K-periodically routed graphs

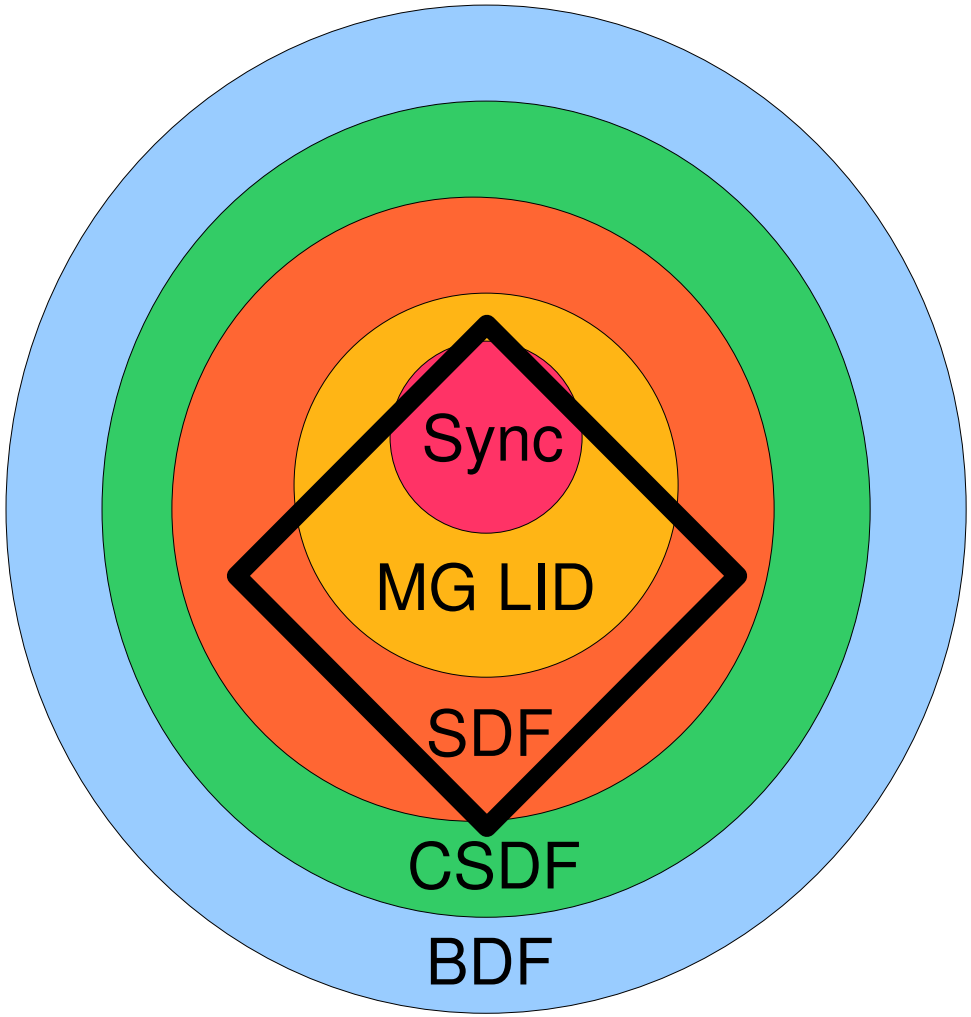
Julien Boucaron
Anthony Coadou
EPI Aoste

Basics

Motivation

- **Introducing control** in data-flow process networks while ensuring strong desirable properties such as **determinism**, decidable **safety** and **liveness**.
- Such control is mandatory for modeling and reusing components for **synthesis** on ASIC, FPGA and recent NoCs architectures.
- However, it comes at the cost of having “**static**” **control** otherwise safety and liveness cannot be decided (in general).

Background



Marked Graph,
Latency Insensitive Design
and **Synchronous Data Flow**:

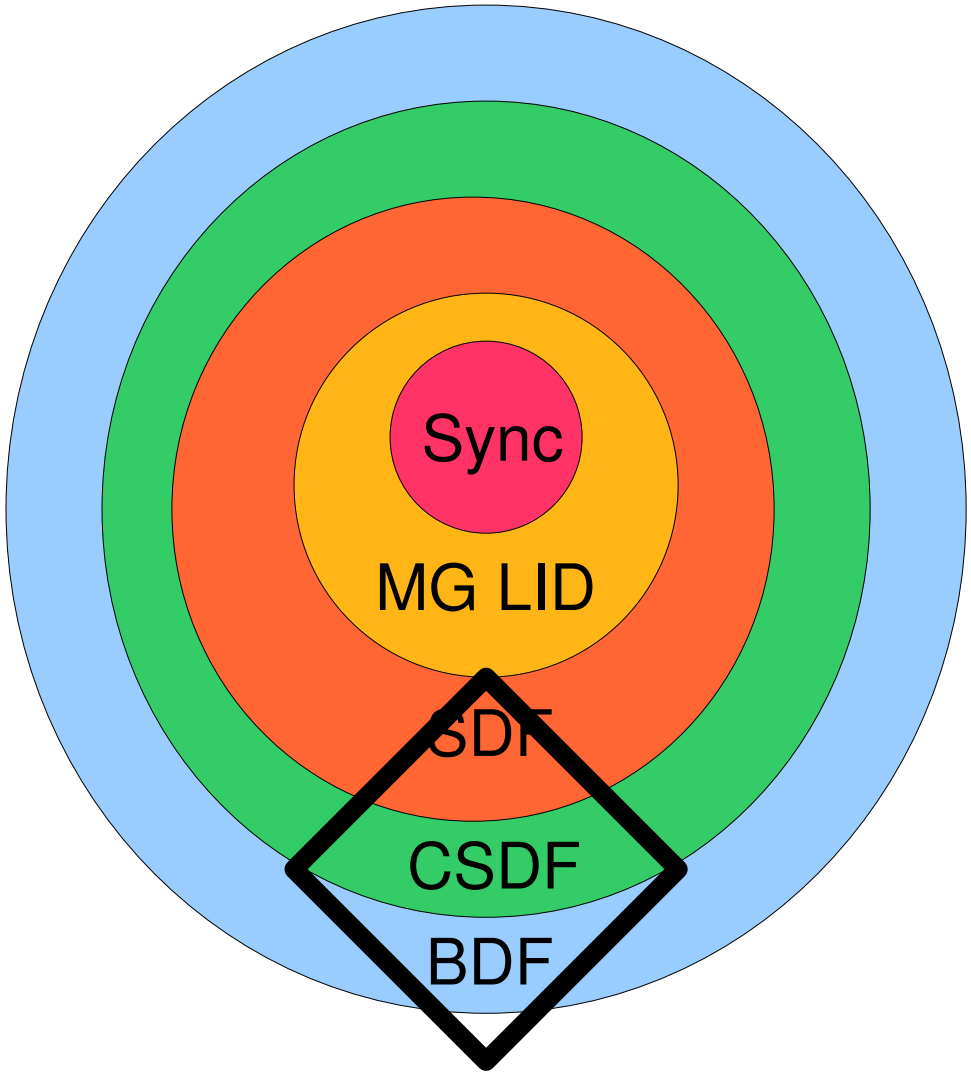
+ safety decidable

+ liveness decidable

(at least one token in each cycle/bounded execution firing)

- only point-to-point communication (no sharing)

Background



Cyclo-Static Data Flow:

+ safety and liveness decidable

- “no explicit” communication, embedded in node

Boolean-controlled Data Flow:

+ explicit communication with Switch/Select nodes

- data dependant control

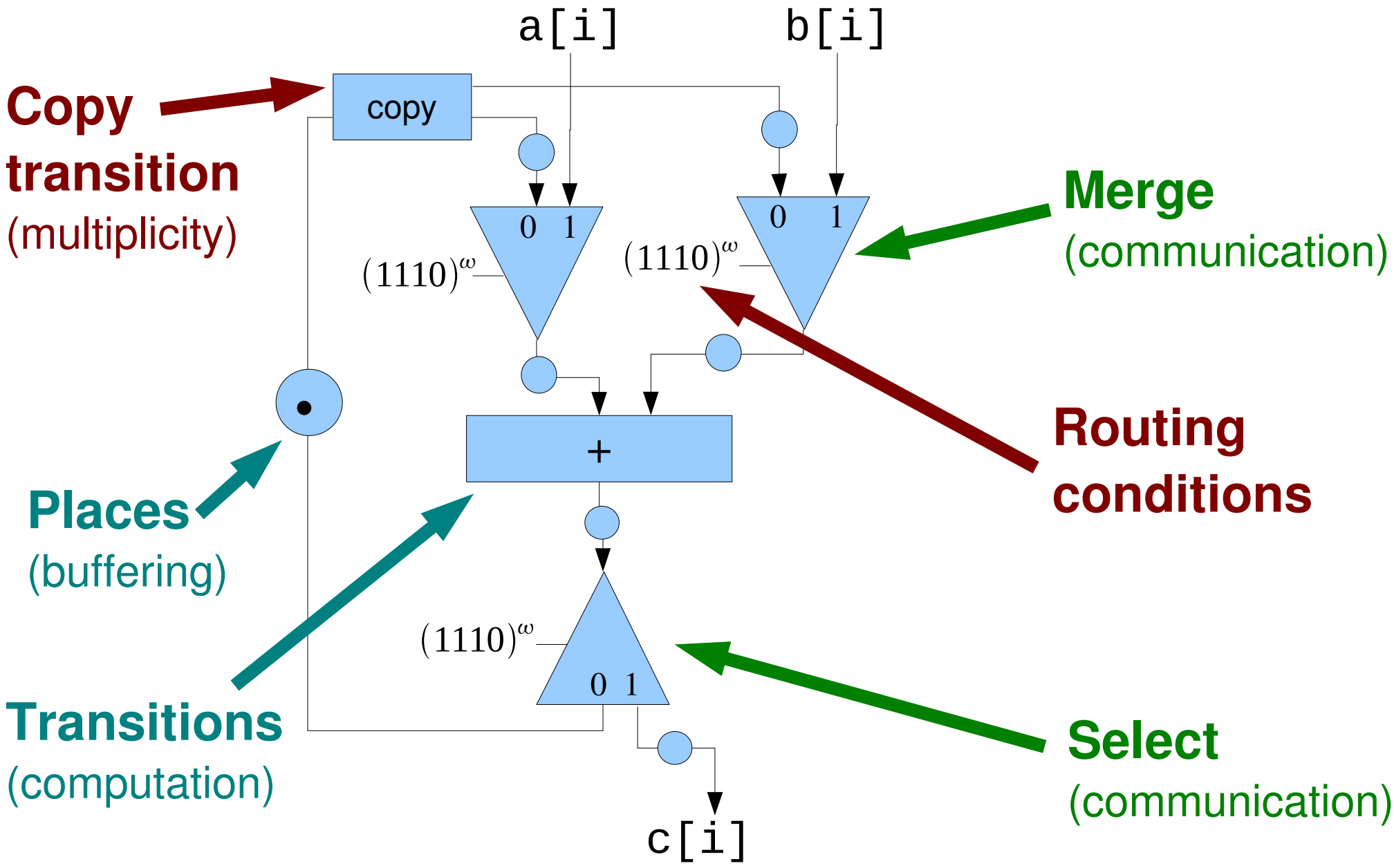
⇒ safety undecidable

So, what is it about?

