

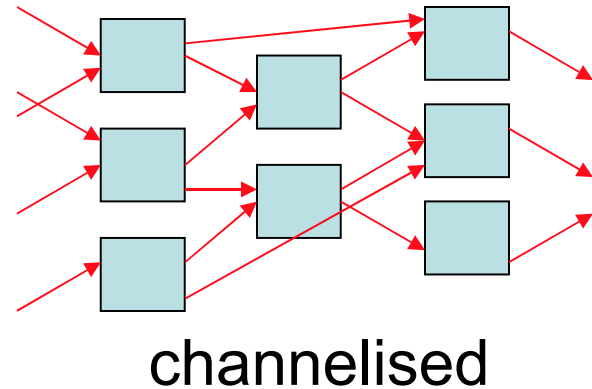
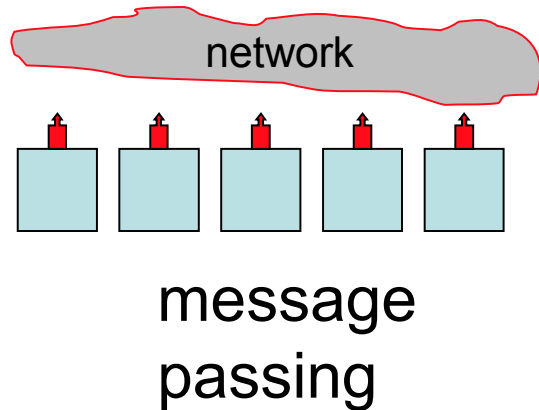
# SNet: a component model for multithreaded programs

Alex Shafarenko

*University of Hertfordshire*



# Communicating components



- Message passing:
  - structureless communication, (generally) no protocol support, no data logic, hard to optimise, hard to get right.
  - no constraints, no topology, all-to-all direct connectivity
- Channelised communication:
  - protocol support, data logic, buffering, topology, locality, etc.
  - lack of flexibility, extensibility, adaptability, etc.

# Component behaviour



Standard view:

awaiting particular input a,b,c

creating output d,e,f

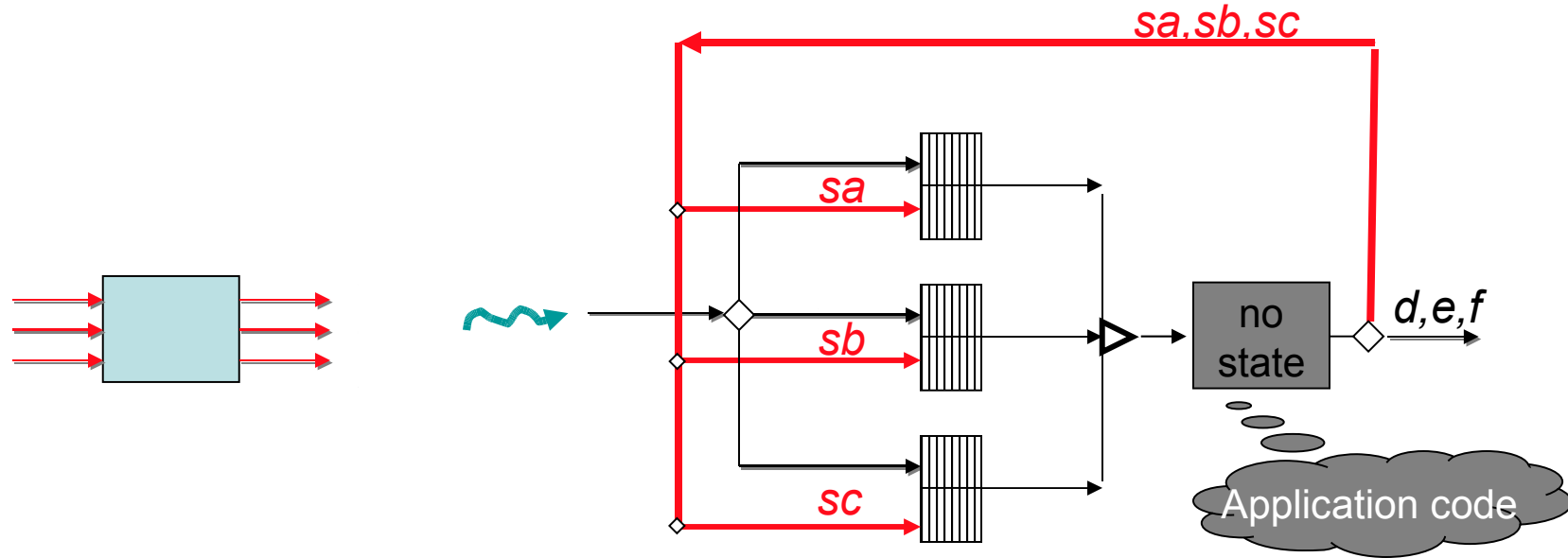
transitioning from state to state in silence

