Performance of a serial-bus based readout control system at Belle

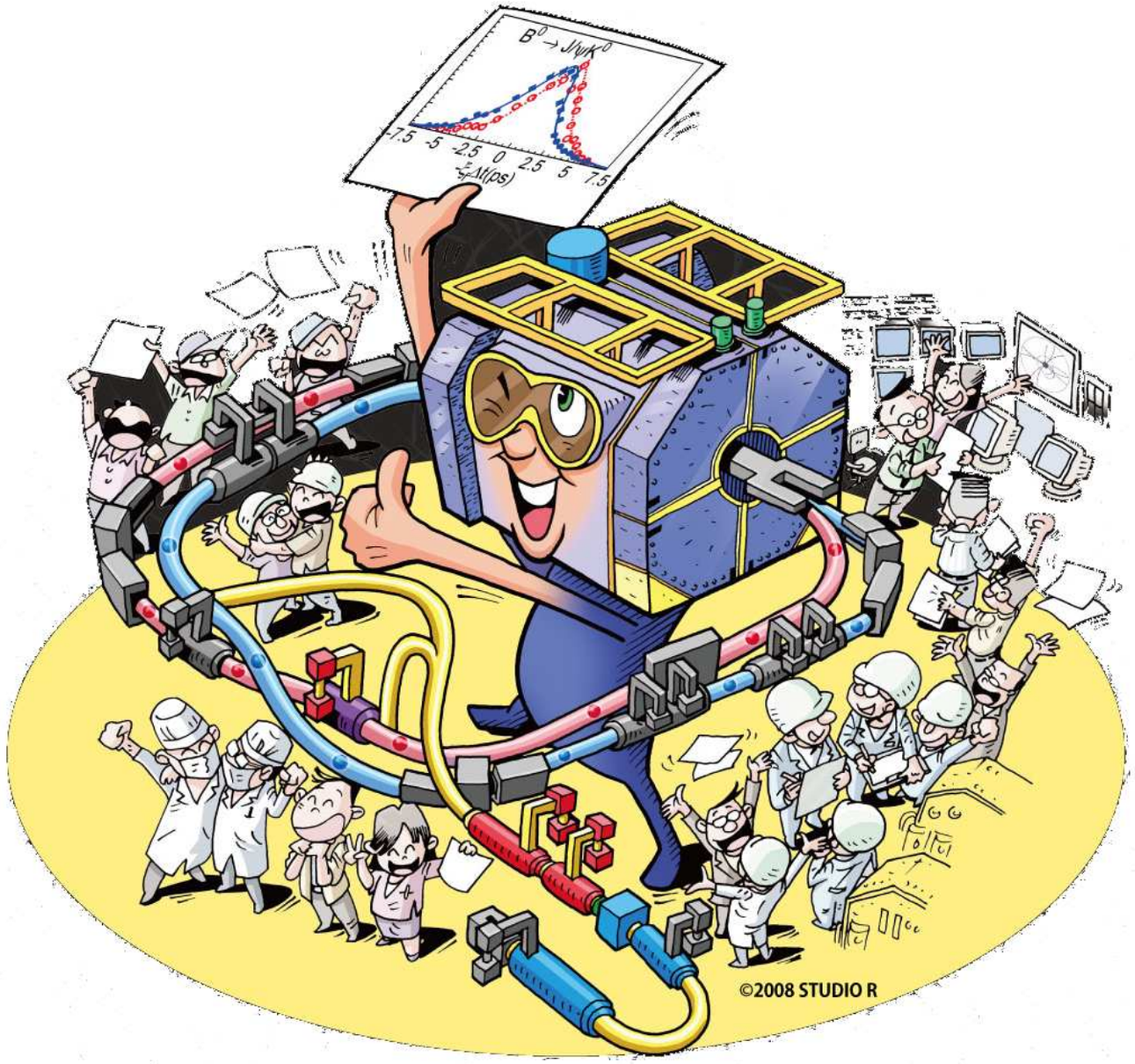
(experiences and lessons for Belle II upgrade)

Mikihiko Nakao (KEK IPNS)