K-periodically Routed Graphs

- **Low-level** (akin. assembly) model for both High-Level Synthesis/Compilation.
- **Concurrent**, **deterministic**, **confluent** (execution differs only by timing, compatible partial orders).
- **Safety**, **liveness** are decidable.
- Distributed memories.
- **Explicit communications** sharing through *Select* and *Merge* nodes annotated with offline computed “routing patterns”.

So, what does it look like?



Definition

A KRG is a collection of:

- **Computation nodes** (transitions)**:** consume and produce one token when fired, need at least one token on each input.
- ***Select*** and ***Merge*** **nodes:** consume and produce one token on associated input/output, with respect to a routing condition.
- **Places** (edges)**:** exactly one input and one output, key property for both confluence and determinism.

Routing conditions

- Specific binary sequence associated to each Select/Merge node where:

  0 (resp. 1) means “take 0 (resp. 1) branch”

- Binary sequences of the form: $u.(v)^{\omega}$

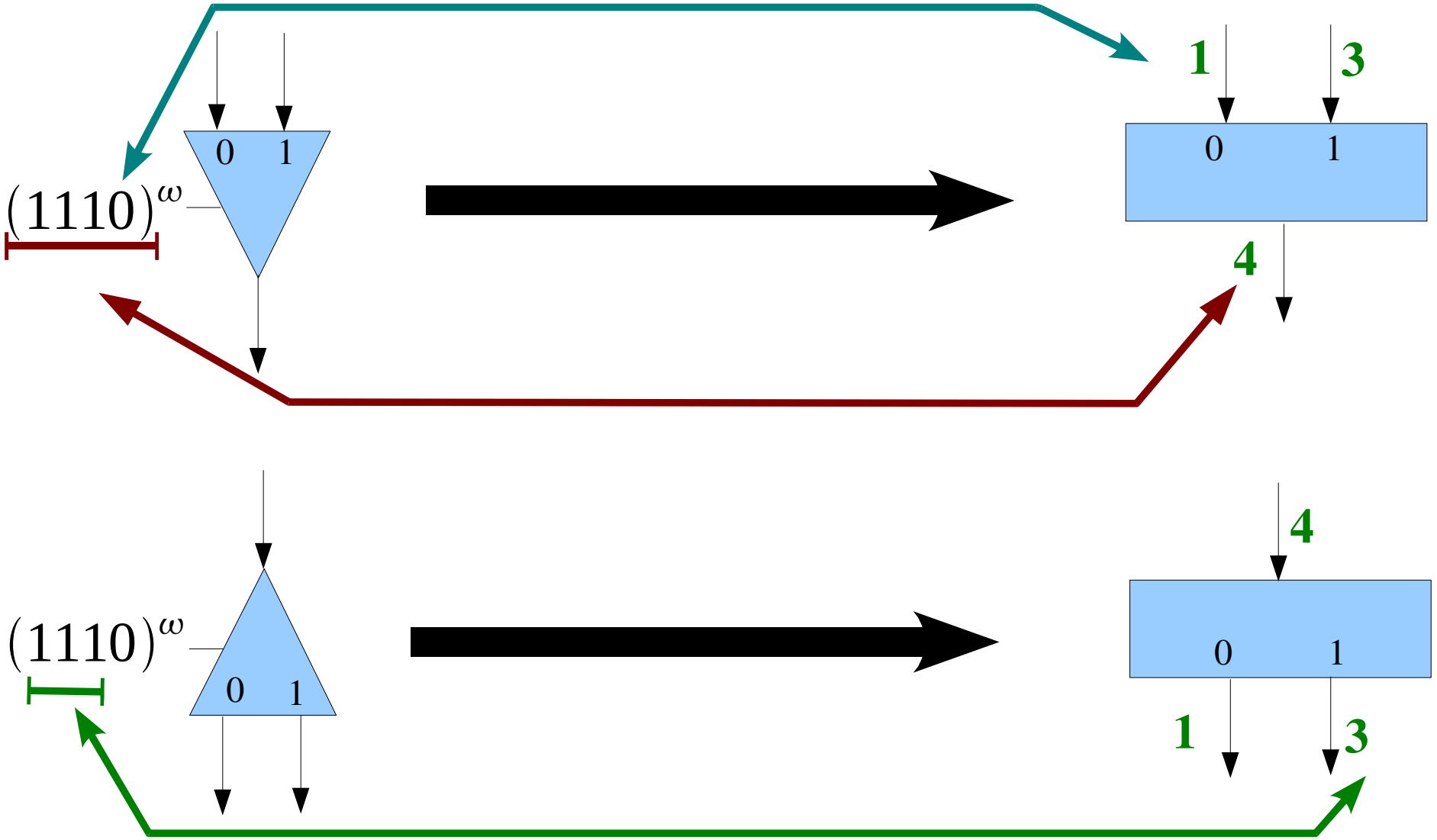
  where ***u*** is the **initialization** part, and ***v*** the **periodic** part (repeated infinitely).

- Those conditions are **computed off-line**, safety is thus decidable.

Decidability of safety

- How?
  - As done in CSDF, **abstraction reduction** from KRG to SDF.
- Construction:
  - Places, computation nodes and initial marking unchanged.
  - For each Select/Merge, create a **new SDF node** associating the global number of production/ consumption associated for the **periodic** part of the switching condition.

Decidability of safety



Liveness checked with bounded-length execution

KRG Transformations

What's new w/ CSDF?

- Both KRG and CSDF use sequence of booleans/ integers to describe the static control.
- The advantage of KRG is the axiomatization we have built using ***on*** operator (borrowed from N-Synchronous theory) and the ***when*** operator (different from the synchronous one).
- Let us build correct by construction transformations.