Hazards:

deadlock, starvation, overproduction, etc.

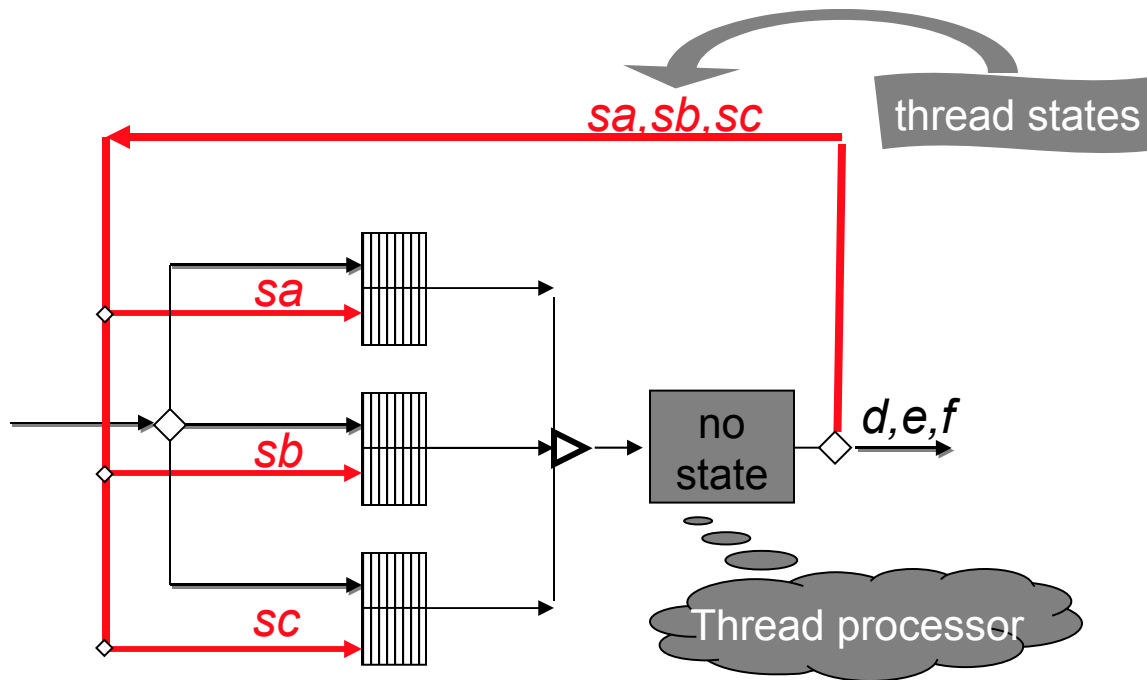
**Functionality** (output in response to input) **and control** (readiness for a channel, output on a channel) **are mixed up with computation, confused and usually mishandled.**

# Solution



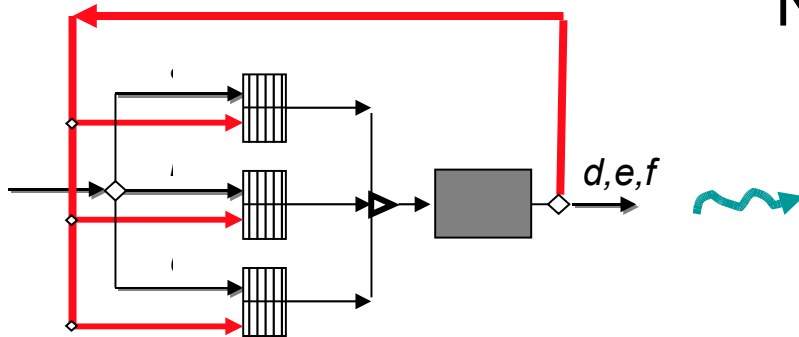
- Reveal synchronisation structures *in data*, using *synch-queues*, and *feedback*;
- No need for multiple channels (provided there is enough synch-space)... Single Input Single Output (SISO)
- still “channels” can interfere with one another, while original ones do not, but *is it an application programmer’s issue?*
  - separation of concerns...

