K-periodically Routed (extended) Marked/Event Graphs

Anthony Coadou

Julien Boucaron

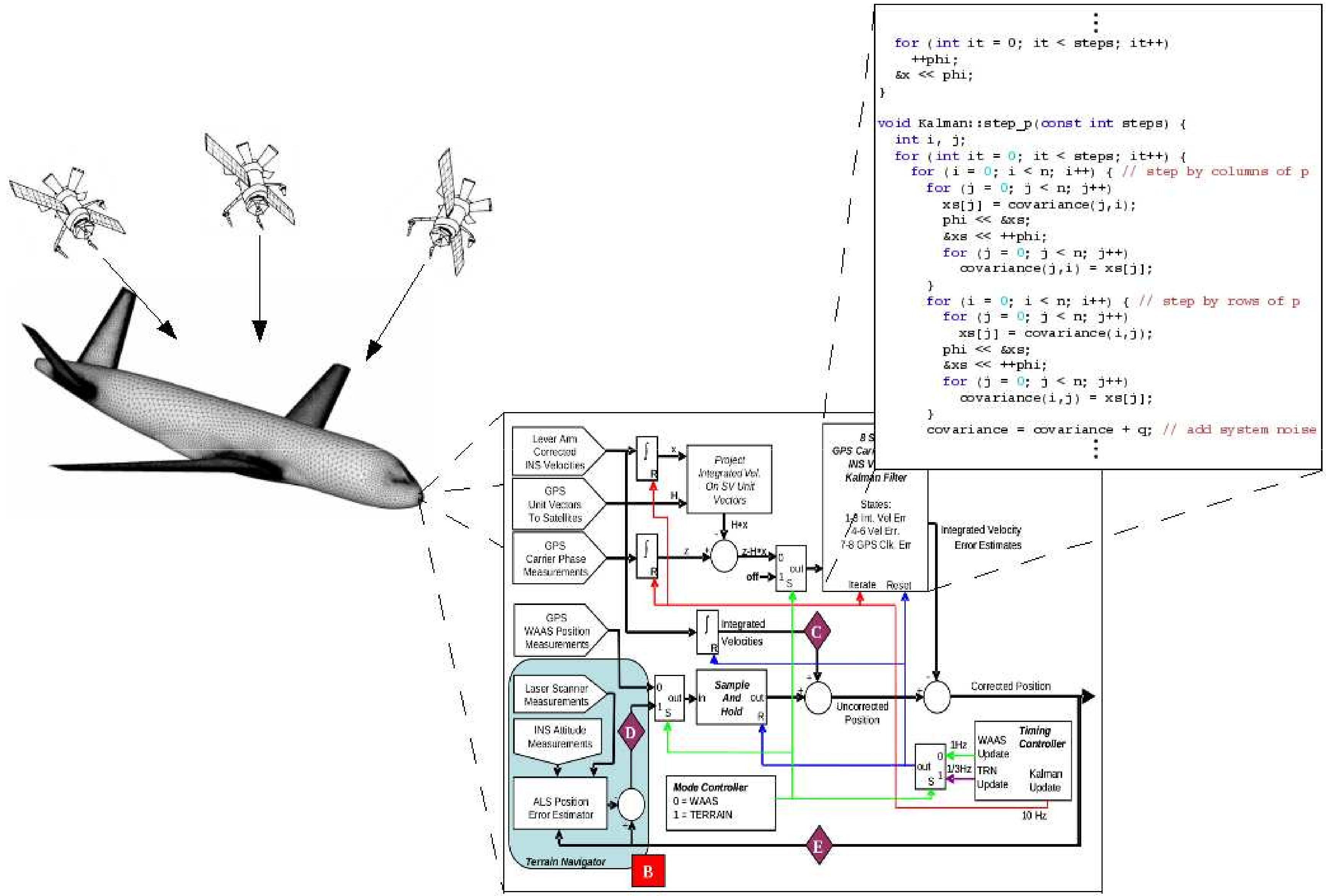
Robert de Simone

Benoît Ferrero

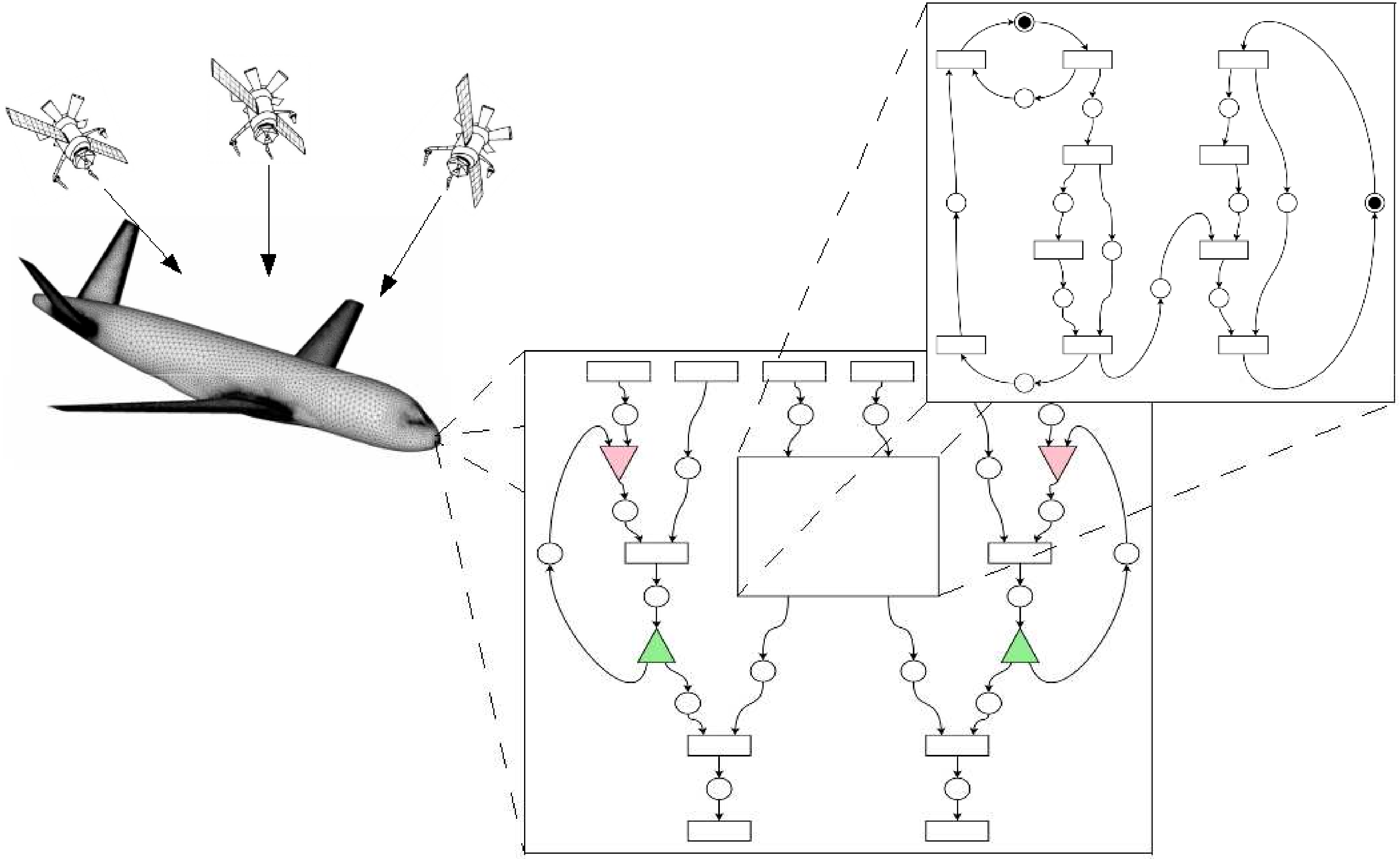
Introduction

- Modern architectures: multicore, networks of processors, with on-chip network interconnects
- Many modern algorithms (multimedia) described also as streaming dataflow functional networks
- Compilation goal: adjust the application to the architecture (mapping)

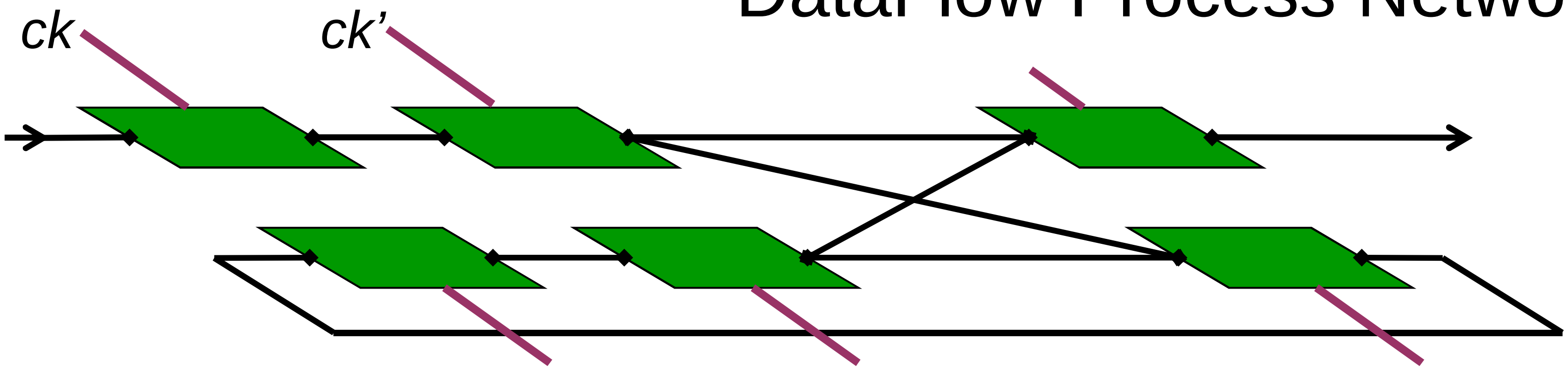
Formal models and methods dedicated to this (contemporary) problem ?



