD05] M.R. Casu and L. Macchiarulo, Throughput-Driven Floorplanning With Wire Pipelining, IEEE TCAD, May 2005.

[Markov03] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: Enabling hierarchical design,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[TCAD06] M.R. Casu and L. Macchiarulo, Floorplanning With Wire Pipelining in Adaptive Communication Channels, IEEE TCAD, Dec. 2006.

[Ekpaniapong04] M. Ekpanyapong *et al.*, “Profile-guided microarchitectural floorplanning for deep-submicron processor design,” in *Proc. DAC’*04

[Long04] C. Long *et al.*, “Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects,” in *Proc. DAC’*04

[Nookala05] V. Nookala *et al.*, “Microarchitectural-aware floorplanning using a statistical design of experiments approach,” in *Proc. DAC’*05

[NTT96] T. Kondo *et al.*, “Two-Chip MPEG2 Video Encoder”, IEEE Micro, April ‘96

[NTT99] T. Kondo *et al.*, “Superenc: MPEG-2 Video Encoder Chip”, IEEE Micro, Jul-Aug. 99

[INTEL07] N. Borkar *et al.*, “An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS”, in *Proc. ISSCC’07*

Theoretical issues

- Is an adaptive LIP system *equivalent* to the original (i.e. no LIP) system and to its static variant?
- Definition of latency equivalence requires some formalism: Tagged signal model.
- Suppose a system with M channels. Original system behavior in $[t_1,t_N]$:
  - $(v_1^{(i)},t_1),(v_2^{(i)},t_2),\ldots,(v_N^{(i)},t_N)$, **i = 1 : M**
- LIP system behavior in $[t_1,t_N]$:
  - $(v_1^{(i)},t_1),\tau,(v_2^{(i)},t_3),\ldots,(v_K^{(i)},t_N)$, **i = 1 : M, K ≤ N**

Latency equivalence

- After [Carloni01]:
  - *"Two signals are latency equivalent if they present the same sequence of informative events, i.e., they are identical except for different delays between two successive informative events."*
- n-equivalence definition:
  - If, in a given time interval $[0, t_N]$, $\exists n$ s.t. every signal in LIP system has at least n valid events equal and ordered as in the original case, said system is said "n-equivalent" or "equivalent of degree n."
- n-equivalence $\forall n$ coincident with Carloni's equivalence

Two steps to equivalence

- Evolutionary proof approach:
- Step A: equivalence between no LIP system and static LIP system
- Step B: equivalence between no LIP system and adaptive LIP system

Step A (1/2)

- Features of static LIPs (abstract) wrappers
  1. τ-filtered inputs are buffered in fifos (possibly of zero depth)
  2. A synchronizer keeps track of the current tag (local tag counter) and, as soon as all inputs with the same tag are available:
     a) dispatches them to the internal process and remove them from the fifos;
     b) if at least one of the inputs is not available, i.e. it does not have the current tag, the process is *stalled*.
  3. If at least one input fifo is full, a back-pressure signal called *stop* is sent back to that input channel.
  4. If a *stop* is received from one of the output channels on a valid computation (i.e. when the process is not stalled), the wrapper stalls the process for the next cycle and propagates the stop to all inputs. If the stop is received on a τ value, the stop is *absorbed* and will not be back-propagated.
  5. In correspondence with the stall, τ is sent to all outputs.

Step A (2/2)

- It is possible to prove that a LIP system with wrappers as of step A is equivalent to the original system
- Need Relay Stations to hold data in case of back-pressure
- Wrappers as well as RS implement "stop absorption"
  - back-pressure signals are absorbed when τ (void) events are pipelined and are not back-propagated
- Last property is the key to prove j-equivalence (by induction):
  - sooner or later tags labeled "j" already computed will reach their destination. Moreover, output stops cannot stall computation indefinitely (a stop received on a stall event will be ignored). Therefore inputs "j" will eventually enable computation of "j+1" tags.
- j-equivalence can be proved $\forall j$, $\Rightarrow$ equivalence
- No actual need for "tag labels" nor for tag counters
  - valid/stop signals suffice (from abstract to real…)

Step B (1/2)

- Adaptive LIPs wrappers' features
- An *oracle* decides which inputs will be actually needed for the next computation.
- Properties 1 to 5 as before

if the subset of inputs *required by the oracle* are present, i.e. they have the same tag as the current local tag, the computation is triggered and the fifos updated.

The synchronizer discards all inputs whose tag is smaller than the current value (tags "older" than local current tag).

Step B (2/2)

- Again, it is possible to prove equivalence
- Differently from the static case, simple check of validity (i.e.$\neq \tau$) is not sufficient:
  - wrapper should be able to identify and discard “older values” from inactive inputs
  - if tags are not used (for practical reasons) and validity signals are employed, it is necessary to count *how many tags have been discarded*.
  - thanks to strong ordering, counting the number of valid events is equivalent to keeping track of their tags

Outline of possible design flow

- System is developed using standard methodologies
  - possibly, blocks inputs are associated with processing signals
- Blocks are encapsulated with wrappers
  - with or w/o oracle
- Logic synthesis provides area estimates and clock frequencies for each block
  - global interconnects ignored
- Floorplan gives estimates of global wires length
  - highlights new critical path if max wire delay exceeded
  - estimates performance if LIPs are used
  - allows evaluating data-rate/throughput tradeoffs
- Post-floorplan netlist includes RS locations
  - allows new system simulations *back-annotated* with real latencies

Floorplanning in Adaptive LIPs

- Problem: The worst loop cannot be statically determined
  - depends on communication profile which varies during computation
- Enriching floorplan cost functions with full profile information is impractical. We use *α* [TCAD06]*:*
  - *logical time fraction in which a channel is active*
- *Logical time* in terms of logic computation steps
  - physical design effects ignored: no need to iterate between floorplan and channel back-annotation, as opposed to [Ekpaniapong04][Long04][Nookala05]
  - can be assessed through a single profiling experiment (or better, averaging significant profiling)
- Assumption: activation ratios statistically independent