May 15, 2009

16th Real Time Conference

IHEP, Beijing

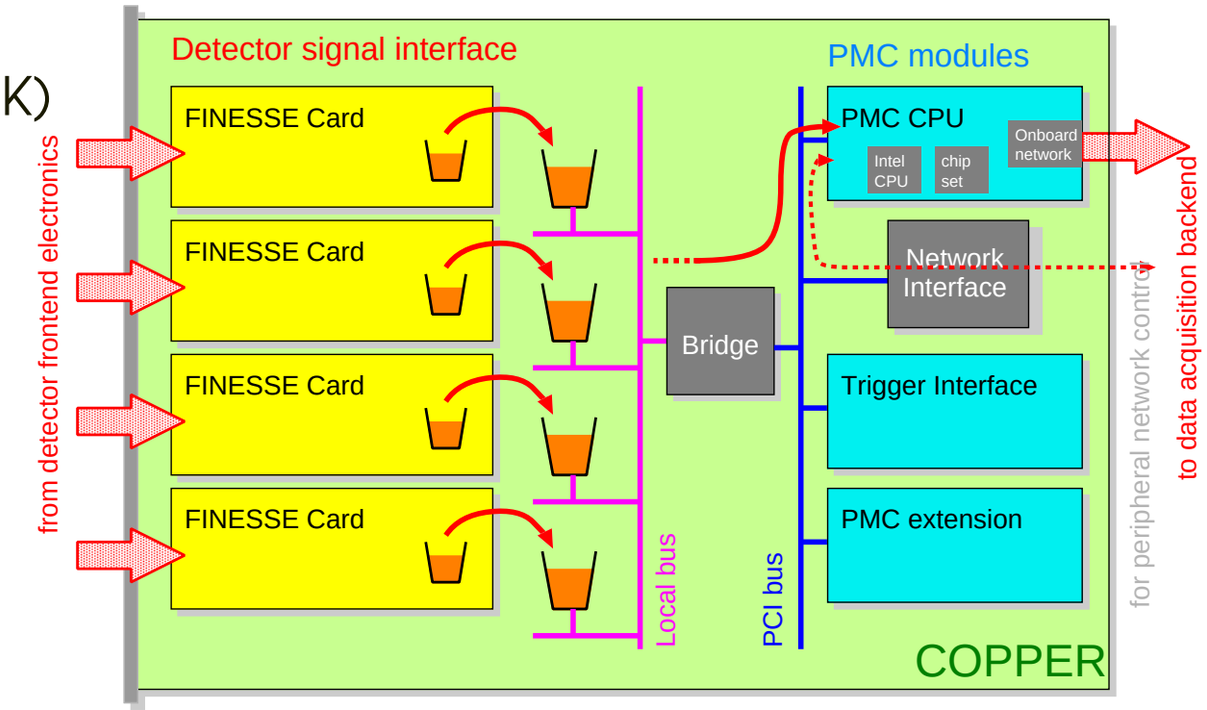


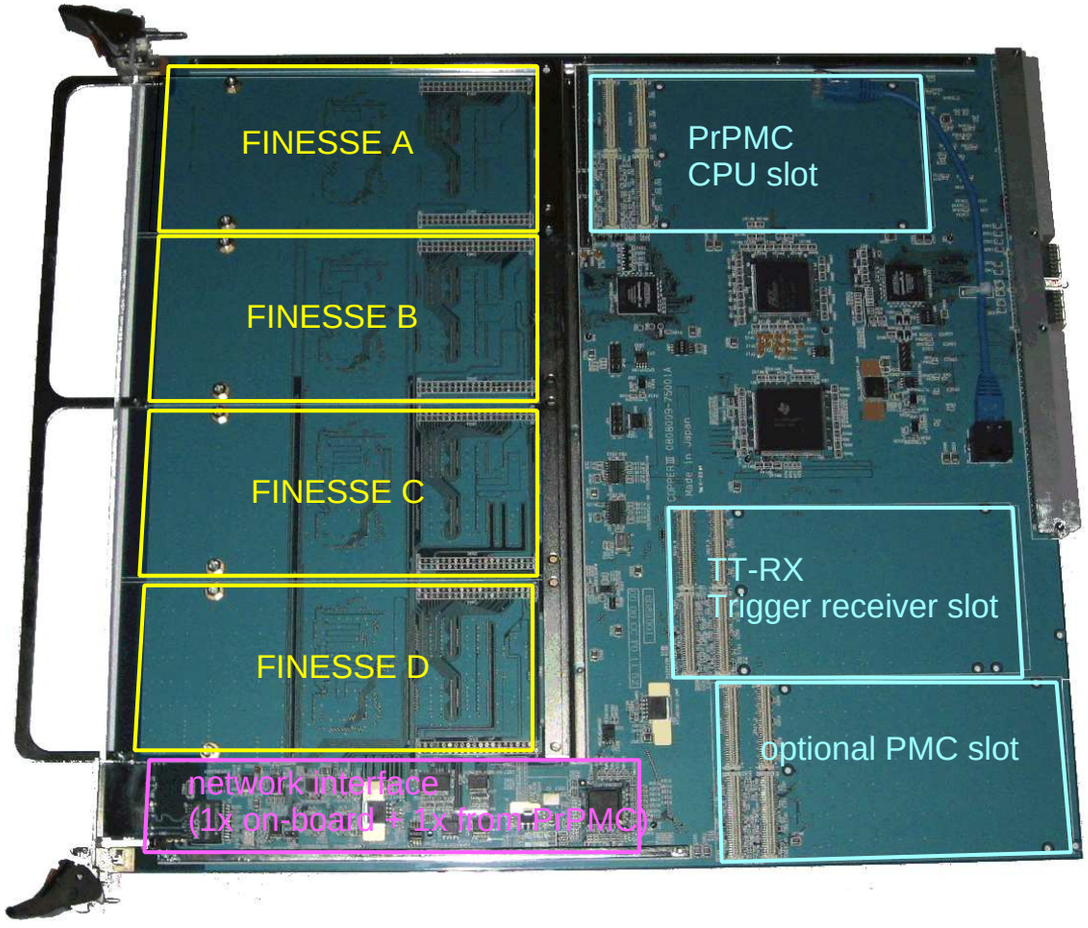
COPPER platform

(common pipeline platform developed at KEK)

Building block of Belle DAQ

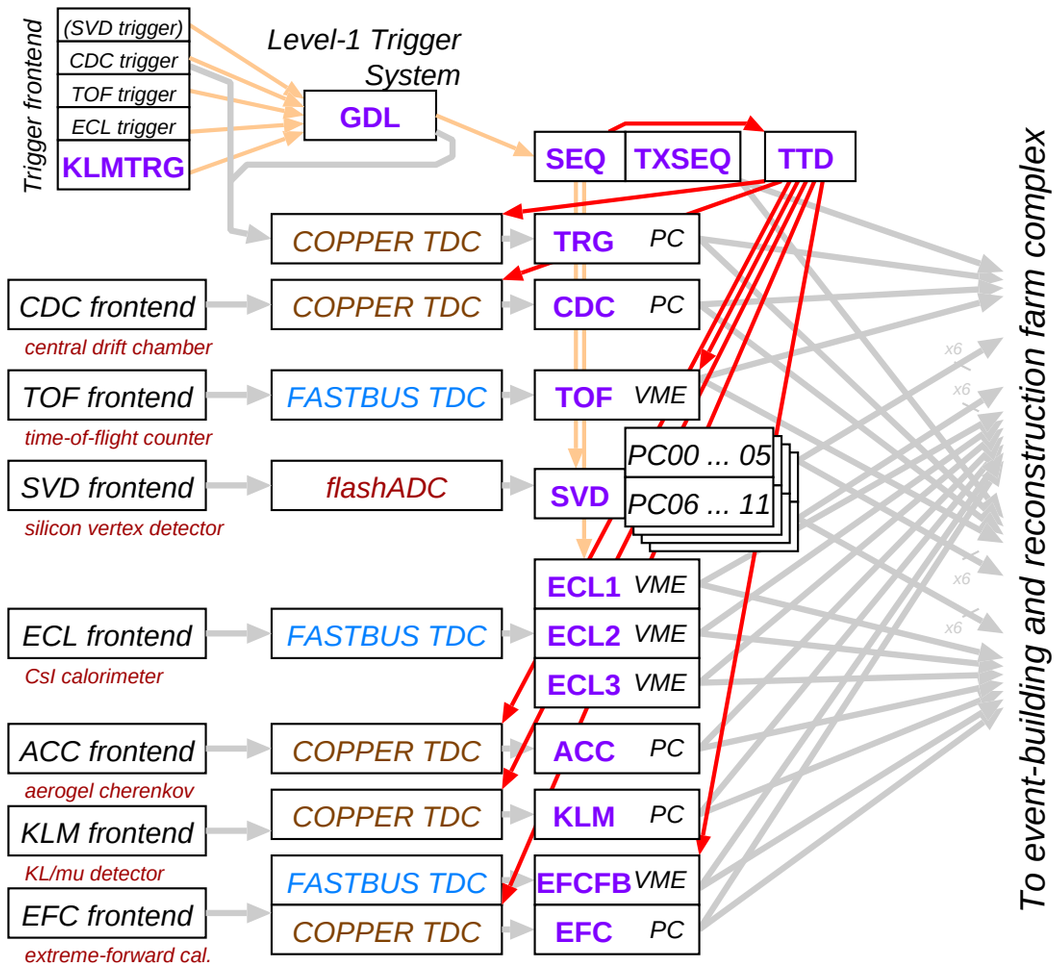
- Fully point-to-point data and control flows
  - VME9U just for power and secondary purposes
  - Data goes out through Fast/Gb Ethernet
- Modular structure
  - 4x FINESSE readout cards
  - 1x PrPMC CPU
  - 1x PMC trigger receiver
- PMC trigger receiver (TT-RX)
  - Fast control of four FINESSEs