# Alternative view

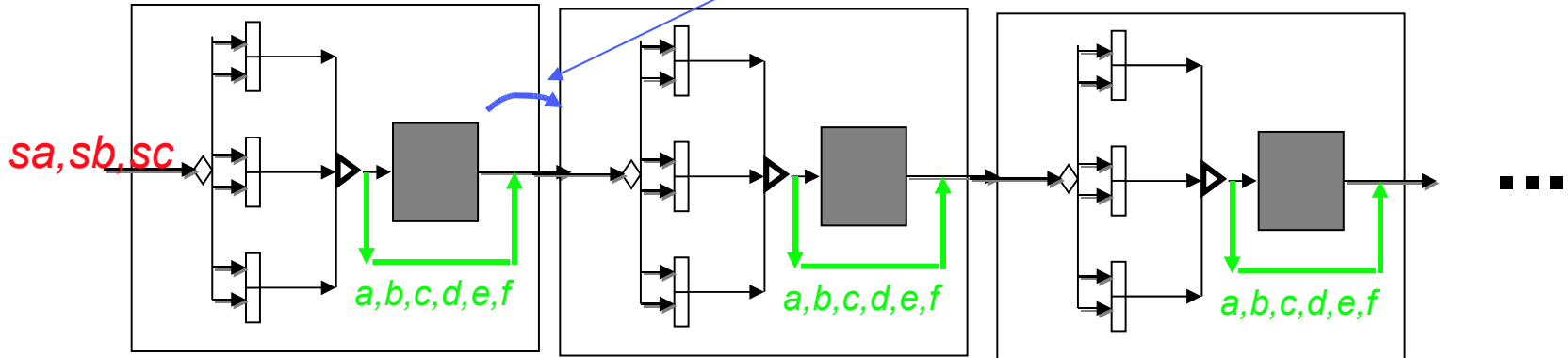


# Feed forward idea

No deadlock given the resources



Thread family launch



Type system to provide automatic bypassing

# Gather requirements

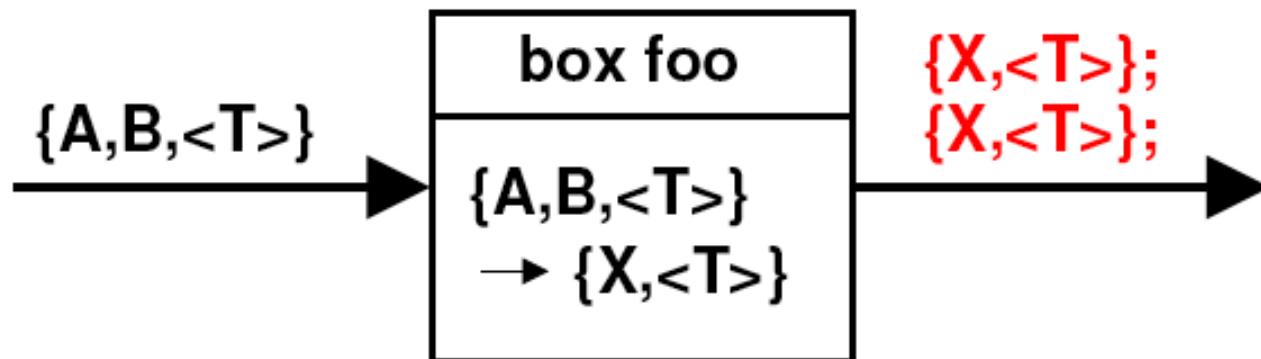
- Stateless SISO boxes
- Synchrocells with internal bypass
- Effective bypassing mechanism
- Replication