for (int it = 0; it < steps; it++)
++phi;
&x << phi;
}
void Kalman::step_p(const int steps) {
int i, j;
for (int it = 0; it < steps; it++) {
for (i = 0; i < n; i++) { // step by columns of p
for (j = 0; j < n; j++)
xs[j] = covariance(j,i);
phi << &xs;
&xs << ++phi;
for (j = 0; j < n; j++)
covariance(j,i) = xs[j];
}
for (i = 0; i < n; i++) { // step by rows of p
for (j = 0; j < n; j++)
xs[j] = covariance(i,j);
phi << &xs;
&xs << ++phi;
for (j = 0; j < n; j++)
covariance(i,j) = xs[j];
}
covariance = covariance + q; // add system noise
Lever Arm Corrected INS Velocities
GPS Unit Vectors To Satellites
GPS Carrier Phase Measurements
GPS WAAS Position Measurements
Laser Scanner Measurements
INS Altitude Measurements
ALS Position Error Estimator
Terrain Navigator
Project Integrated Vel. On SV Unit Vectors
Kalman Filter
States:
1-3 Int. Vel Err
4-6 Vel Err.
7-8 GPS Clk. Err
Iterate
Reset
Integrated Velocity Error Estimates
Integrated Velocities
Sample And Hold
Uncorrected Position
Corrected Position
Mode Controller
0 = WAAS
1 = TERRAIN
Timing Controller
WAAS Update
TRN Update
Kalman Update
1Hz
1/3Hz
10 Hz
z-H*x
H*x
off
B
C
D
E



DataFlow Process Networks



- Original intent (first specification semantics, self-timed):
  - –Computation blocks executed/performed as soon as they get enough input data
- But computers usually do not work that way (asynchronous circuits?),
  - –so control in the form of activation clocks need to be introduced

→ second, scheduled semantics

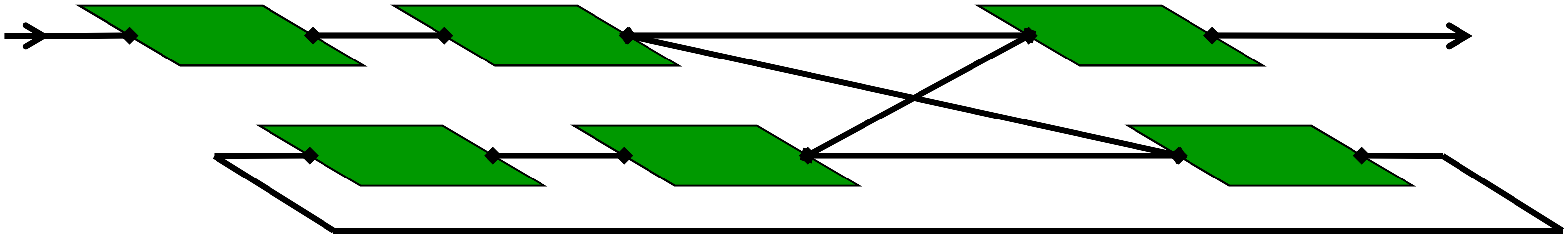
•

Intermediate stages, between self-timed/asynchronous and totally timed (strict order) → **Logical time** (several time threads partially related).

–**optimize** buffer sizes, **optimize** latencies,

•One main property: Conflict Freeness

DataFlow Process Networks



- Many variants:
  - Event Graphs (Petri Net subclass, 1- 1 channels for places)
  - SDF extension (packet sizes for consumption/production, equalization issue)
  - BDF, CSDF (introducing control variants and switches)
  - Kahn PNs (implicit sequential algorithms in computation nodes)
  - Internal computations with similar features: nested loops, polyhedral models
  - Use in HLS: Pico Express/Synfora, Tensilica
  - Historically, several INRIA efforts (cf. other talks)

Process Networks

- Purely data-flow models (no control switches): **Event/Marked Graphs, SDF**

  Main issue: **scheduling**
  → from self-timed to rigidly clocked (possibly multiclock)
  → regular (ultimately k-periodic) ASAP runs
  -> optimization according to various criteria

- Models allowing (*conflict-free*) control switches: **BDF, CycloStaticDF, Kahn PN**

  Additional issues: **routing**
  → optimization: from ideal communication structures to specific network topologies

Conflict–freeness

→ Forbids choices based on input token availability.
→ entails ***monotonicity*** (Kahn PNs), ***confluence*** (CCS), ***latency insensitivity***
→ all runs are just different ordering of the same partilly ordered trace

→ **Not true** of general Petri Nets, Synchronous Languages (instantaneous preemption)

Process Network based tools

- Ptolemy (UC Berkeley)
- Synfora Pico Express
- StreamIt
- AutoPilot
- SDF3
- Compaan
- PN
- K-Passa (EPI Aoste)
- Gaspard2/Array-OL (EPI DaRT)

link with polyedra parallel compiling techniques ???

Determinism ? Conflict-freeness !

•It states that an enabled computation must eventually be performed

Counterexamples:

- Choice between input guards (channels) in CSP
- Absence testing in Esterel (abort await S do P when T); interrupts
- Petri Nets conflicts on a shared places (so not a channel)

•It provides:

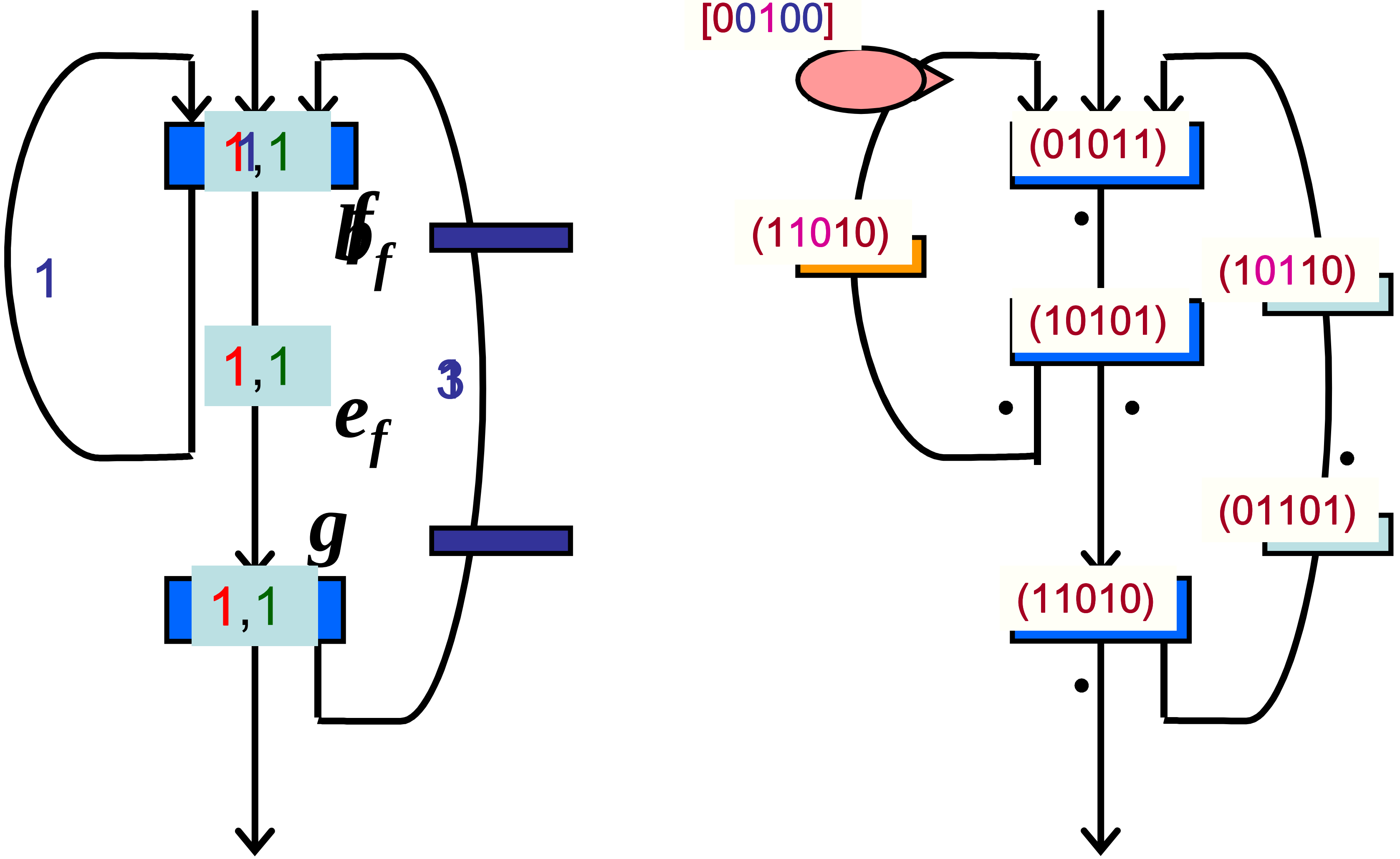
- Confluence (in Process Algebras): diamond property, independence
- Monotonicity (in Kahn Proces Networks), compositional
- *Endo/isosochrony*
- **Latency Insensitivity**: elastic design, important for modular SoC design

•In essence, all potential behaviors are just different schedules of same functional one, partially ordered

- Many theoretical results on static optimal scheduling and buffer sizing

(k-periodic modulo scheduling, max-plus algebra, N-synchronous models)

Ultimately k-periodoc scheduling



Role of KREGs

- **Data-flow streaming** computations
- **Interconnect networks** with (predictible, static) routing
- Generated from **nested loop programs**, **NoC**s, **Kahn networks**