Belle DAQ frontend in a nutshell

7 subdetectors (+ trigger info) to be read out at up to 500 Hz



- Unified Q-to-T + multi-hit TDC
  - LeCroy FASTBUS TDC
  - COPPER with AMT3 TDC
  - FlashADC for SVD (exception)
- Deadtime
  - O(a few μs) window for TDC
  - ~30μs deadtime (FASTBUS/SVD)
  - A few % deadtime fraction
  - Trigger-Busy handshake is acceptable

- old FASTBUS systems have been replaced by COPPER (signal cables are compatible)
- Readout control system for both FASTBUS/SVD and COPPER

Stepwise Belle DAQ upgrade

- History of FASTBUS to COPPER replacement
  - (2005 summer) EFC (extreme forward calorimeter) readout for testing
  - (2007 winter) CDC (drift chamber) readout was replaced
  - (2007 summer) ACC (aerogel cherenkov) readout was replaced
  - (2008 winter) TRG (trigger) readout was replaced
  - (2008 winter) TOF (time-of-flight) was connected to new timing system
  - (2009 winter) KLM (K_L and muon detector) readout was replaced (currently no urgent plan for three remaining detectors)
- Fast readout control system (TTD) was fully revised
  - Compact and scalable system
  - Clock and trigger to every (a few hundred of) COPPER boards

(Concept and initial test was reported at IEEE/NSS 2004(Rome))

Simple fast readout control scheme at Belle

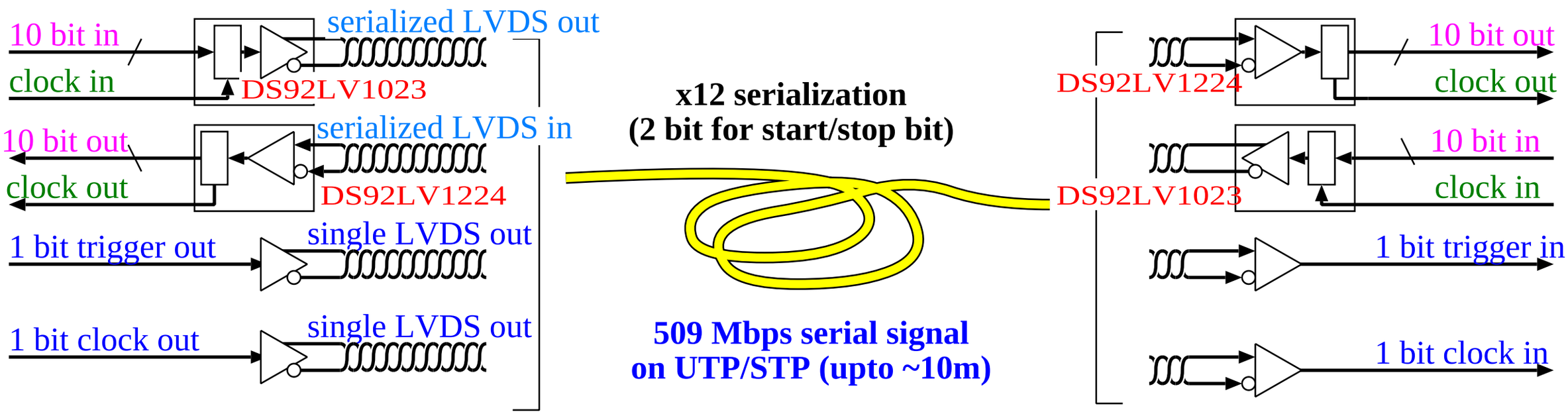
- Signals
  - System clock
  - L1 trigger
  - BUSY response (no next L1 trigger until BUSY is received and cleared)
  - "other" trigger related info
  - "other" status info

- Source and destination
  - Source: L1 trigger logic and RF clock/revolution signal from KEKB
  - Destination: ~180 COPPER boards + a few other systems

Original system with parallel cables and LEMO cables,
is not acceptable anymore

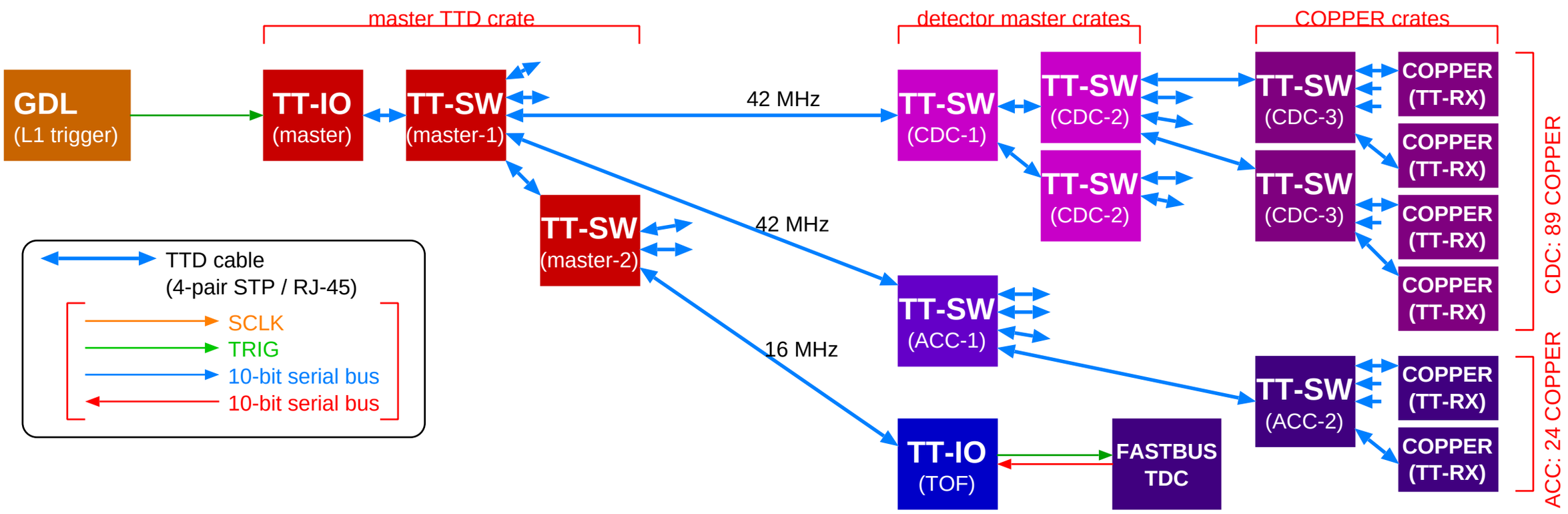
Serialized control line

- Cable
  - Shielded twisted pair LAN cable (cheap and compact)
- LVDS driver and receiver
  - Directly connected to Xilinx FPGA (Spartan3 or Virtex2Pro)
- Serializer and deserializer
  - NS DS92LV1023 and DS92LV1224, 10 bit × 42 MHz (up to 66 MHz)
  - Three modes of the 10-bit bus usage:
    Run mode, Counter mode and Command sequence