# Key ideas

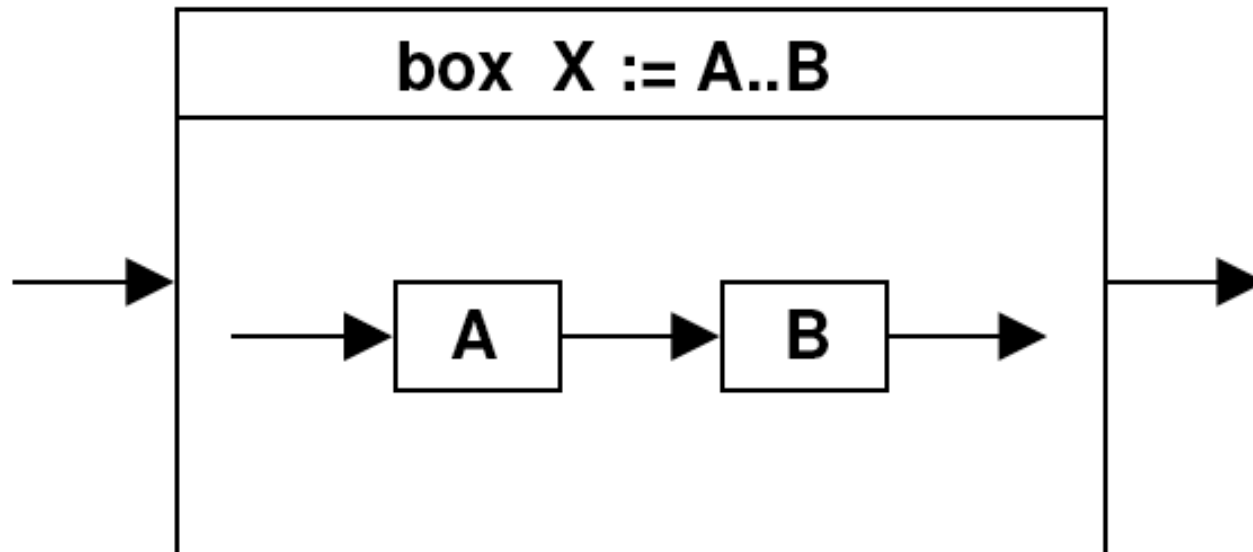
- Isolate application programming inside SISO stateless boxes.
  - No box-to-box communication other than by streams, i.e. box *side* walls opaque
  - Use *any* programming language: the result will be functional, i.e. stateless if no I/O and no mutable static memory is permitted.
- Use the type system to do part of the wiring
- Use SISO to SISO combinators to define topology
- Exploit nondeterminism rather than fighting it
- Separate out concerns: data correctness inside boxes and concurrency outside boxes

# S-Net at a Glance

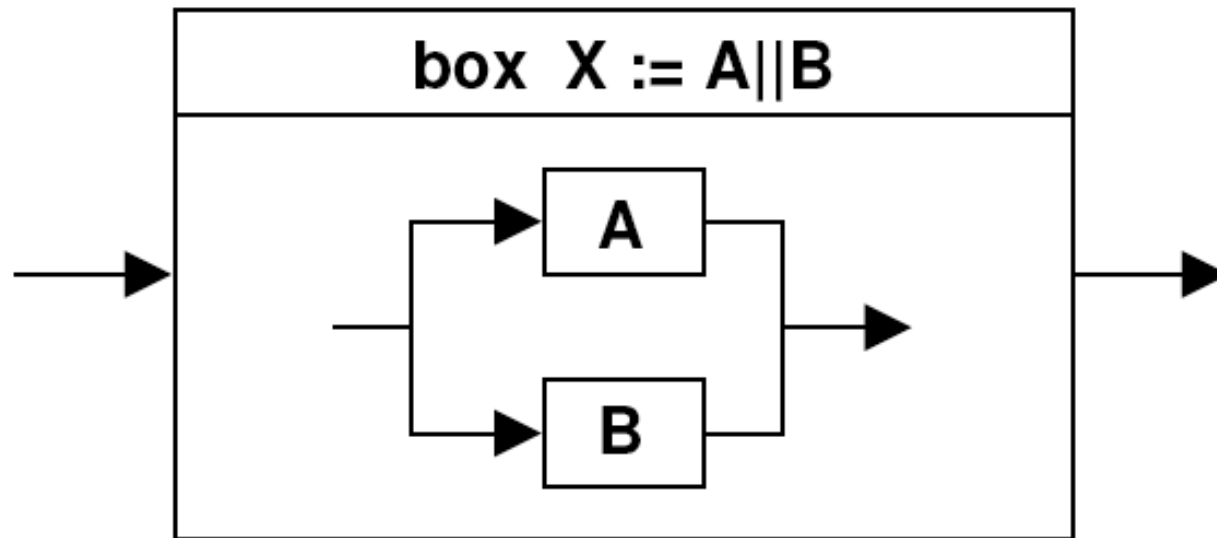
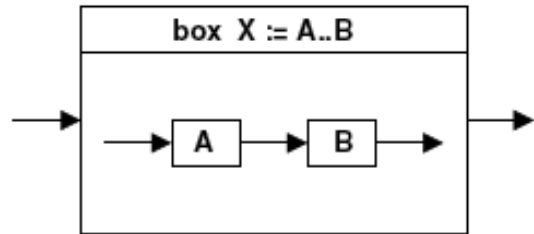
- ▶ central construct: box
- ▶ connected by
  - ▶ single input stream
  - ▶ single output stream
- ▶ streams transport records: sets of named fields
  - ▶ opaque value fields
  - ▶ integer-valued tag fields
- ▶ box behaviour declared by type signature
- ▶ behaviour defined in box language, not **S-Net**
- ▶ **box maps single input record to stream of output records**