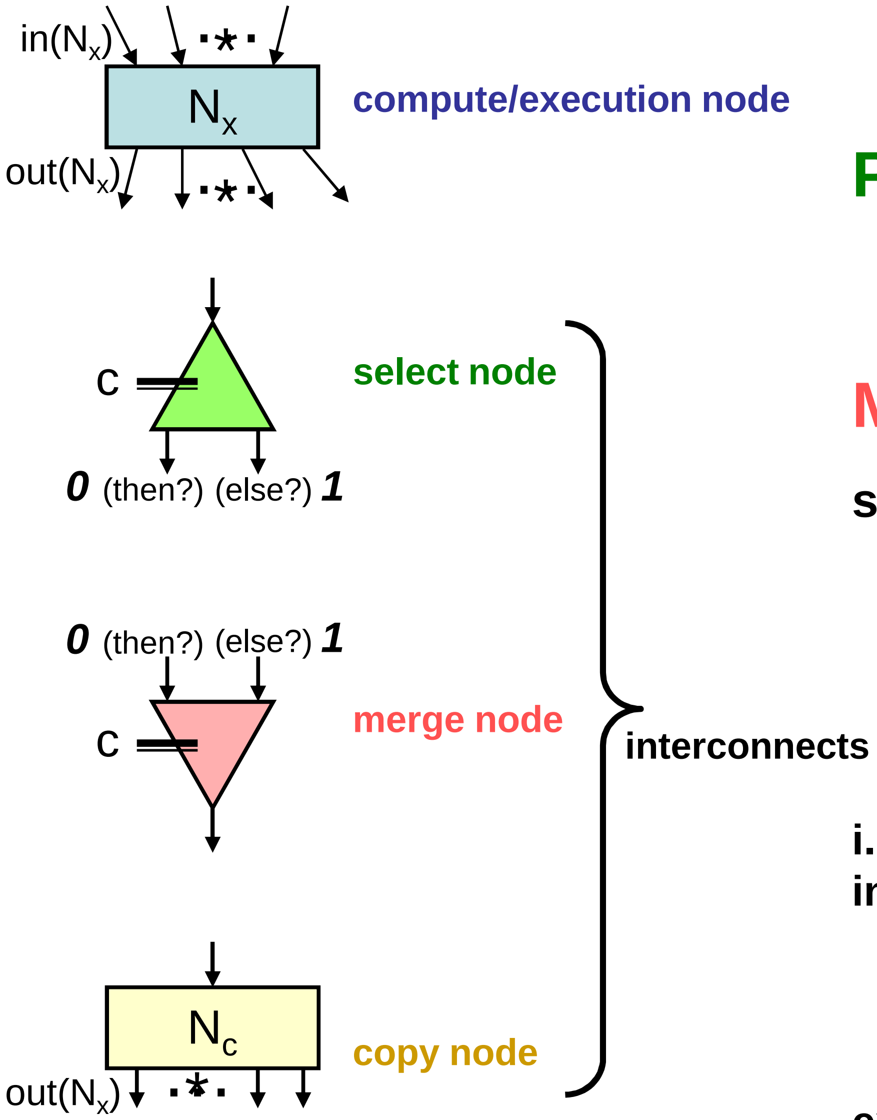
Issues

- **liveness** (absence of deadlocks)
- **safety** (finite buffering of interconnect channels)
- **transformations and optimality** of routing and network topology

depend on:

- precise mathematical semantic formulation (firing rule)
- initial (and latter) **token allocations**

→ **topic of this talk**



Process network tradition

Main feature: new

switching conditions c are

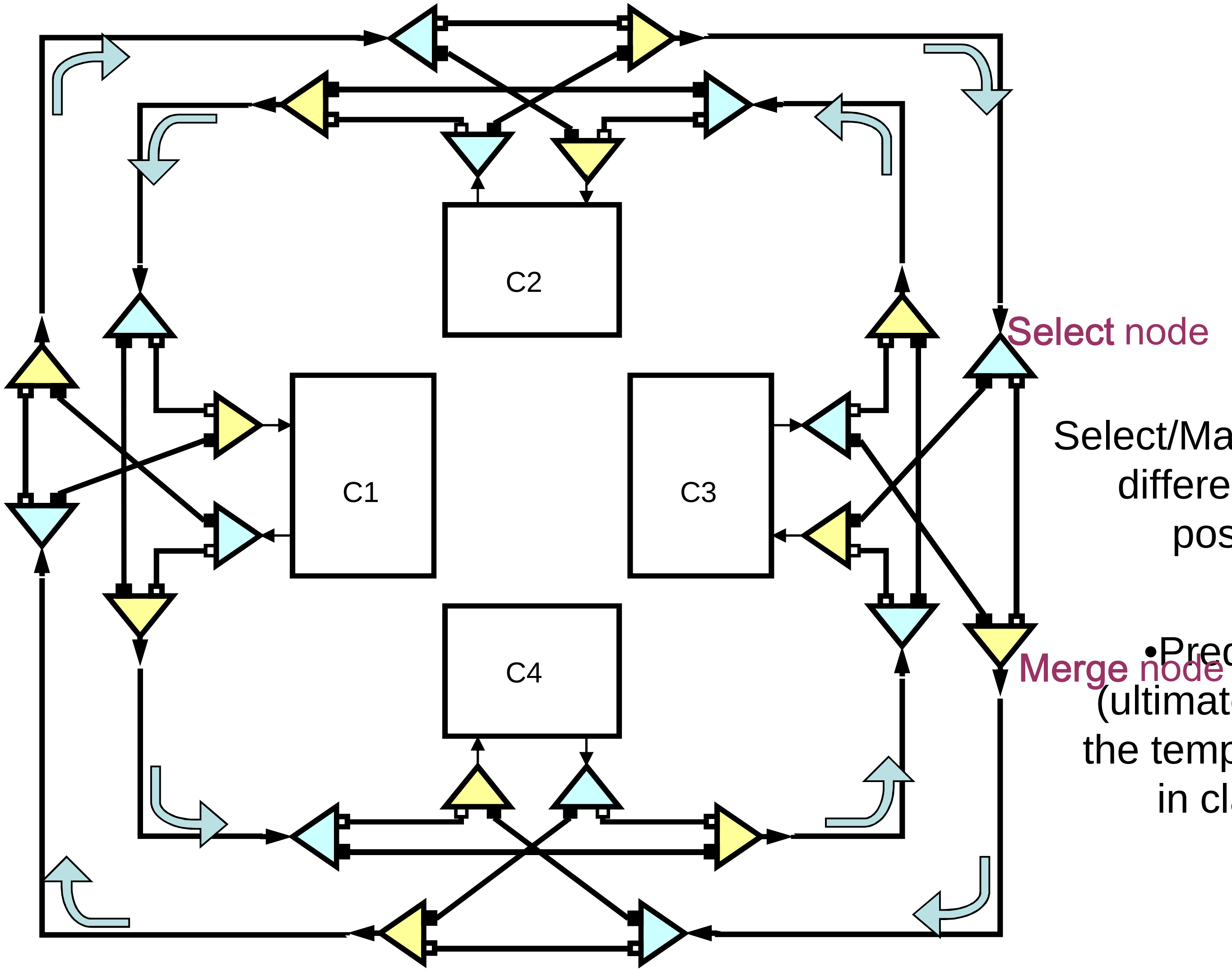
- **explicitely computed (off-line)**
- **ultimately k-periodic**

i.e., c is of the form $u.(v)^{\omega}$ with u, v binary words indicating switching/routing directions,

- **u initial/transient part**
- **v periodic/steady part**

ex: 1100.(10000000)