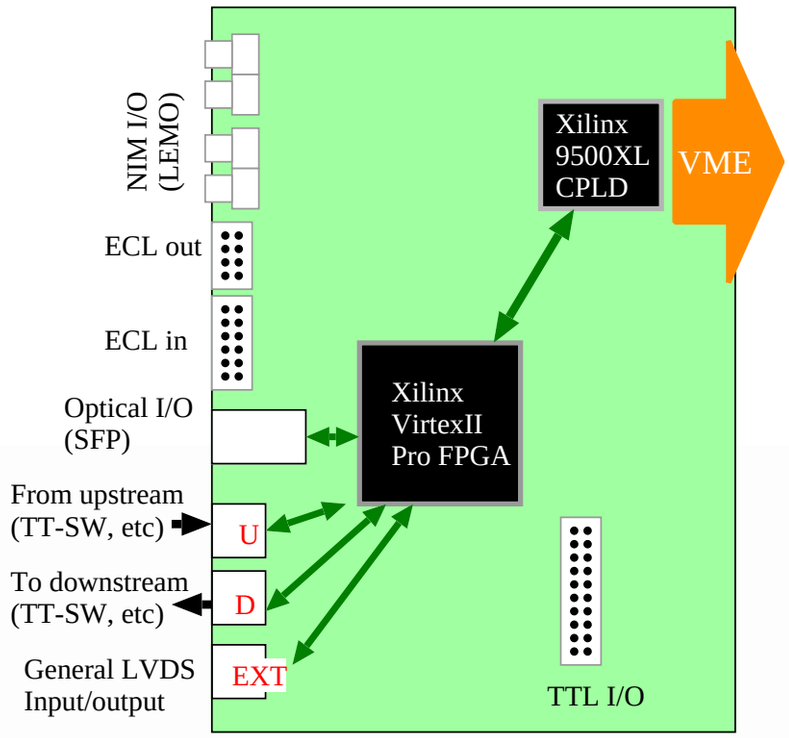
Cascadable distribution module

- Cascading single-type one-to-eight switch modules (TT-SW)
- Up to four-stage cascading:
  - First stage to partition sub-detectors
  - Three more stages to the largest sub-detector (89 COPPERs)
- Only two more types of modules
  - Master (multi-purpose) module (TT-IO), Receiver (TT-RX)

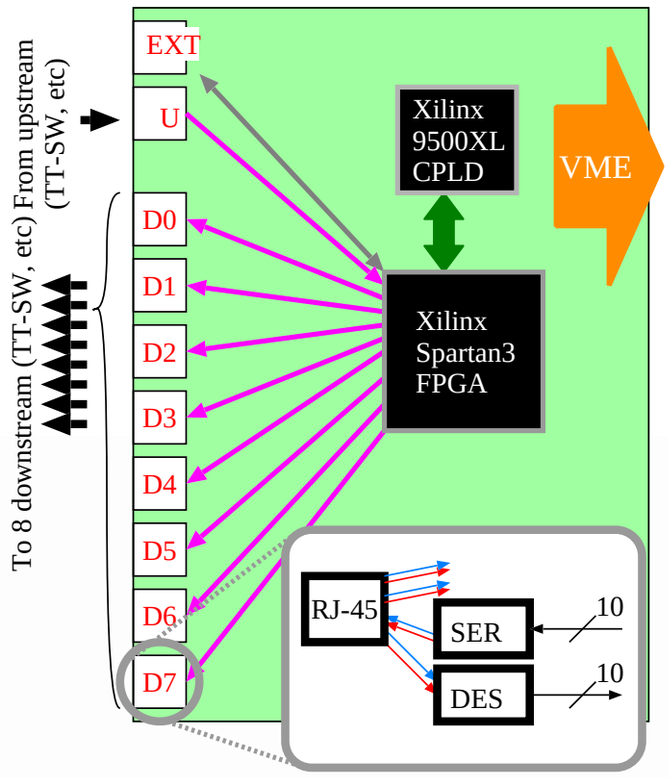


TT-IO and TT-SW are VME6U for a compact distribution system

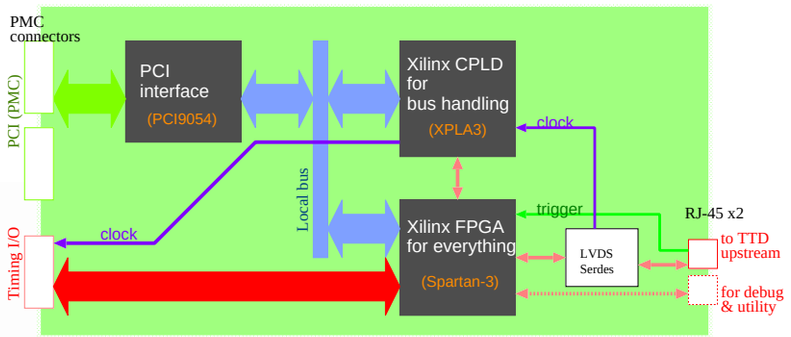
TT-IO (VME6U)

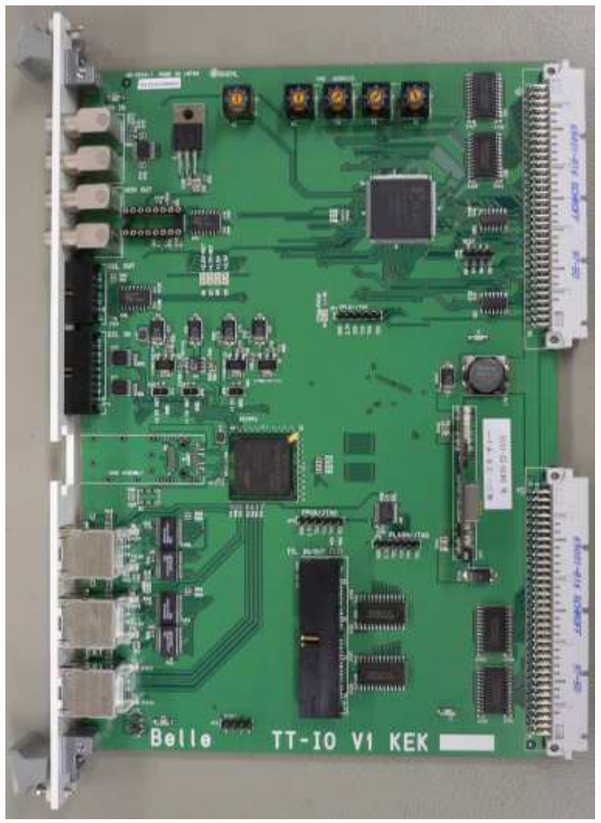


TT-SW (VME6U)