On and *when* operators

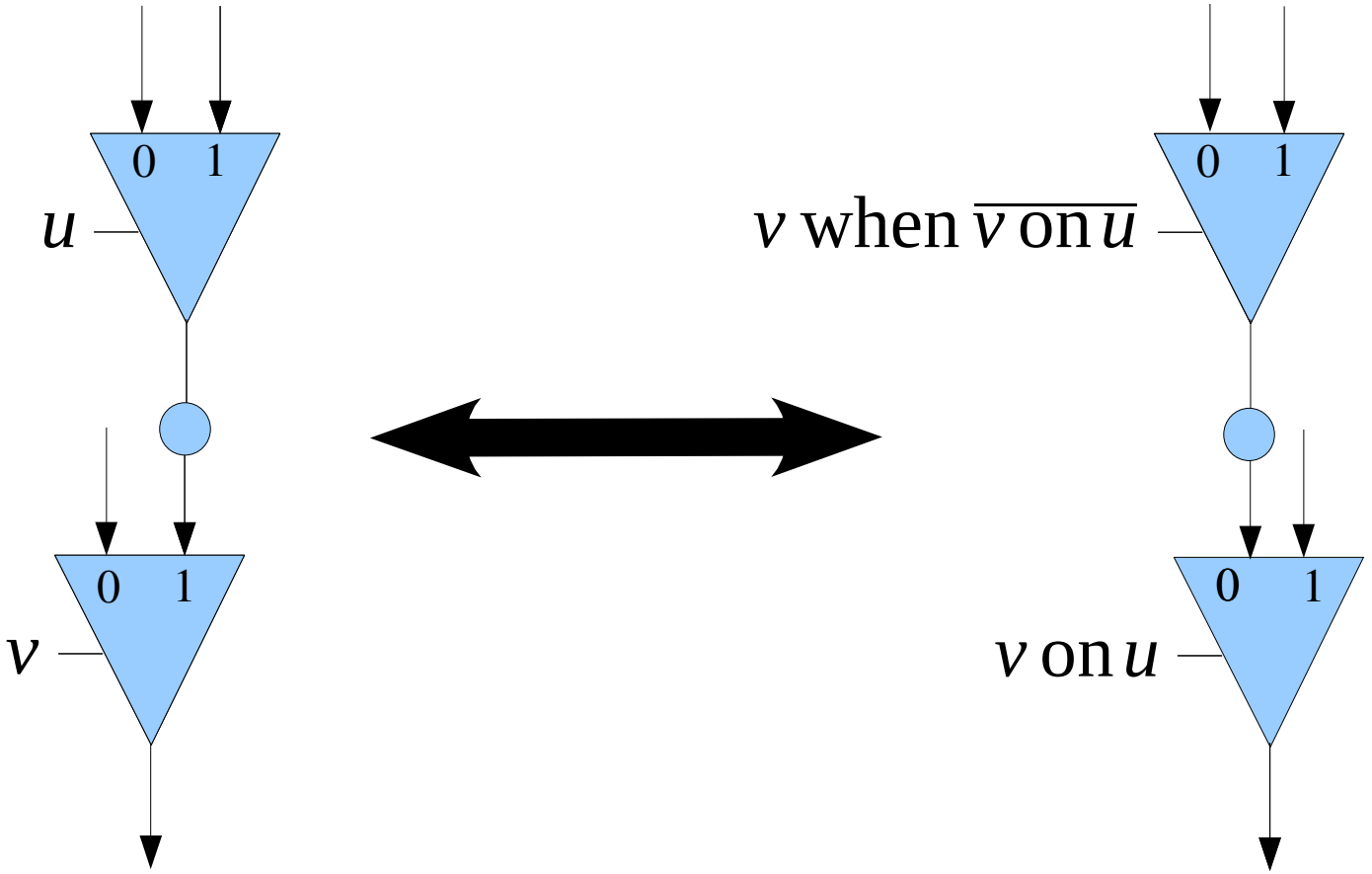
$$(0.u) \textbf{ on } v = 0.(u \textbf{ on } v)$$