Interconnect modeling and optimization

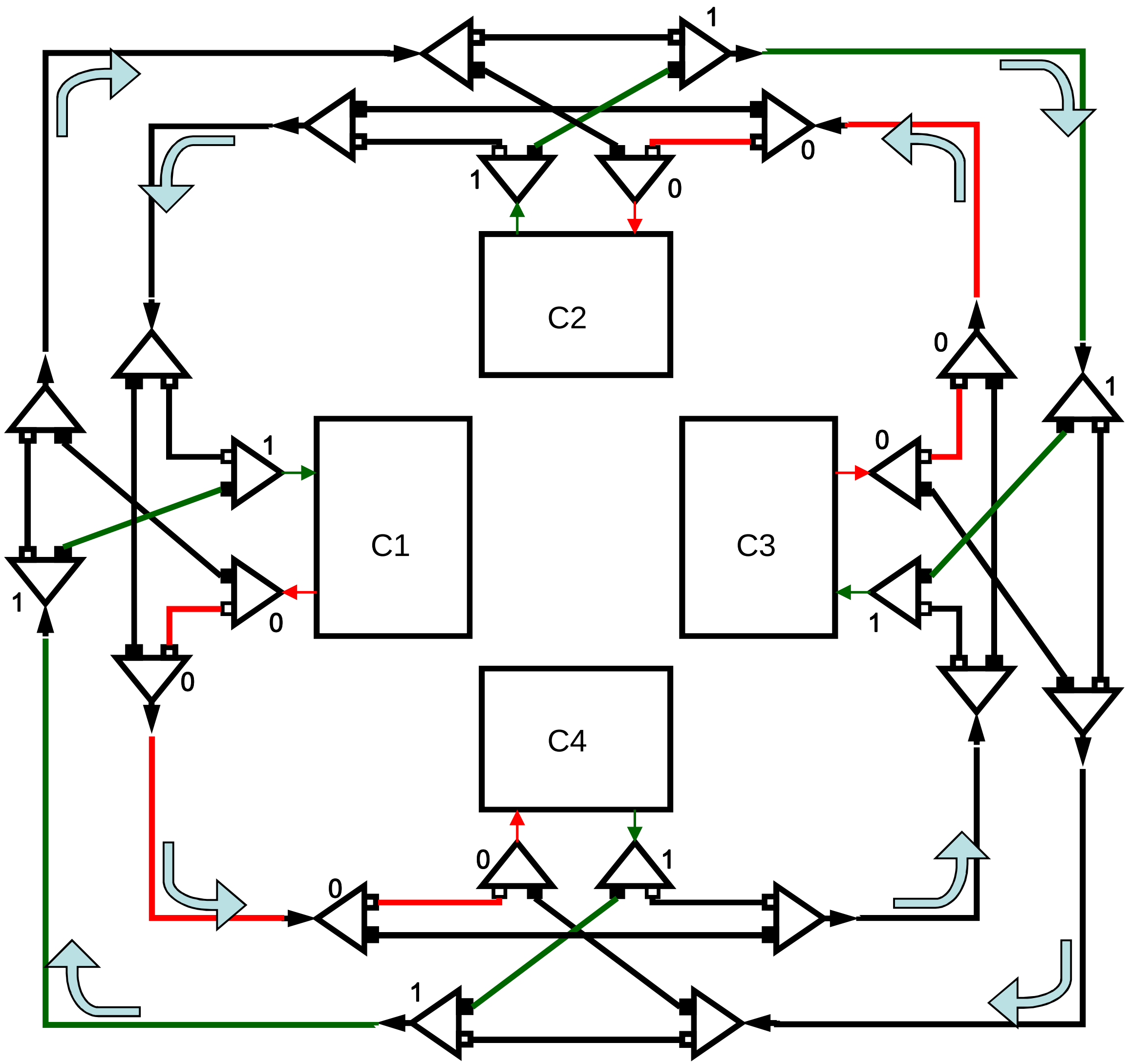


•On-Chip Networks

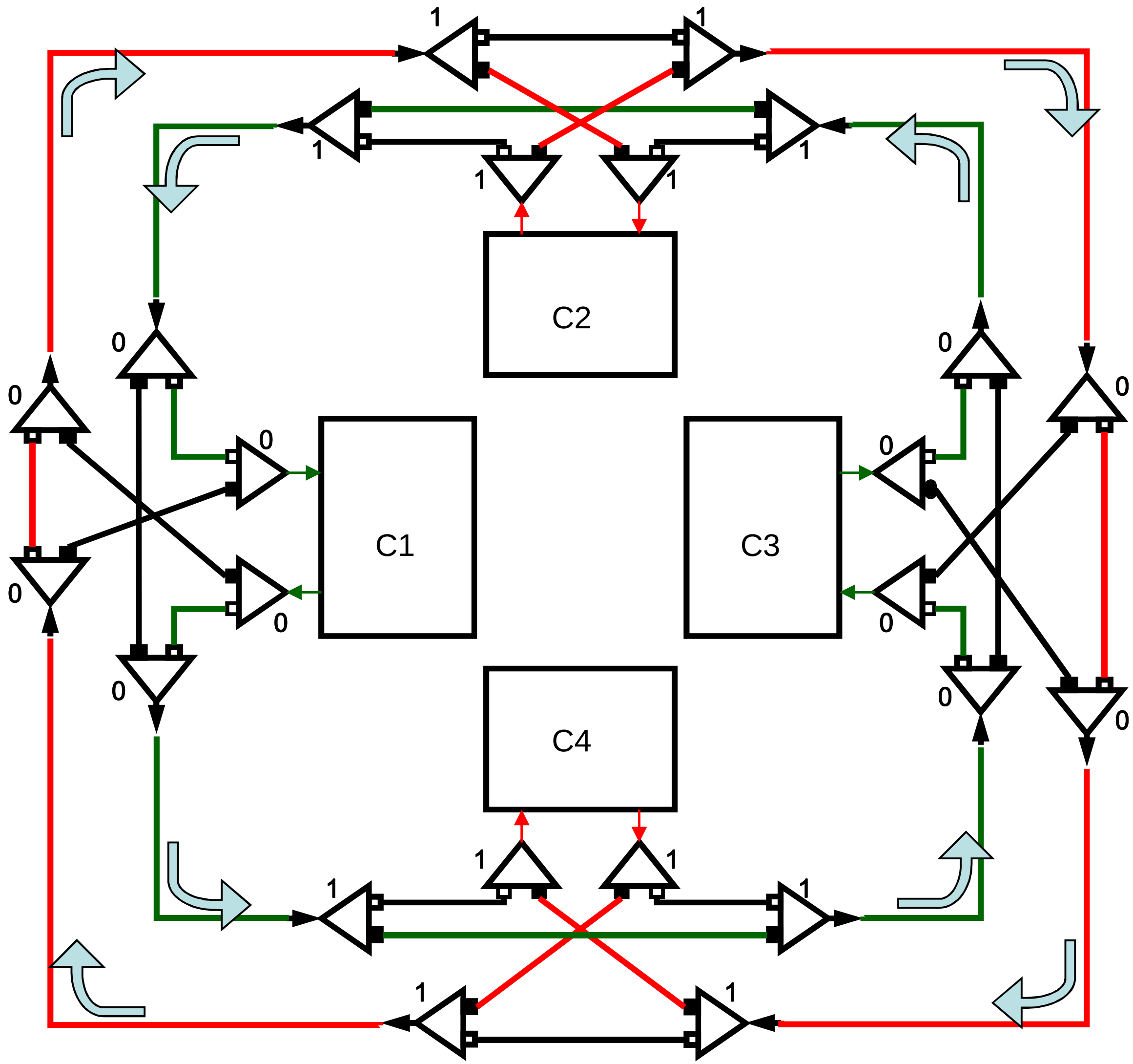
•Switching conditions of Select/Marge nodes can configure different communication paths, possibly overlapping in time

•Predictable routing schemes (ultimately k-periodic) will match the temporal schedules obtained in classical Process Network scheduling theory

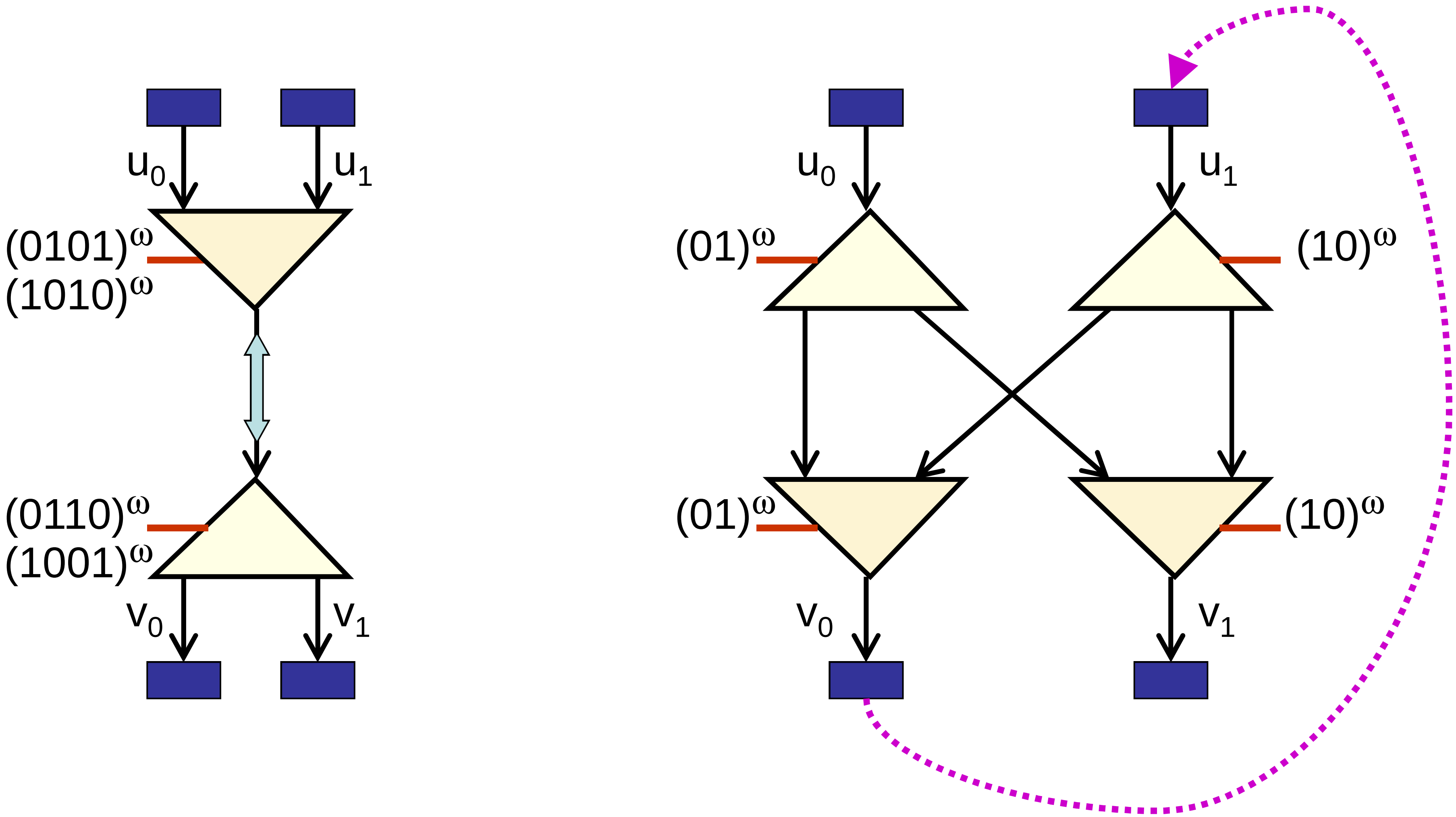
one possible routing configuration



another possible configuration



Interconnect transformation illustrated



new

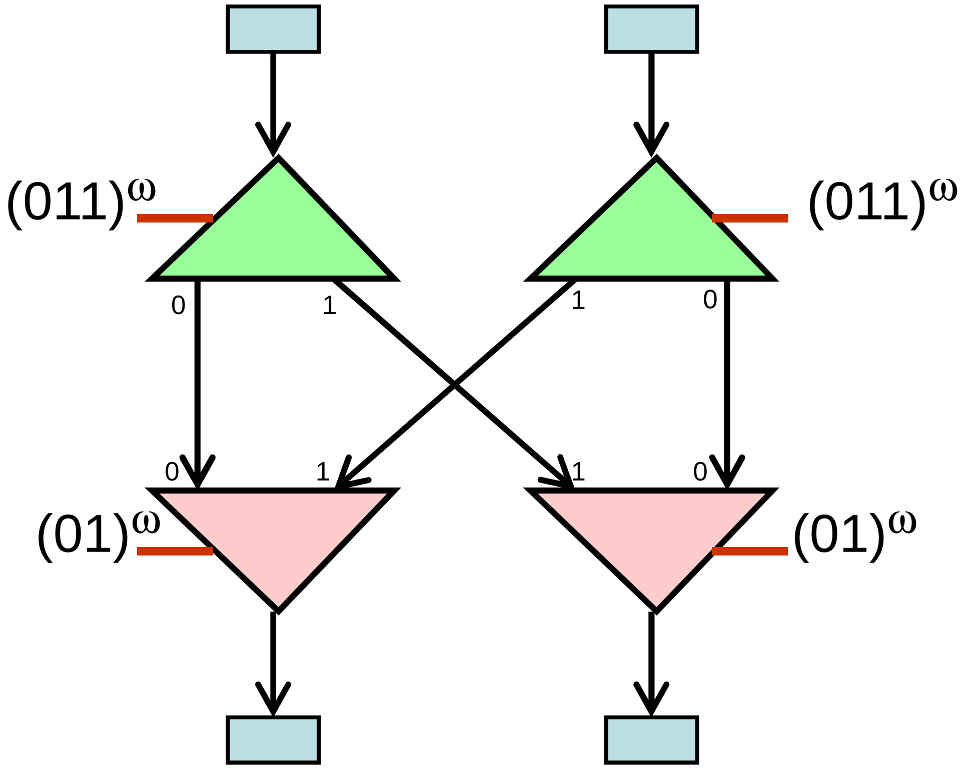
Facts

- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.