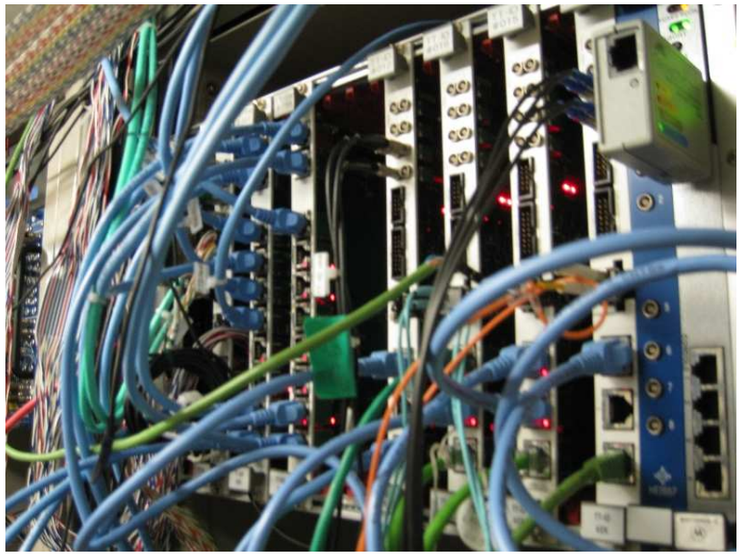


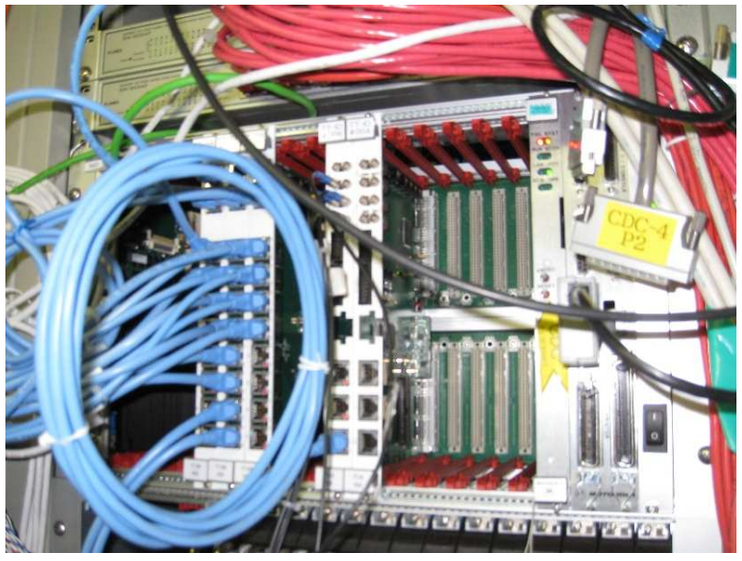
TT-RX (PMC)