$$(1.u) \textbf{ on } x.v = x.(u \textbf{ on } v)$$

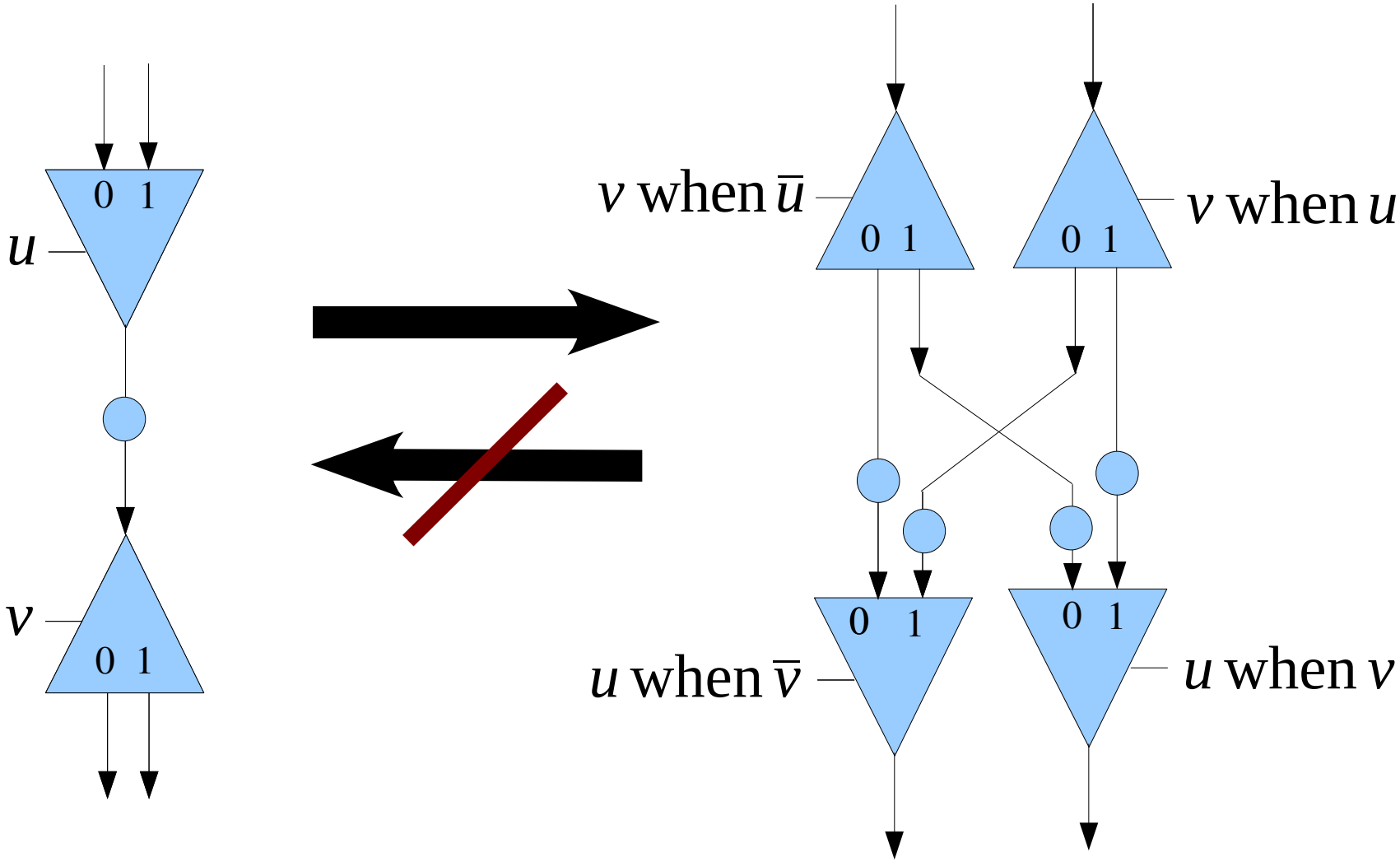
$$(x.u) \textbf{ when } (0.v) = u \textbf{ when } v$$