  → Through this, **safety can be checked**

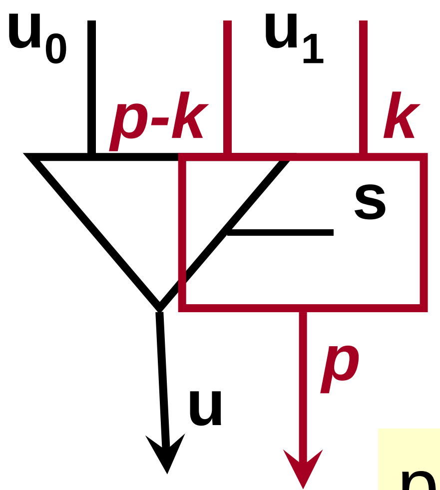
for this presentation we shall ignore copy nodes, and assume that every cycle in the original KRG graph crosses a compute node (not essential restrictions, just make the presentation simpler)

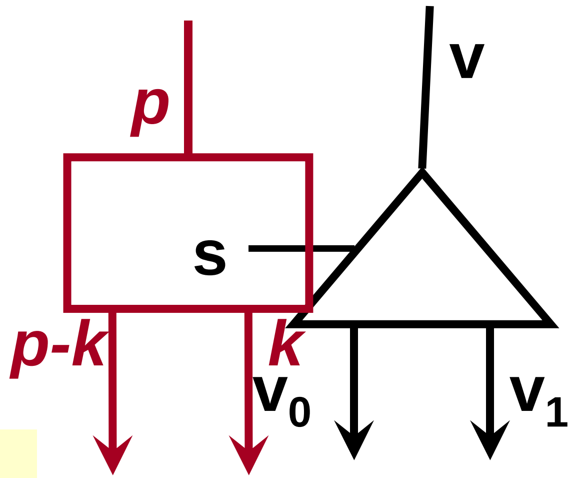
Counter-example: production/consumption mismatch



Tokens accumulate in the “1” channels (to-third production, one half consumption)

SDF abstraction





$p = |s|$
$k = |s|_1$

- Keep the EventGraph parts unchanged (computation/transportation nodes)
- Replace merge/select nodes by weighted nodes averaging consumption production on the period of the switching pattern **s**

new

Facts

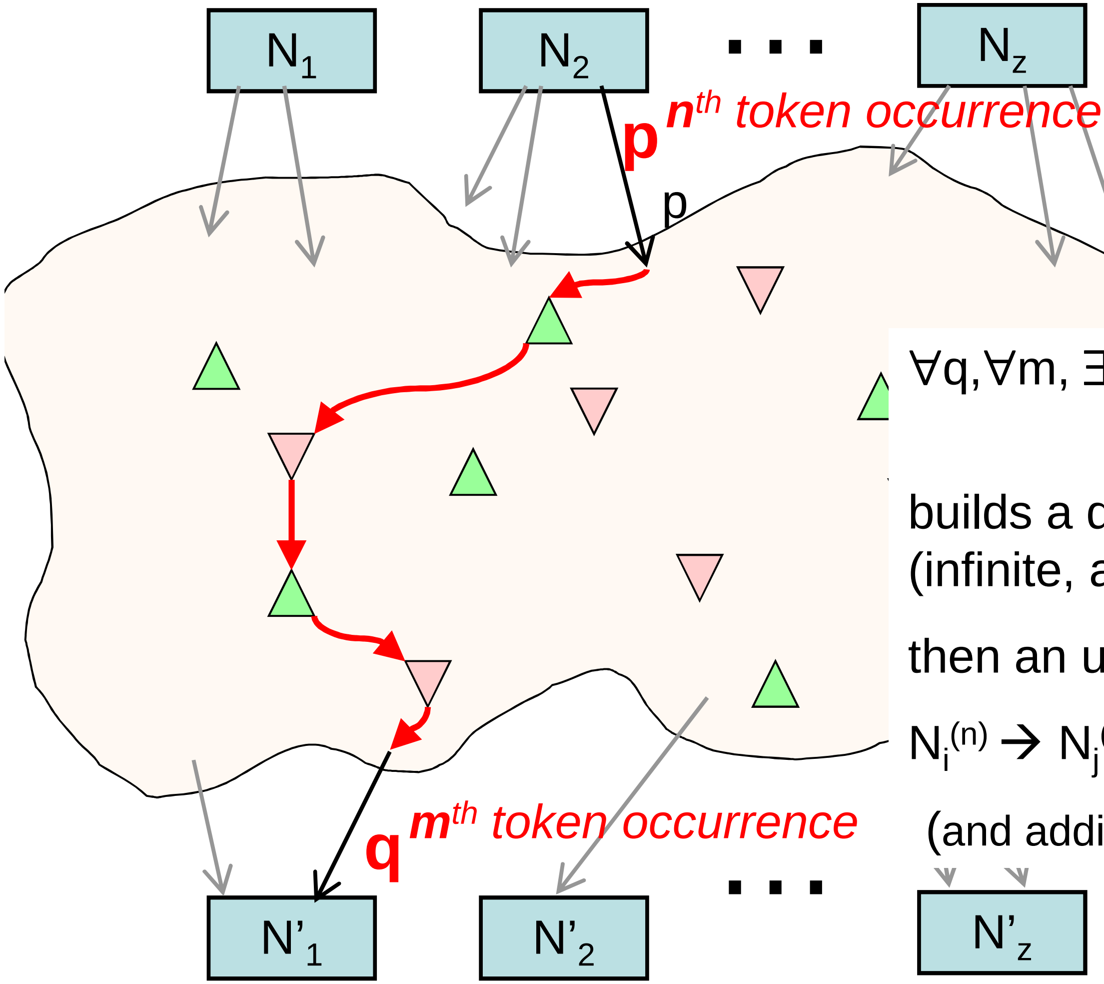
- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.

  → Through this, **safety can be checked**

- **KREG networks can be unfolded into infinite Marked Graphs** expanding computation node occurrences.

  Works by *collecting exhaustively every executions in a single model. Not only structural, depends heavily on the initial markings).*

Tracing the flow of successive tokens



source copy (only outgoing channels)

$\forall q, \forall m, \exists! p, \exists! n$: $p^{(n)}$ causes/becomes $q^{(m)}$

builds a dependency graph: $p^{(n)} > q^{(m)}$ (infinite, acyclic)

then an unfolded Marked graph with

$N_i^{(n)} \rightarrow N_j^{(m)}$ iff $p^{(n)} > q^{(m)}$ for p in $N_i\bullet$, q in $\bullet N_j$

(and additionally $N_i^{(n)} \rightarrow N_i^{(n+1)}$)

target copy (only incoming channels)

new

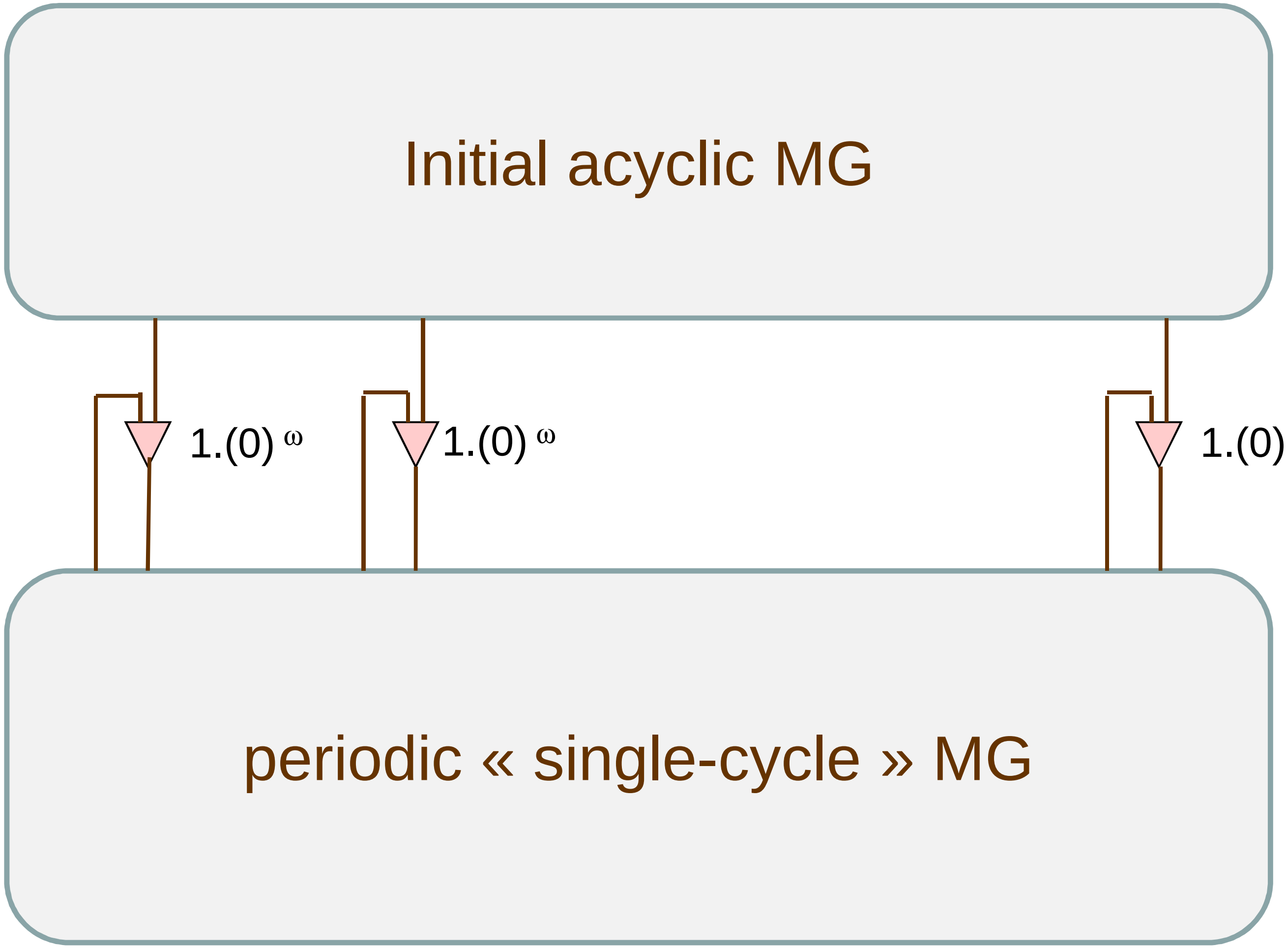
Facts

- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.