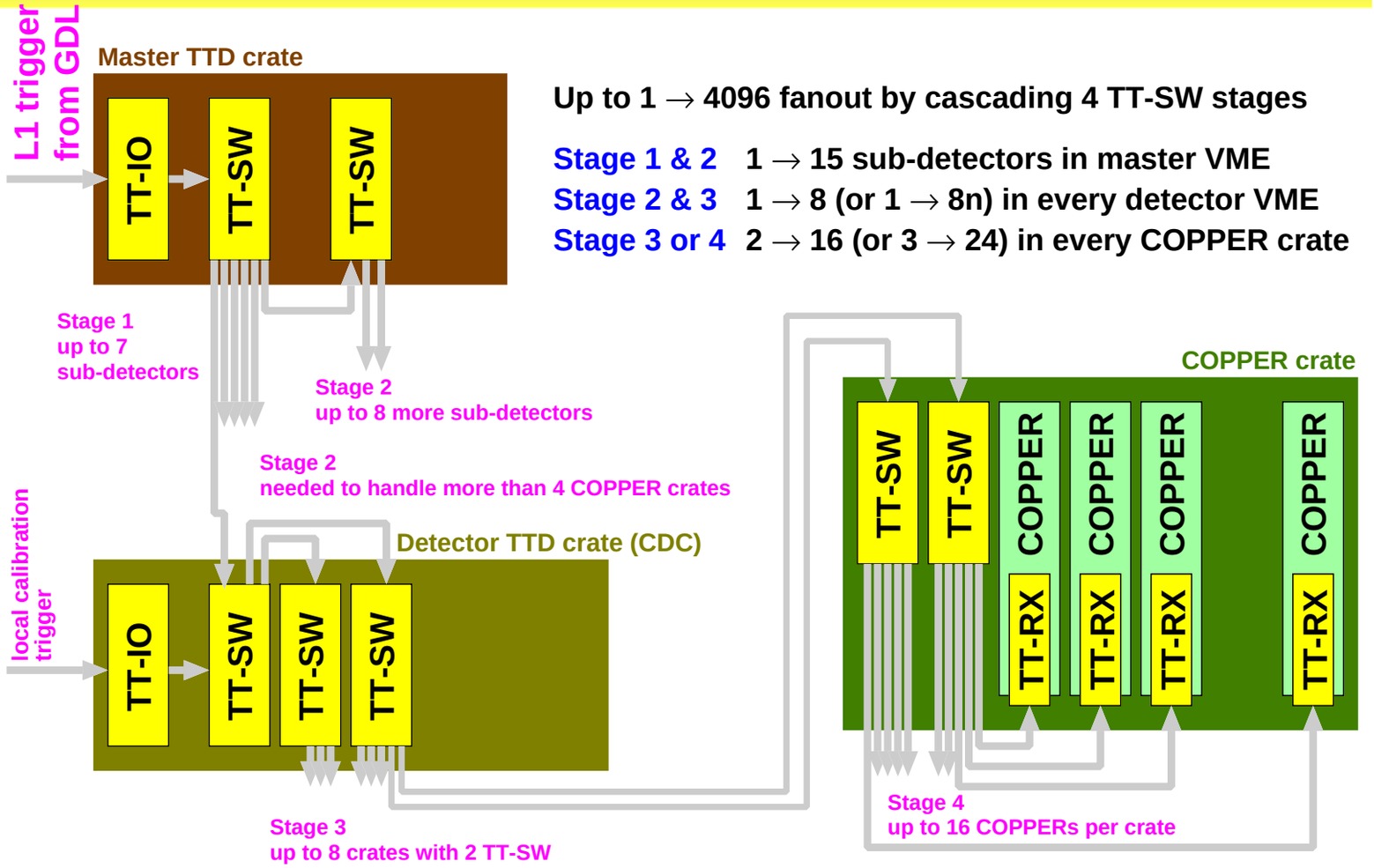




TTD modules in Belle Electronics-Hut

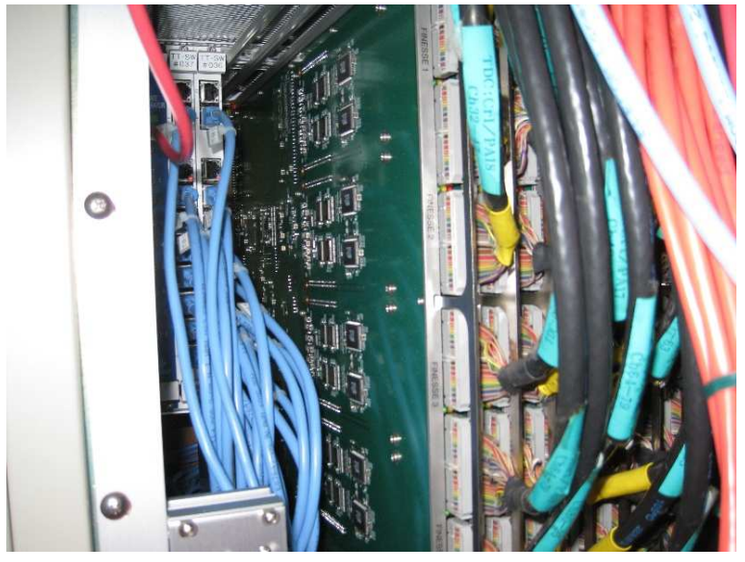


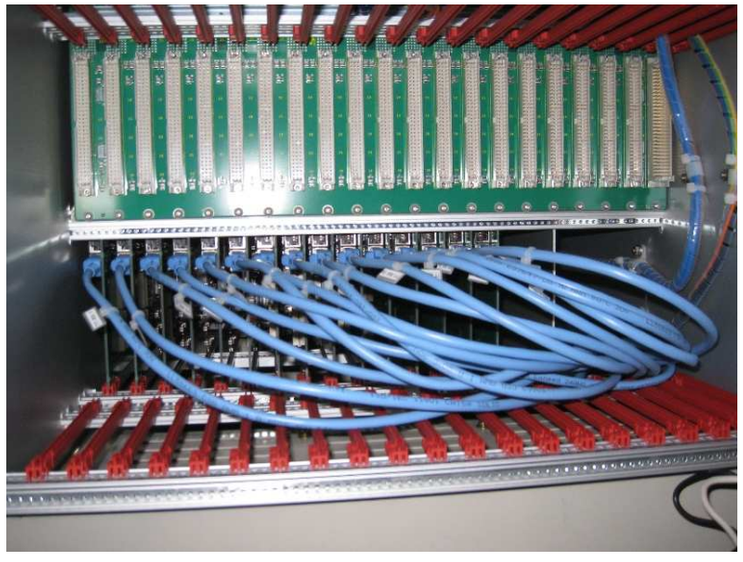




Up to 1 → 4096 fanout by cascading 4 TT-SW stages

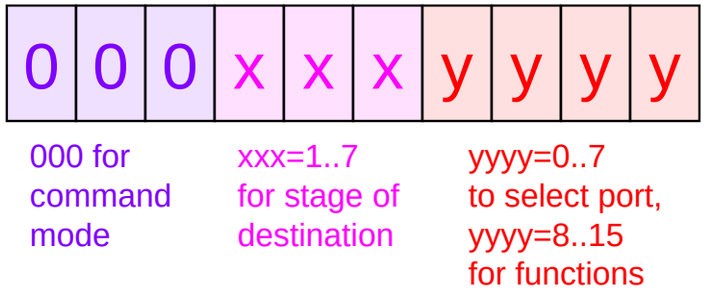
Stage 1 & 2	1 → 15 sub-detectors in master VME
Stage 2 & 3	1 → 8 (or 1 → 8n) in every detector VME
Stage 3 or 4	2 → 16 (or 3 → 24) in every COPPER crate





Command sequence

- 3-bit to identify command sequence
- 3-bit for the destination level
  - up to 7-stage cascade
  - 3-bit level is decremented at every step
- 4-bit (up to 16) functions
  - 8 of them are to select the destination of one-to-eight switch
  - 8 more actions are defined