$$(x.u) \textbf{ when } (1.v) = x.(u \textbf{ when } v)$$

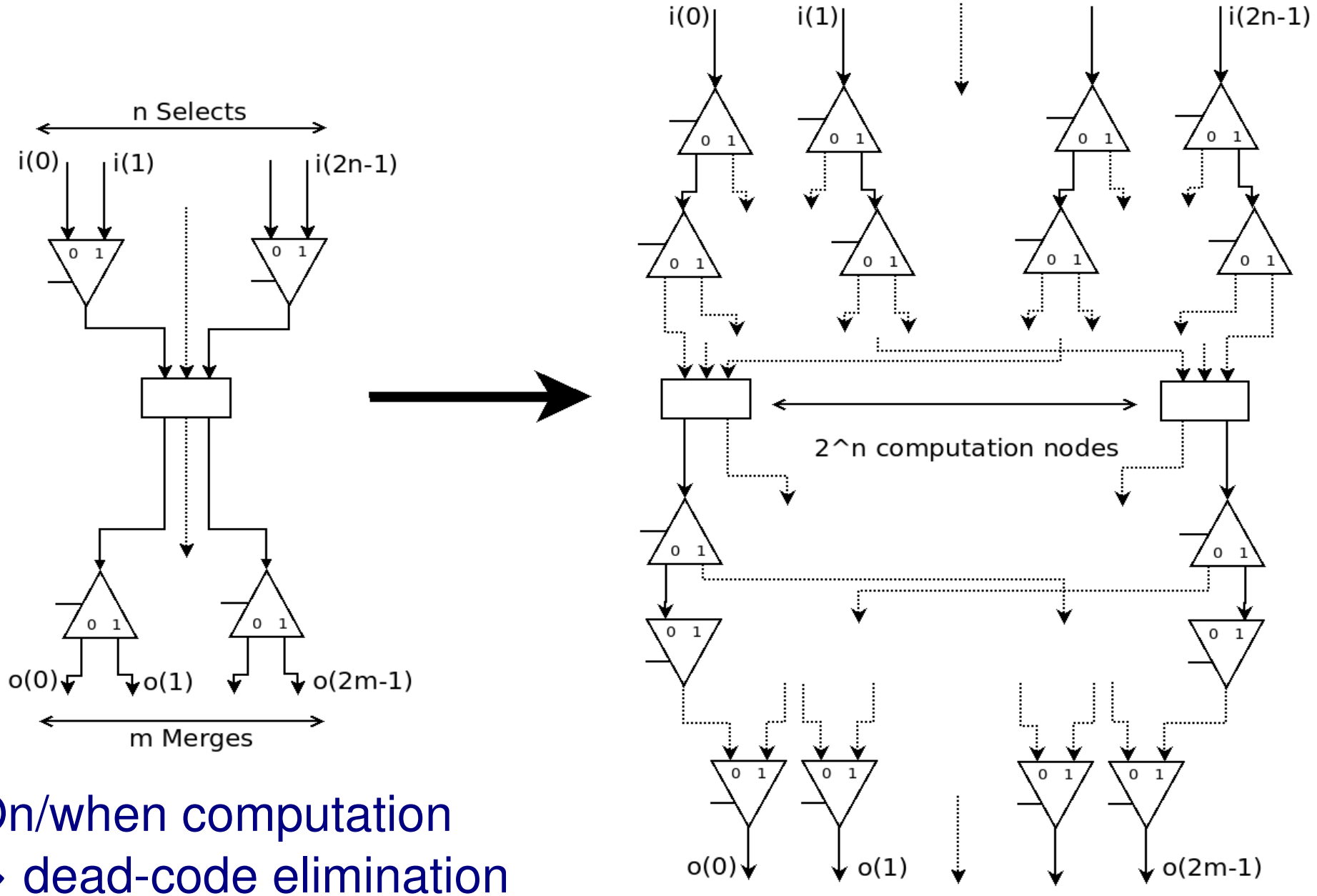
Select/Merge permutations



Splitting shared links



“Shannon-like” expansion



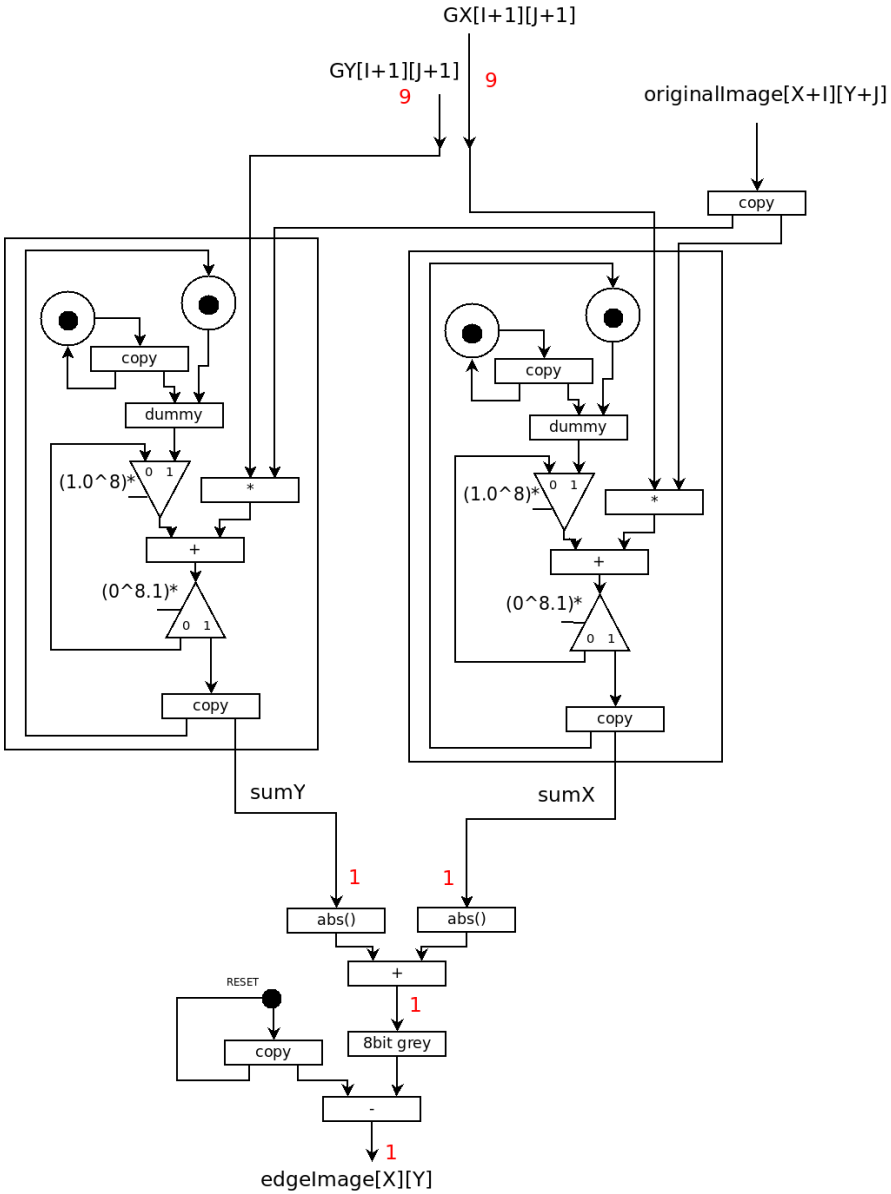
On/when computation

⇒ dead-code elimination

KRG in a nutshell

- Low-level model for compilation and HLS.
- Determinism, concurrent, confluent.
- Communications → Select/Merge + k-periodic conditions computed off-line.
- Multiplicity → Copy transition.
- Safety and liveness decidable.
- On/When operators → transformations on KRG preserving “behavior”.
- “Shannon-like” expansion process → dead-code elimination using On/When ops.

Link with nested loops (Sobel filter)




```
int sumX = 0; int sumY = 0;

/* Fusion of both loop on left and split loop on
 * sumX and sumY */
for(int I=-1; I<=1; I++) { //Domain size 3
  for(int J=-1; J<=1; J++) { //Domain size 3
    sumX += originalImage[X+I][Y+J] * GX[I+1][J+1];
    sumY += originalImage[X+I][Y+J] * GY[I+1][J+1];
  }
}

int SUM = abs(sumX) + abs(sumY);

/* To 8 bits grey levels data dependent control
 * abstracted as dataflow (if-conversion) */
if(SUM>255) { SUM=255; }

edgeImage[X][Y] = 255 - SUM;
```