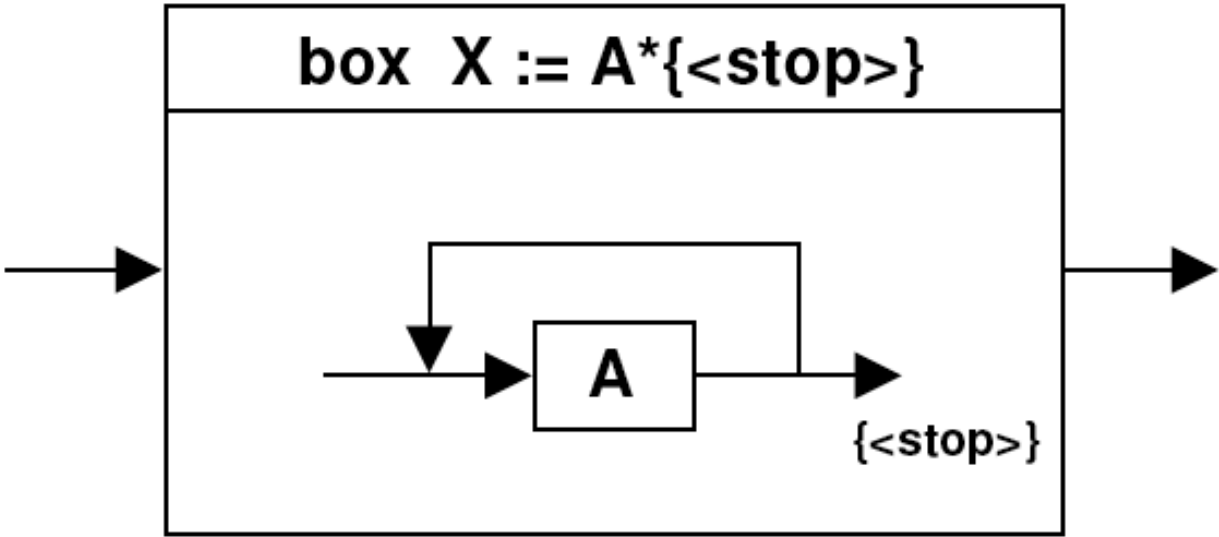
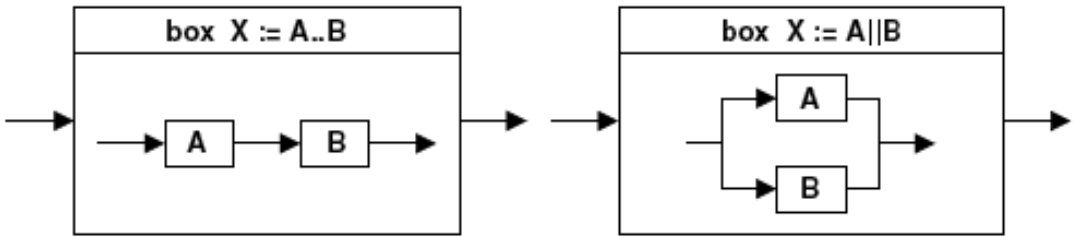
## Network Combinators: Serial