  → Through this, **safety can be checked**

- **KREG networks can be unfolded into infinite Marked Graphs**
- Regularity of switching patterns allows to detect cyclicity, to **fold back into finite (quasi)-Marked Graphs**

Resulting quasi-marked graph



new

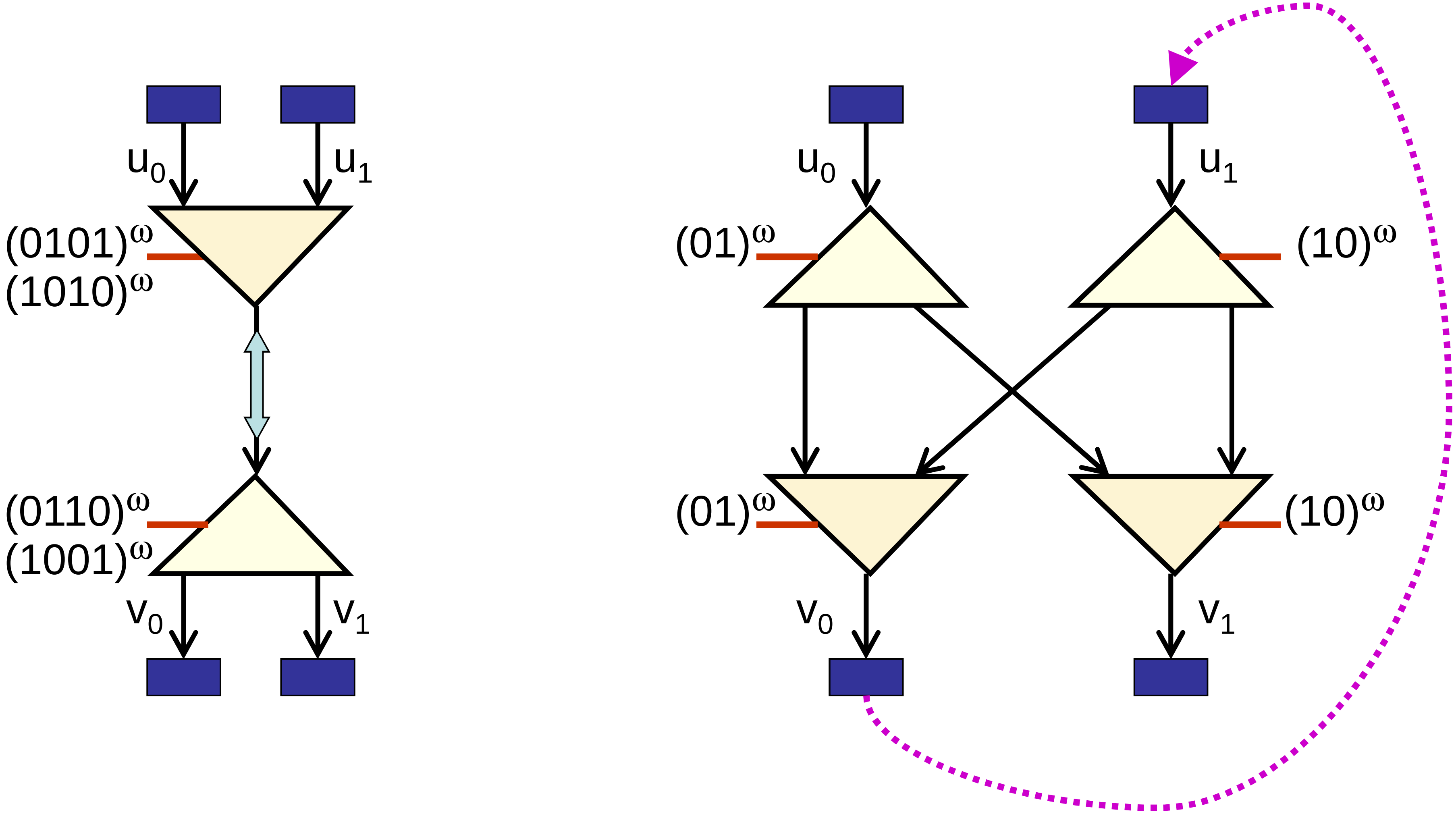
Facts

- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.
  → Through this, **safety can be checked**

- **KREG networks can be unfolded into infinite Marked Graphs**
- Regularity of switching patterns allows to detect cyclicity, to **fold back into finite (quasi)-Marked Graphs**

- **Two notions of equivalences** can be defined, one equates the external token flows, one takes into consideration the internal traffic interleavings in addition **(second one is a congruence)**

Interconnect transformation illustrated



Two equivalences

- First one directly based on isomorphism of expanded Event Graphs

- Second uses a more refined dependency graph, with:

$$p^{(n)} \gg q^{(m)} \quad \text{if} \quad \exists r,i,j, \quad p^{(n)} > r^{(i)} \wedge q^{(m)} > r^{(i+j)}$$

(sequentiality of token traffic in intermediate channels, so that data precedence and interleavings are preserved)

new

Facts

- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.
  → Through this, **safety can be checked**

- **KREG networks can be unfolded into infinite Marked Graphs**
- Regularity of switching patterns allows to detect cyclicity, to **fold back into finite (quasi)-Marked Graphs**

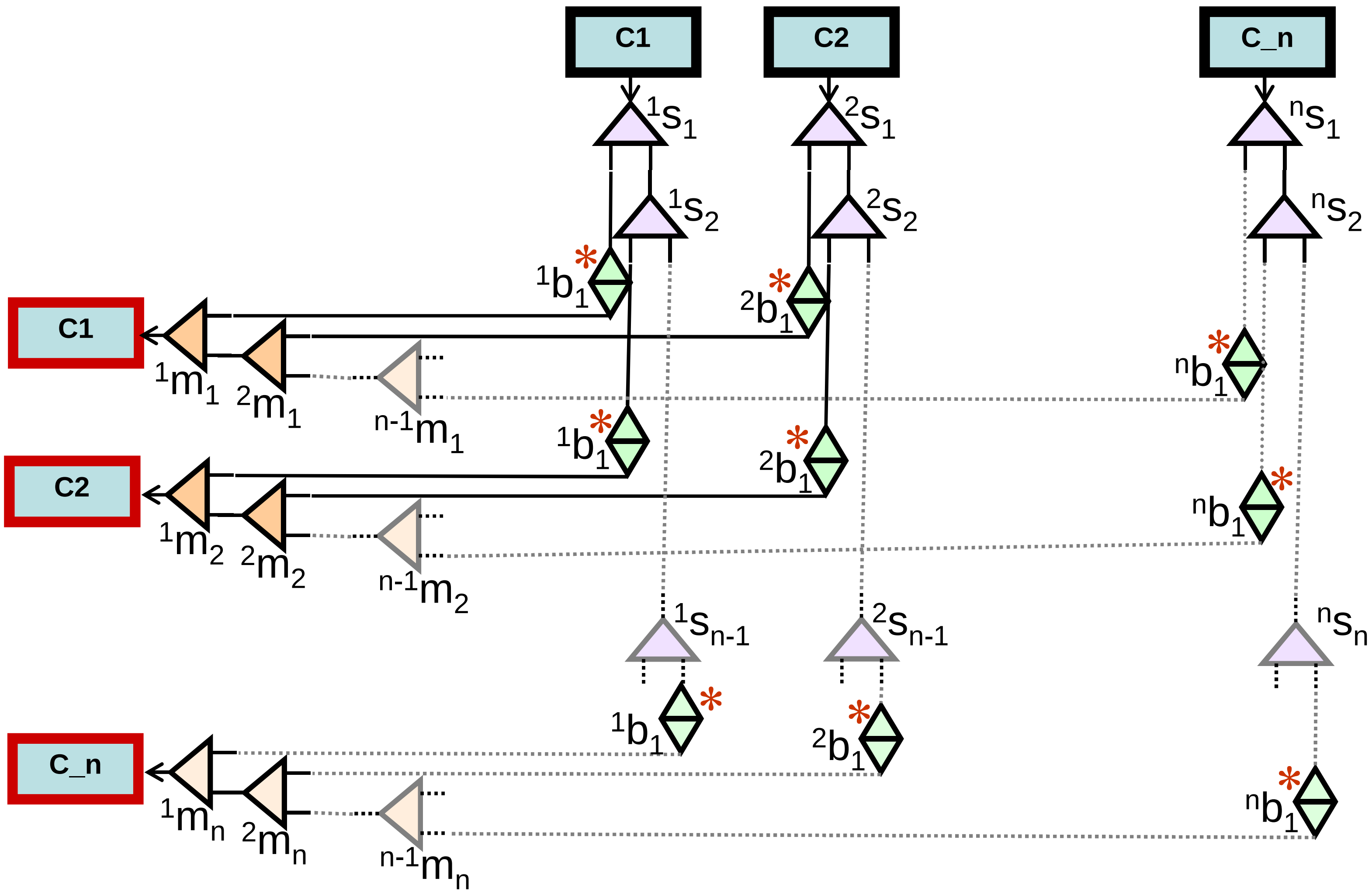
- **Two notions of equivalences** can be defined, one equates the external token flows, one takes into consideration the internal traffic interleavings in addition **(second one is a congruence)**

- **Normal forms and complete axiomatisation** can be defined for the first equivalence (so a network can be transformed in any equivalent one in a finite number of steps)

Algebraic rules and axiomatisation

- We shall define transformations which preserve functionality but not the timing (similar to retiming/recycling)
- The main goal is to figure the proper auxiliary transformations on switching patterns which will realize the same data transport (through a different link topology)
- Normal forms can be defined (but large, by expansion)
- We first introduce transformations on switching patterns (On / When) and their properties
- We then introduce the transformations on Merge/Select patterns

Normal forms (point-to-point links)



on/**when** operators

clock *filter*

(0.u) **on** v = 0.(u **on** v)

(1.u) **on** x.v = x.(u **on** v)

(u on v) is a subclock of u

(x.u) **when** (0.v) = u **when** v

(x.u) **when** (1.v) = x. (u **when** v)