Demo using KPASSA v.2

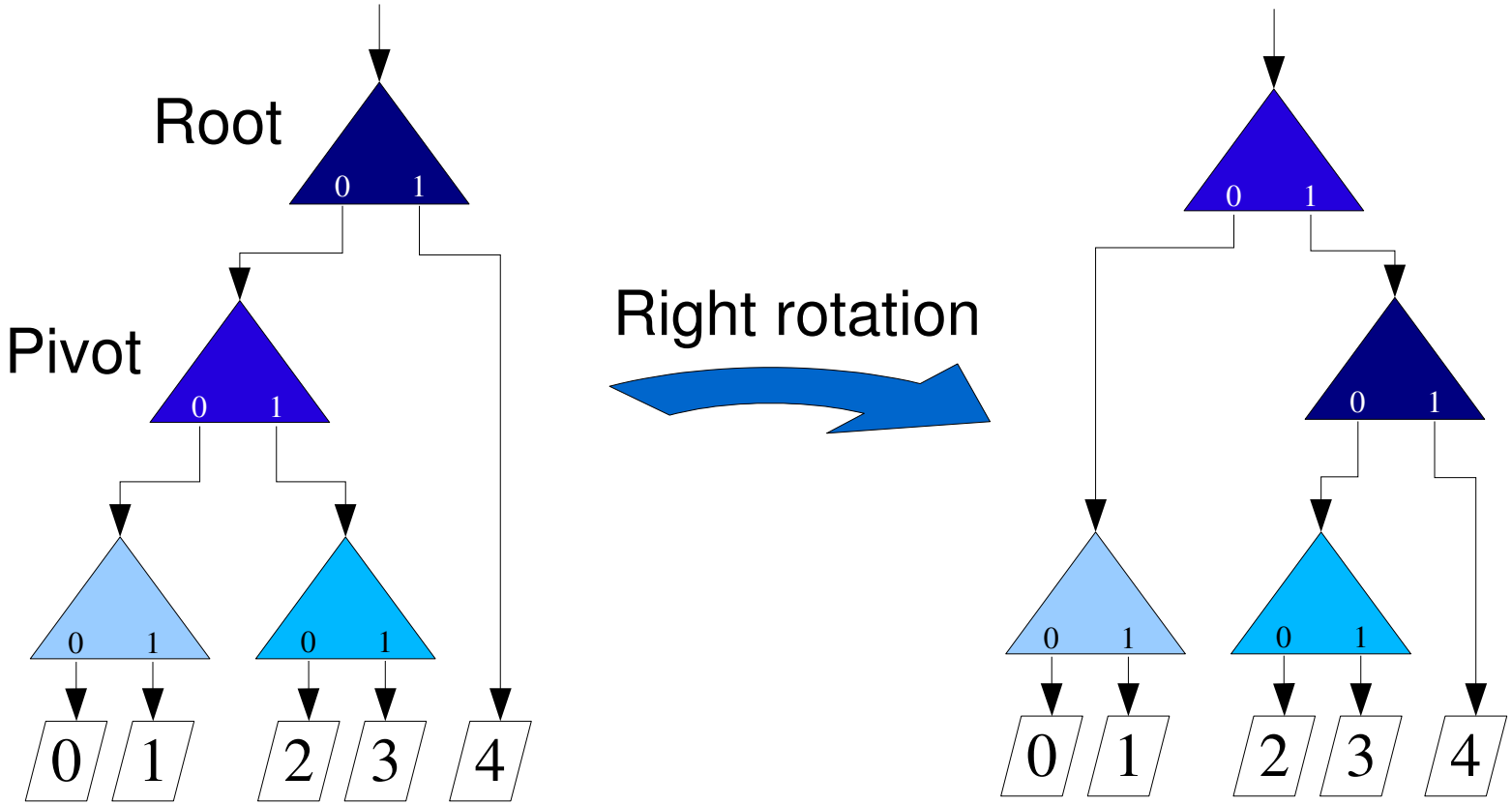
Further works on KRGs

What do bring Select/Merge prop. ?

- Schedules and memories sizes can be computed.
- Equivalence (or not) after routing transformations can be proved.
- Order relations on token flows can be defined and permutations can be characterized.
- Conversely, one can model a design where actors do not consume tokens in the same order as they are produced, using permutation blocks with a minimal number of paths.
- Traffic balancing.

What do bring Select/Merge prop. ?

- Select/Merge permutation equivalent to tree rotation.
- Enable to reuse well-known results on AVL trees to reduce routing tree depth and/or to balance traffic.

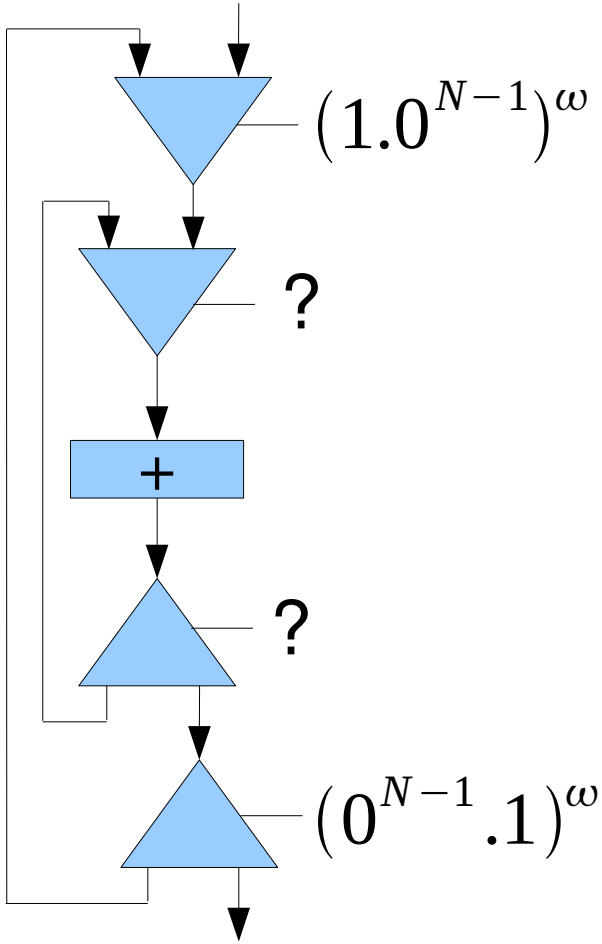


Ongoing work: improving control

- Static periodic behaviors are too limited:

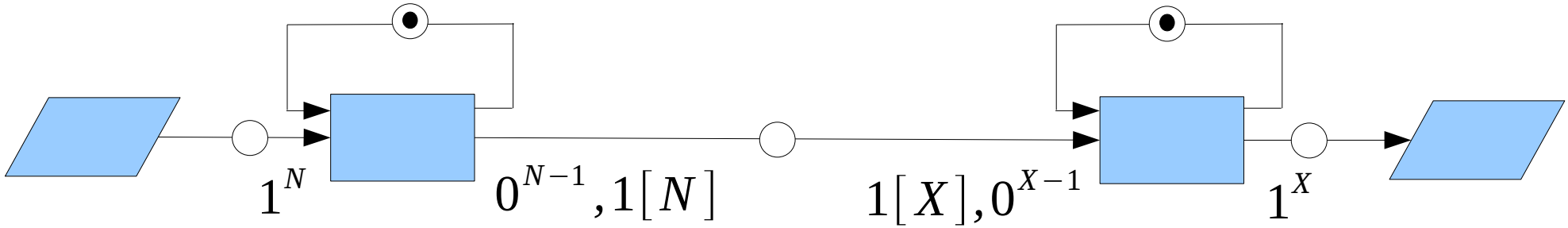
```
for (int i = 0 ; i < N ; ++i)
    for (int j = 0 ; j < i ; ++j)
       S += ...
```

- Loops bounds can be parameterized and/or depend on other indices.
- Need of control FSMs generating binary sequences.
- Extend current model in a CDDF/ SPDF-like way.



Ongoing work: improving control

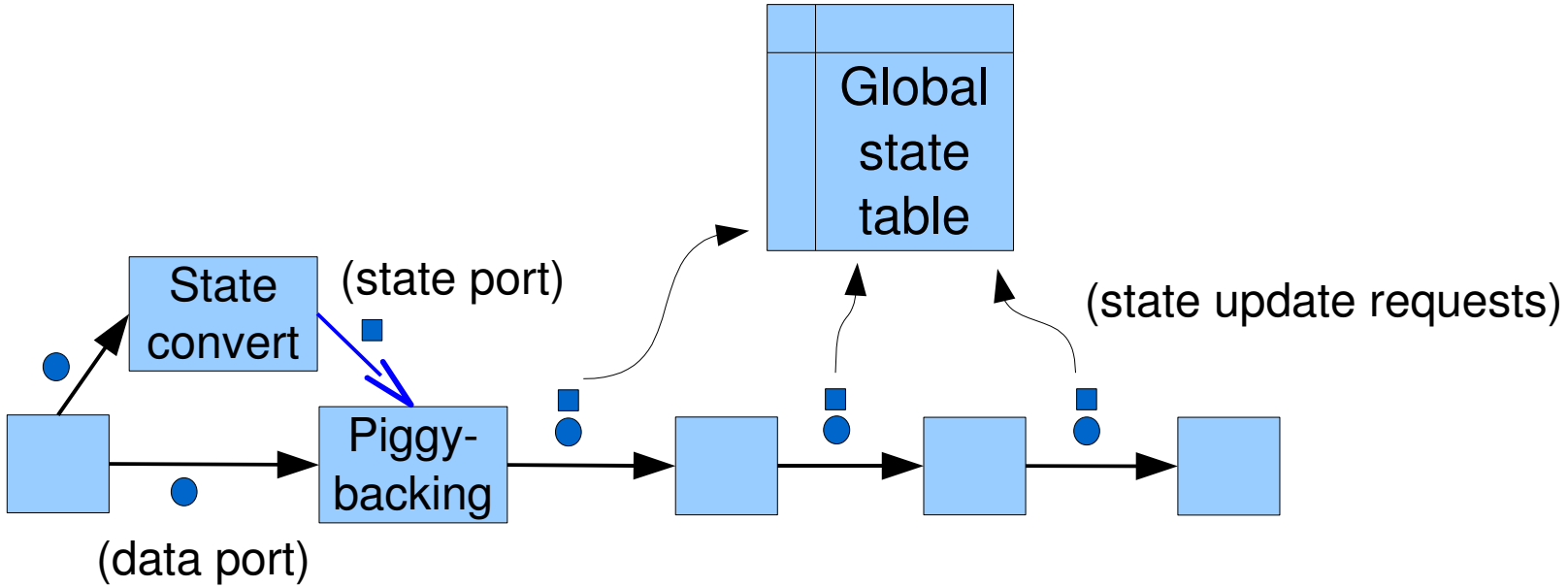
- Example of Cyclo-Dynamic Dataflow design [Wauters et al., 1996]:



- Executions are periodic, but their length depend on special tokens values (between [ ]) or symbolical variables.
- Consistency can be proved, unlike with general BDF.
- Memories can be bounded as long as parameters intervals are known.

Ongoing work: improving control

- Example of Synchronous Piggybacked Dataflow design [Park et al., 2002]:



- Behaviors of the different computation nodes may depend on global settings.
- GST entries sizes can be statically computed.

Ongoing work: improving control

- Both kind of control are commonly used in real-life designs.
  - CDDF: parameters forwarded from node to node as tokens.
  - SPDF: parameters stored in a global memory ; each node update its behavior when needed.
  - E.g. video decoder: **global parameters** set while retrieving data in memory, and parameters between each pipeline stage **encoded in the stream**.

Ongoing work: improving control

- Link KRG—SPDF maybe more natural :
  - In our case, schedules are a consequence of initial marking and routing conditions, while in CDDF, it is the behavior of computation nodes which is parameterized.
  - SPDF present a “trick” to associate both control and data flow. Control can be led by a syncChart as well.

Ongoing work: hierarchy

- Hierarchy is useful for the reusability of a design.
- Does there exist a nice way to express hierarchy?
  - Hard to handle fine-grain hierarchy (e.g. how to abstract schedules?)
  - Even coarse-grain hierarchy is useful for GALS designs.

Ongoing work: hierarchy

- Links with HCFSM [Girault et al., 1997] and DFCharts [Radojevic et al., 2006]:
  - Basically, layered abstraction where a FSM is abstracted as an SDF node and conversely.
  - Extended in DFCharts, where FSMs and SDFs may communicate through asynchronous ports.