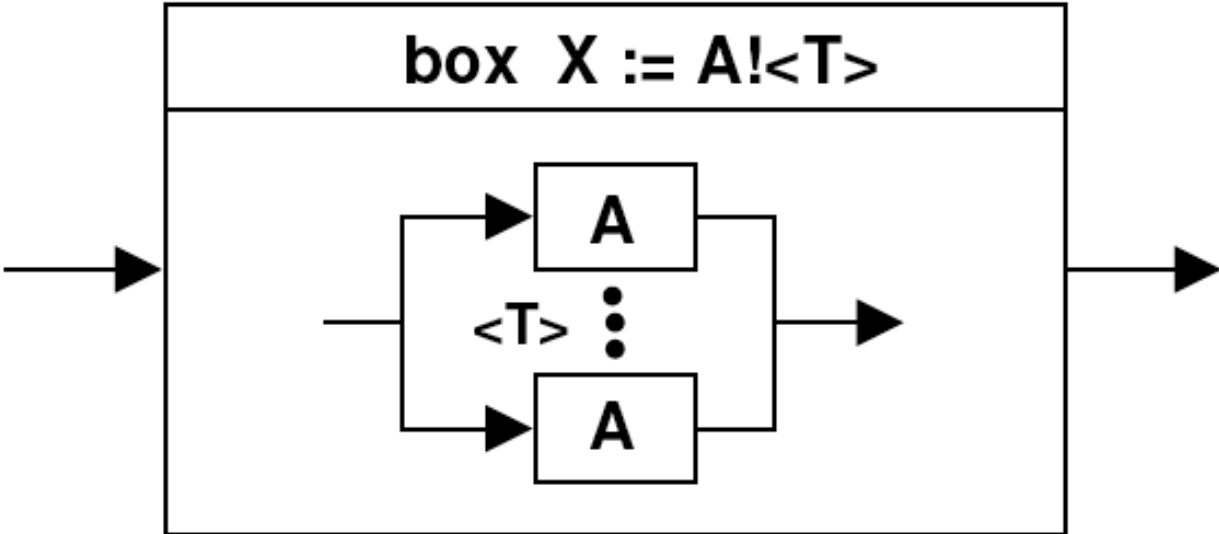
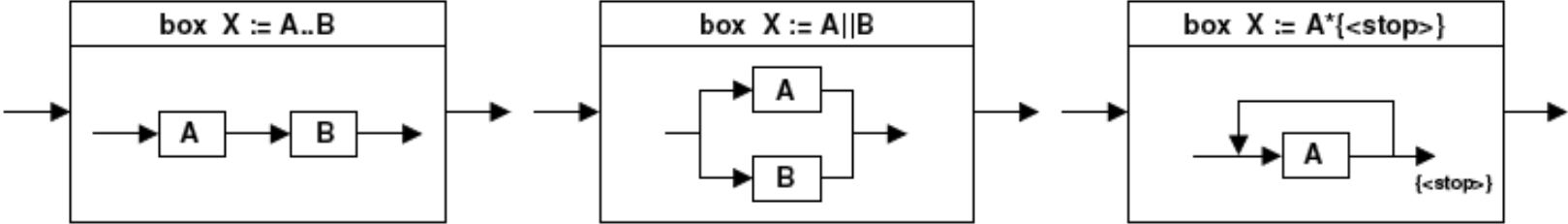
# Network Combinators: Choice



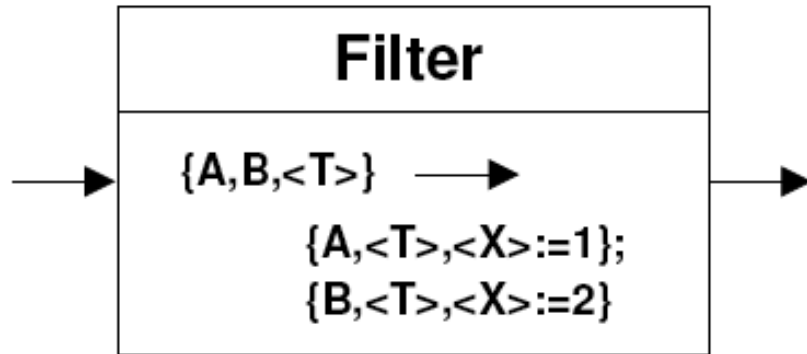
# Network Combinators: Infinite Replication