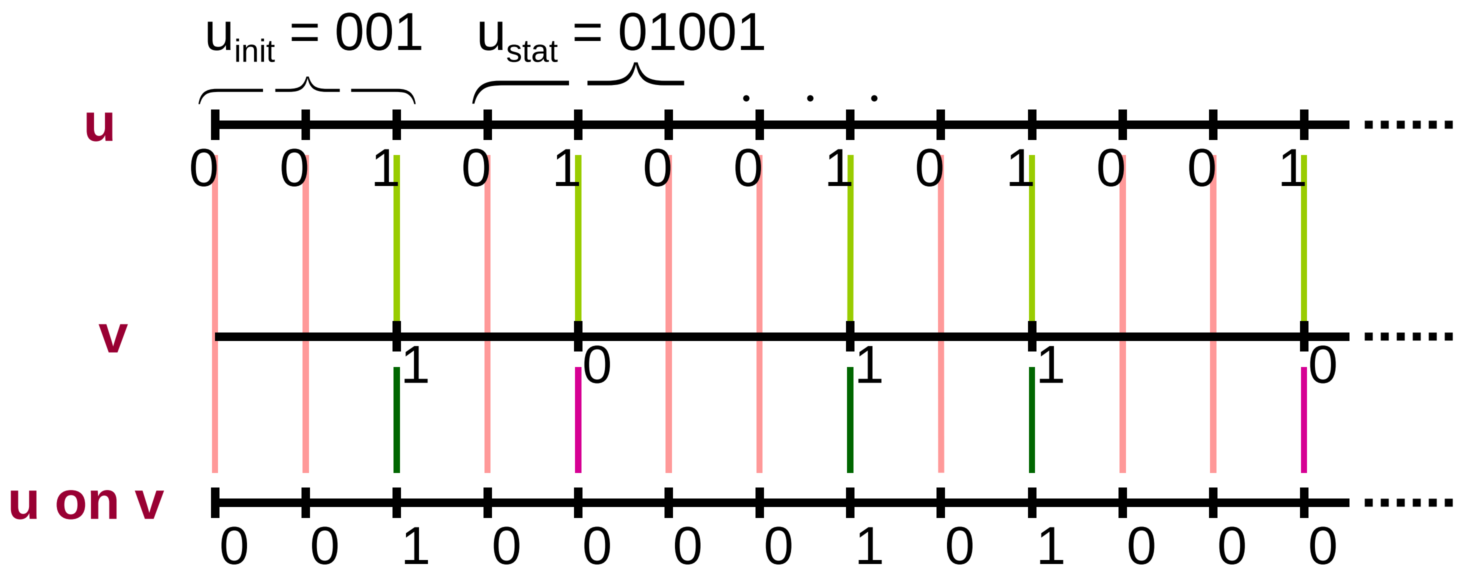
when u subclock of v, returns the filter

((u on v) when u) = v

if u subclock v, then u = (v on (u when v))

on effects

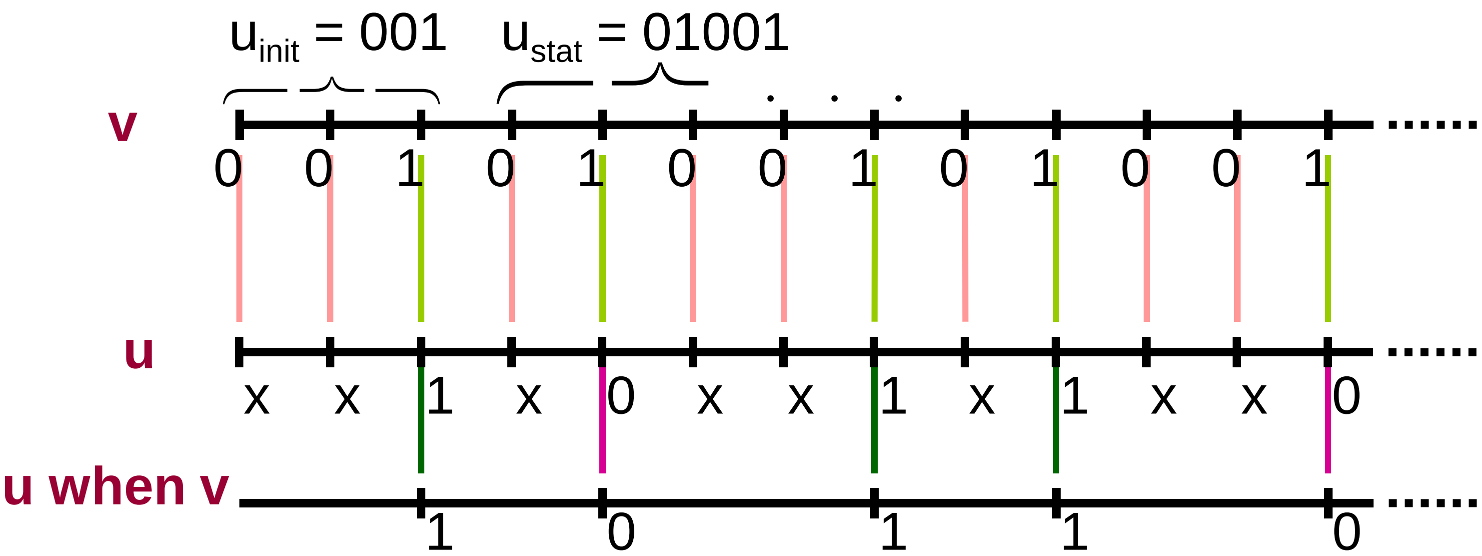
(0.u) **on** v = 0.(u **on** v)
(1.u) **on** x.v = x.(u **on** v)



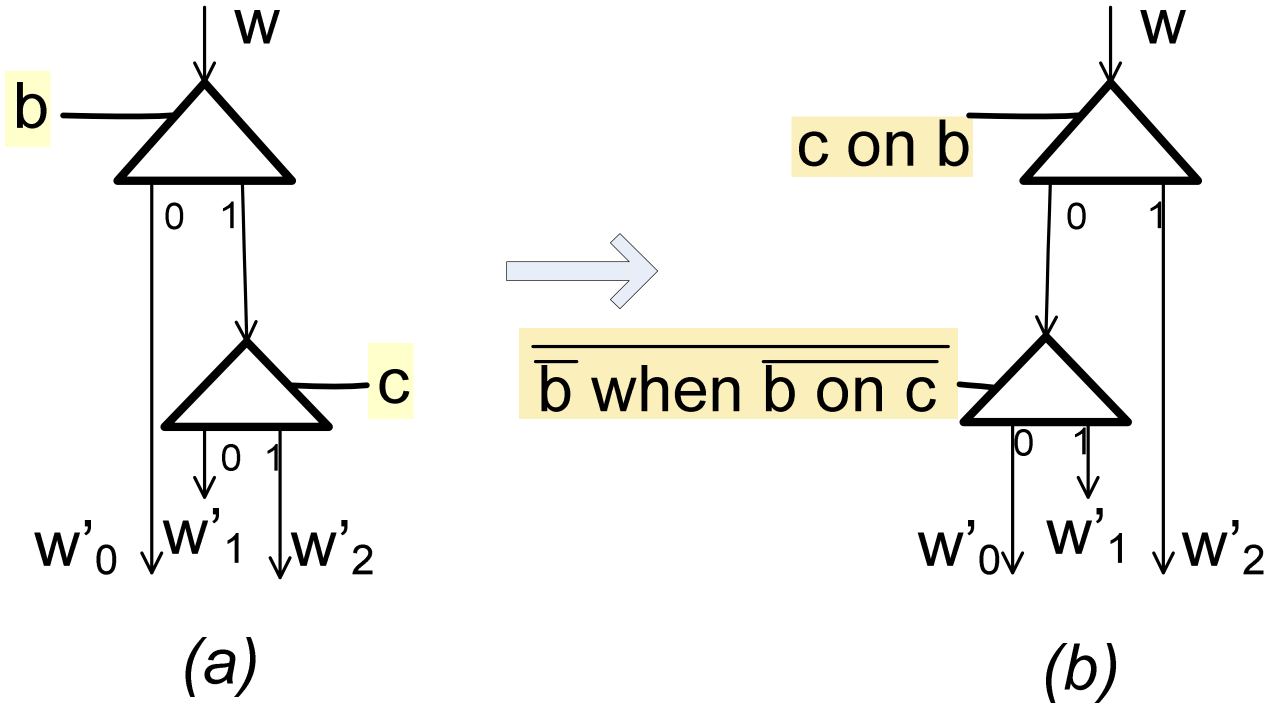
when effects

(x.u) **when** (0.v) = u **when** v
(x.u) **when** (1.v) = x. (u **when** v)

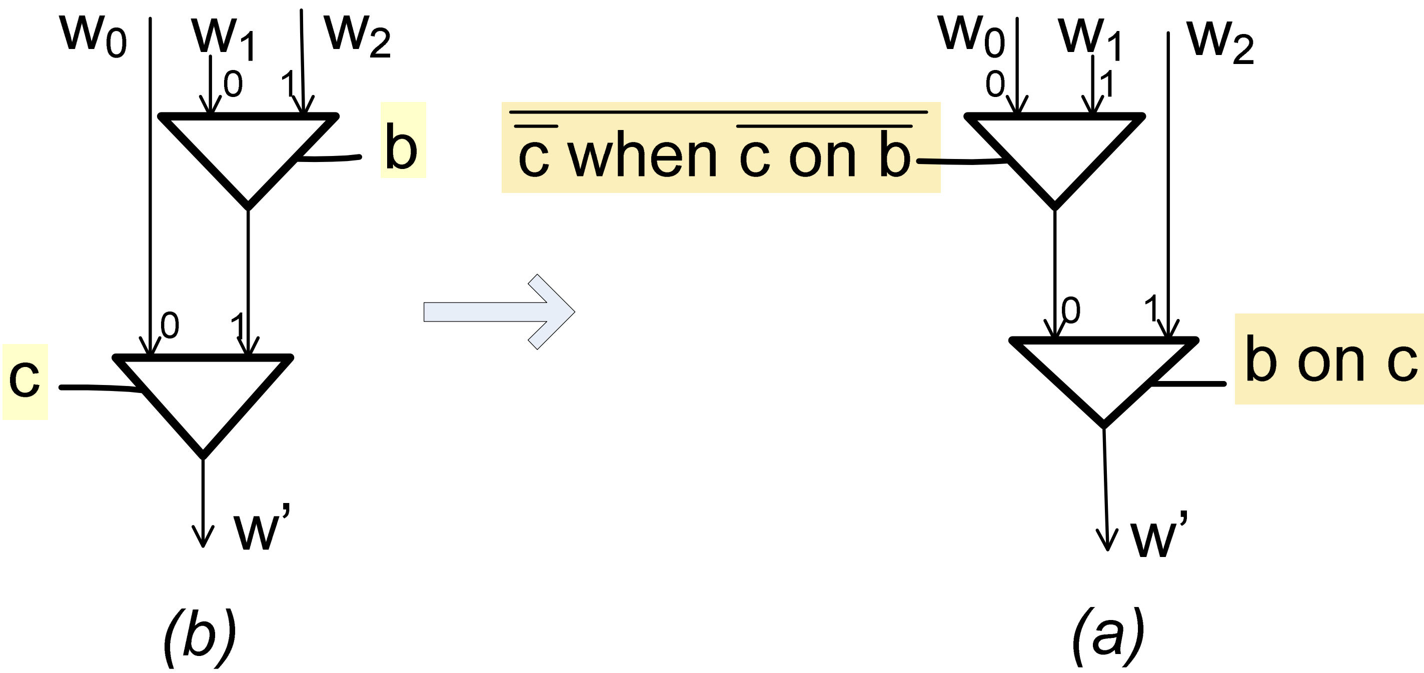
most meaningful whenever u subclock of v



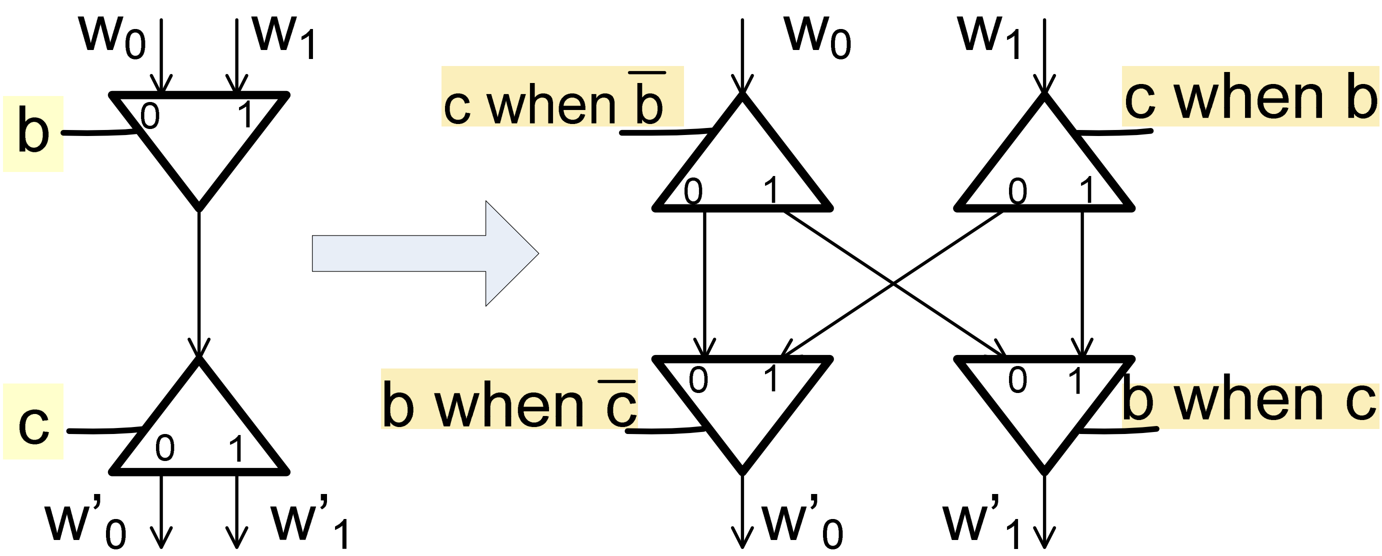
Transitivity of Selects



Transitivity of Merges



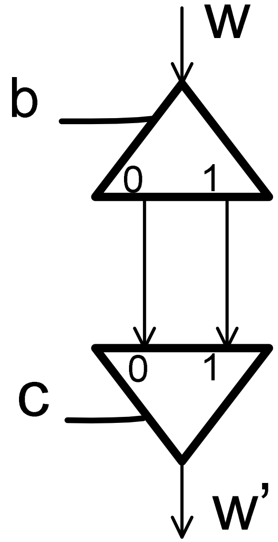
Selects up across Merges



(a)

(b)

A basic form



- Simplest, and disposable, when b=c (and initially empty inside)
- If throughputs match on the (hyper)-period, but order differ, may require buffering

→ then tokens are bypassed by others

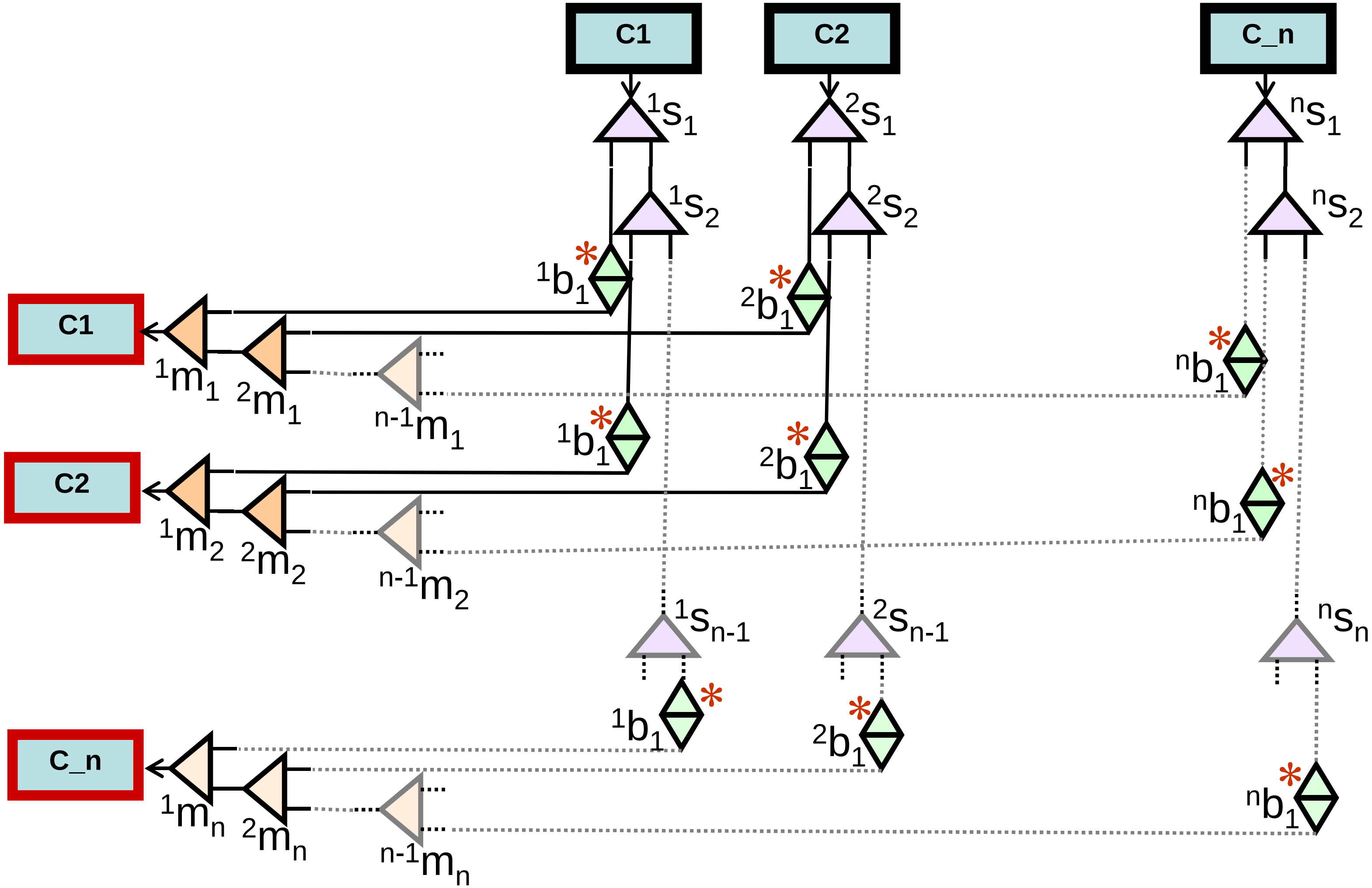
- Buffering remains bounded and predictable if ever b and c fixed and known.

- General permutations may require a sequence (or a combinaison) of such elements

→ **Even & Pnueli Permutation Graphs (1971)**

→ Optimization of the physical realization of the permutation using basic forms.

Normal forms (point-to-point links)



new

Facts

- **KREG traffic balance (token production = token consumption)** can be checked by **abstraction into SDF** over a hyperperiod.

  → Through this, **safety can be checked**
- **KREG networks can be unfolded into infinite Marked Graphs** *(construction sketch later)*
- Regularity of switching patterns allows to detect cyclicity, to **fold back into finite (quasi)-Marked Graphs**

  → **transient and steady phase can be expanded** into MGs, **remaining issue is to glue them back** together.
- **Two notions of equivalences** can be defined, one equates the external token flows, one takes into consideration the internal traffic interleavings in addition **(second one is a congruence)**
- **Normal forms and complete axiomatisation** can be defined for the first equivalence

for this presentation we shall ignore copy nodes, and assume that every cycle in the original KRG graph crosses a compute node (not essential restrictions, just make the presentation simpler)

Remaining issues

- Smoothness
  - Schedules with binary words as close as possible to the mean throughput (mechanical words) are feasible. Desirable ?
  - Extension to smooth routings ? How to report that data following a same route in burst mode may be beneficial ?
- Dependencies brought back to inputs: contracts ?
- Extension of the conflict-freeness property in the case of Esterel ?
  - Start P when S
  - Suspend P from S_begin to S_end
  - Abort P when S

Thank you !!

Questions ?

Process Networks

- **Conflict–freeness**

  Usual important requirement.

  Forbids choice between alternative local computations based on input token availability.

  Consequences: ***monotonicity*** (Kahn PNs), ***confluence*** (CCS), ***latency insensitivity*** (once enable, a behavior will eventually be performed, w/ a natural fairness assumption)

  Not true of general Petri Nets, Synchronous Languages (instantaneous preemption) → explicit absence notification in distributed implementations

- **Scheduling**

  Because of conflict-freeness, all computation traces are essentially the same partial-order under different timings. One specific **ASAP** representative.

  Rich theory of static regular scheduling (***k-periodic schedules***)

Self-timed → clocked