# Network Combinators: Index Split



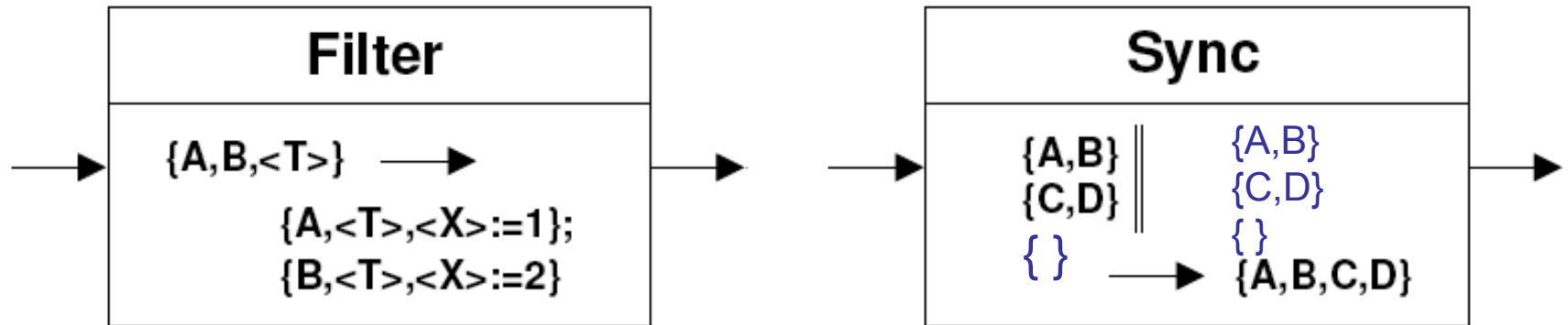
# Primitive Boxes: Filter and Sync



housekeeping:

- ▶ eliminate record fields
- ▶ duplicate record fields
- ▶ add tags
- ▶ manipulate tag values
- ▶ ...

# Primitive Boxes: Filter and Sync



housekeeping:

- ▶ eliminate record fields
- ▶ duplicate record fields
- ▶ add tags
- ▶ manipulate tag values
- ▶ ...

synchronisation:

- ▶ keep record that matches pattern
- ▶ all patterns matched: release merged records
- ▶ pattern already matched: pass through

# Nondeterminism

- Combinators in fact come in pairs
  - Serial ..
  - Choice |, and n/d choice ||
  - Replicator \*, and n/d replicator \*\*
  - Index split !, And n/d index split !!

The difference is in the output merger: in order and out of order.

# High-level features

- Boxes have type signatures  
*in-record* → out-record, ..., out-record
- The signature names fields and tags of input and output records
- The data types of fields can form hierarchies (generic components)
- Networks have inheritance properties:
  - Flow inheritance due to SISO

# Record type inference

- Signatures for networks are inferred
- Including the \* combinator
- Fast algorithms
- The multiplicity problem
  - Can approximate the proliferation factor
- Field type inference: an algebraic solution that involves semirings and graph path problems

# Semantics

- Simple denotational semantics developed for S-Net
- Defines behaviour of the \* operator
- Work underway to connect it with Broy-Stefanescu semantics of streaming functions
- Collaboration with University of Lübeck

# Implementation

- Bottom-up approach
  - Develop run-time library for dynamic unfolding using families of threads
  - Lots of hand-compiled examples
  - Develop SNet compiler
  - Network optimisation using cost calculus

# Outstanding issues (in lieu of conclusions)

- Resource management (!) for micro-threads
- Can infer finer properties: multiplicity and constancy
  - Type inference implementation
- Can support field subtyping
  - Field subtyper
- Some simple semantic formalism
- The compiler (looked at by VTT Finland)
- Graphical front-end (being developed by VTT)