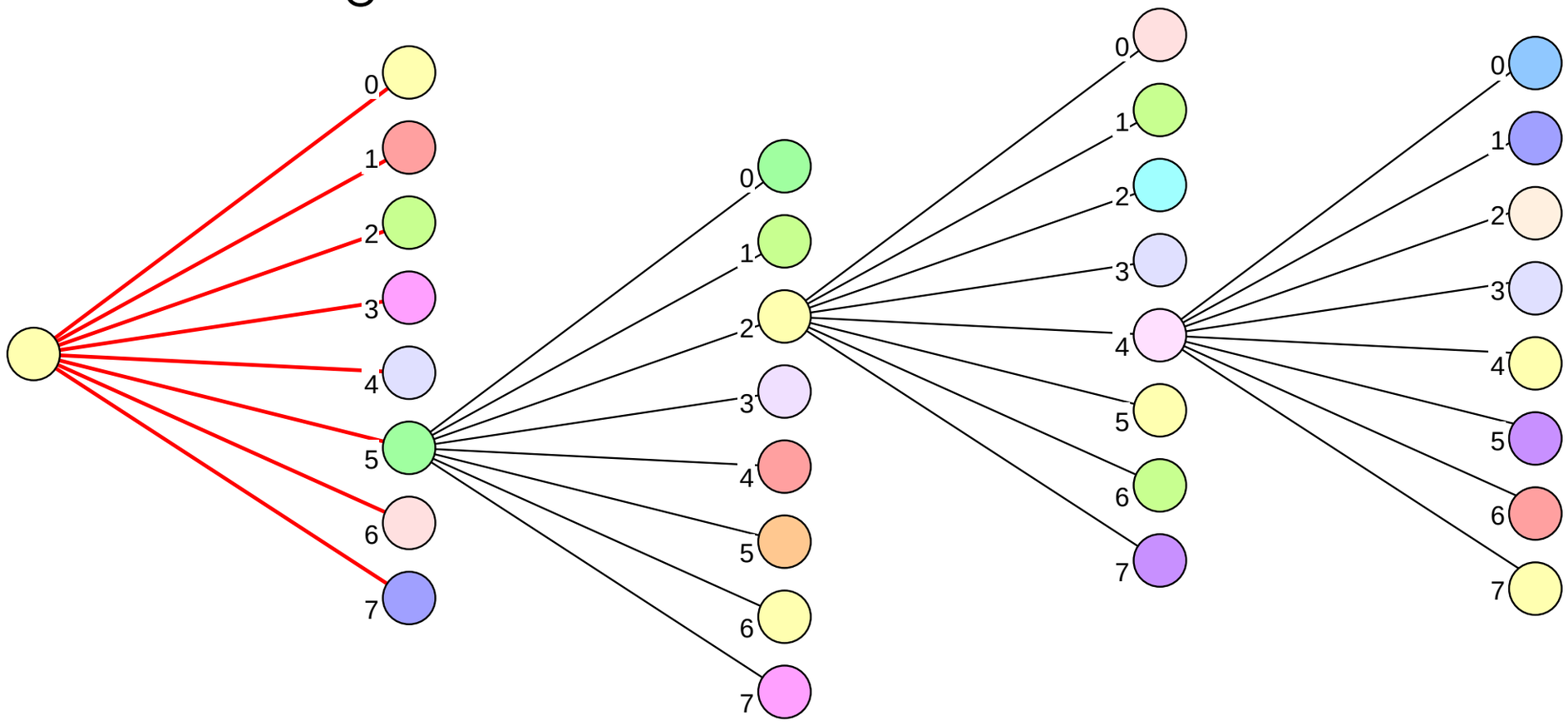

```
constant TCMD_NOP     : tcmd_t := "00000000";
constant TCMD_SELKID  : tcmd_t := "00010000";  -- 0x10..7
constant TCMD_TCLEAR  : tcmd_t := "00011000";  -- 0x18
constant TCMD_DMASK   : tcmd_t := "00011001";  -- 0x19
constant TCMD_BSYIN   : tcmd_t := "00011010";  -- 0x1a
constant TCMD_NOBSY   : tcmd_t := "00011011";  -- 0x1b
constant TCMD_EXBSY   : tcmd_t := "00011100";  -- 0x1c
constant TCMD_PASMODE : tcmd_t := "00011101";  -- 0x1d
constant TCMD_RETMODE : tcmd_t := "00011110";  -- 0x1e
constant TCMD_SRST    : tcmd_t := "00011111";  -- 0x1f
```

Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

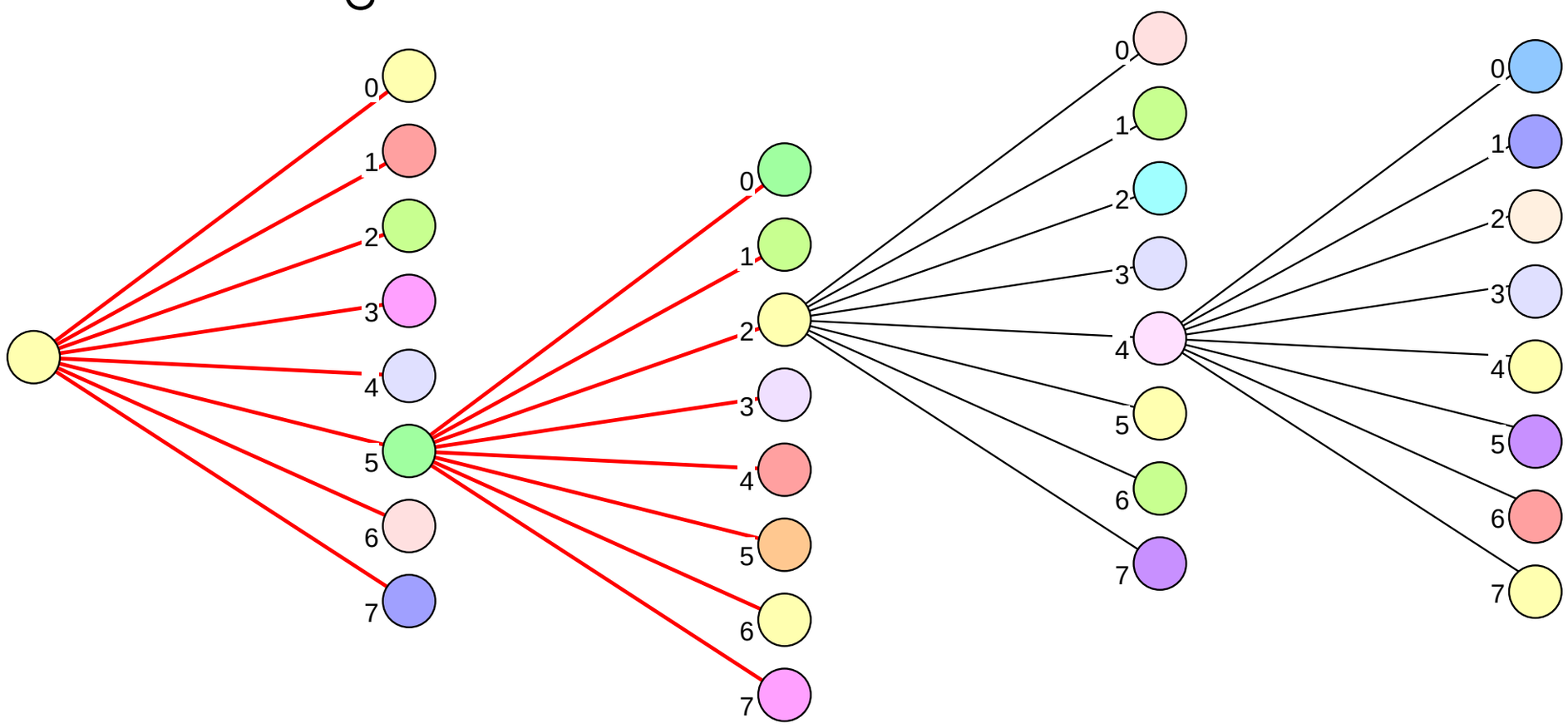


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

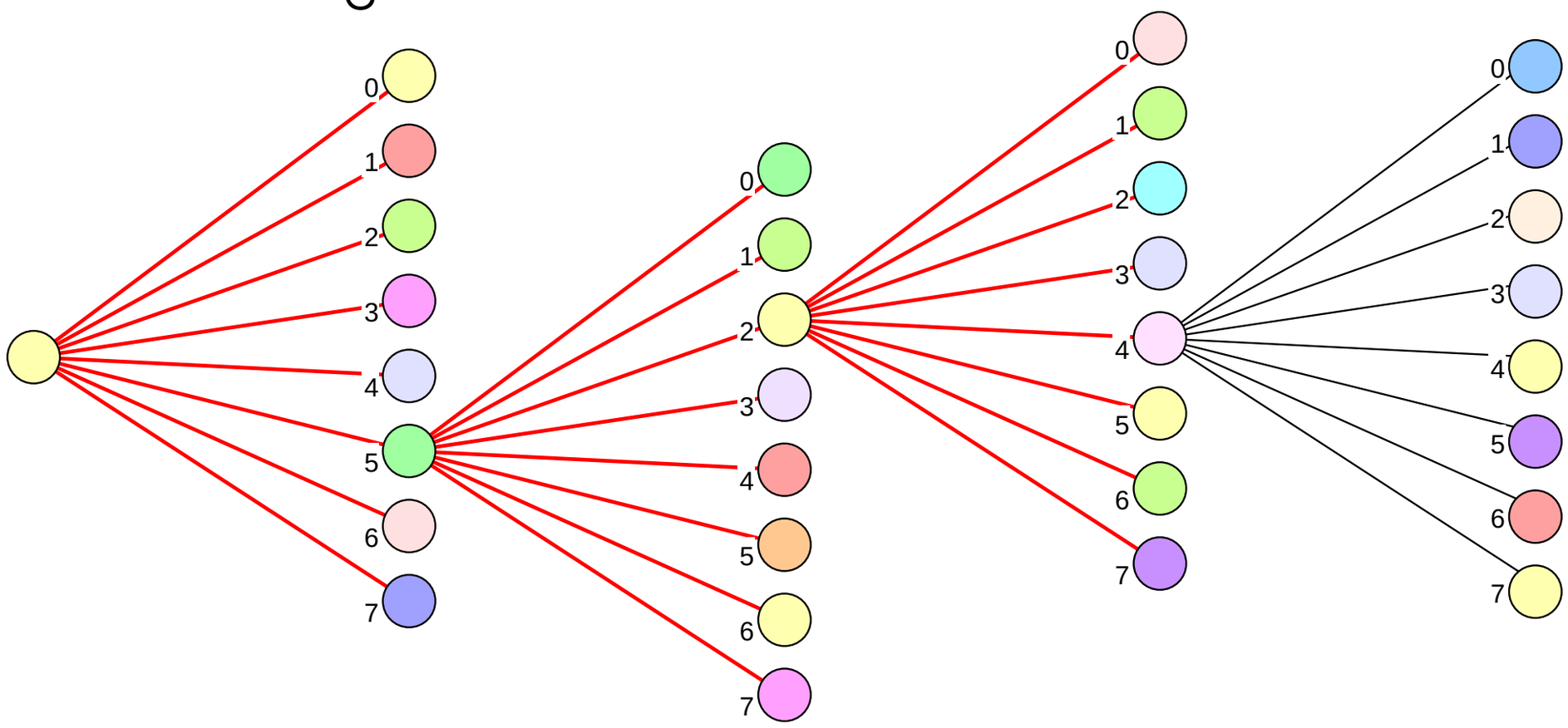


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

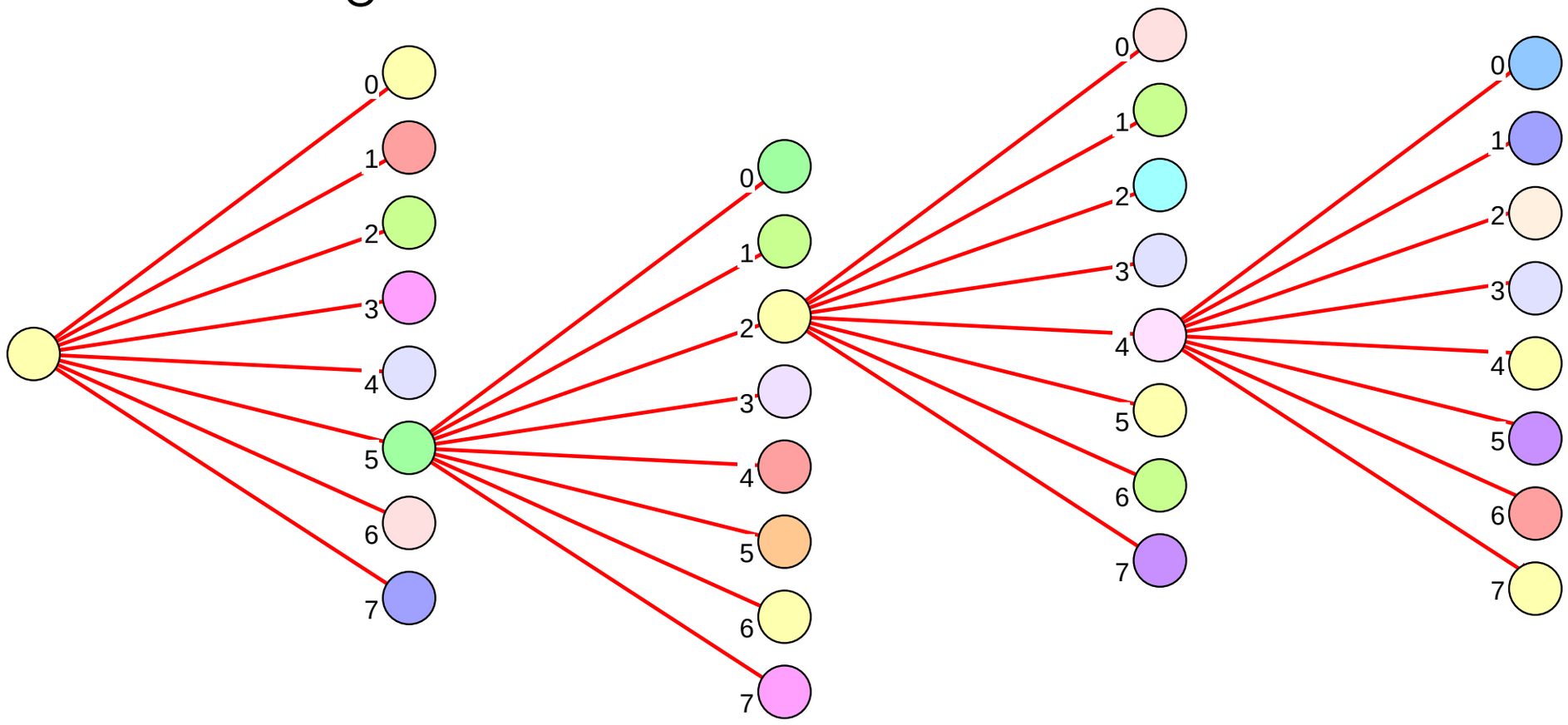


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

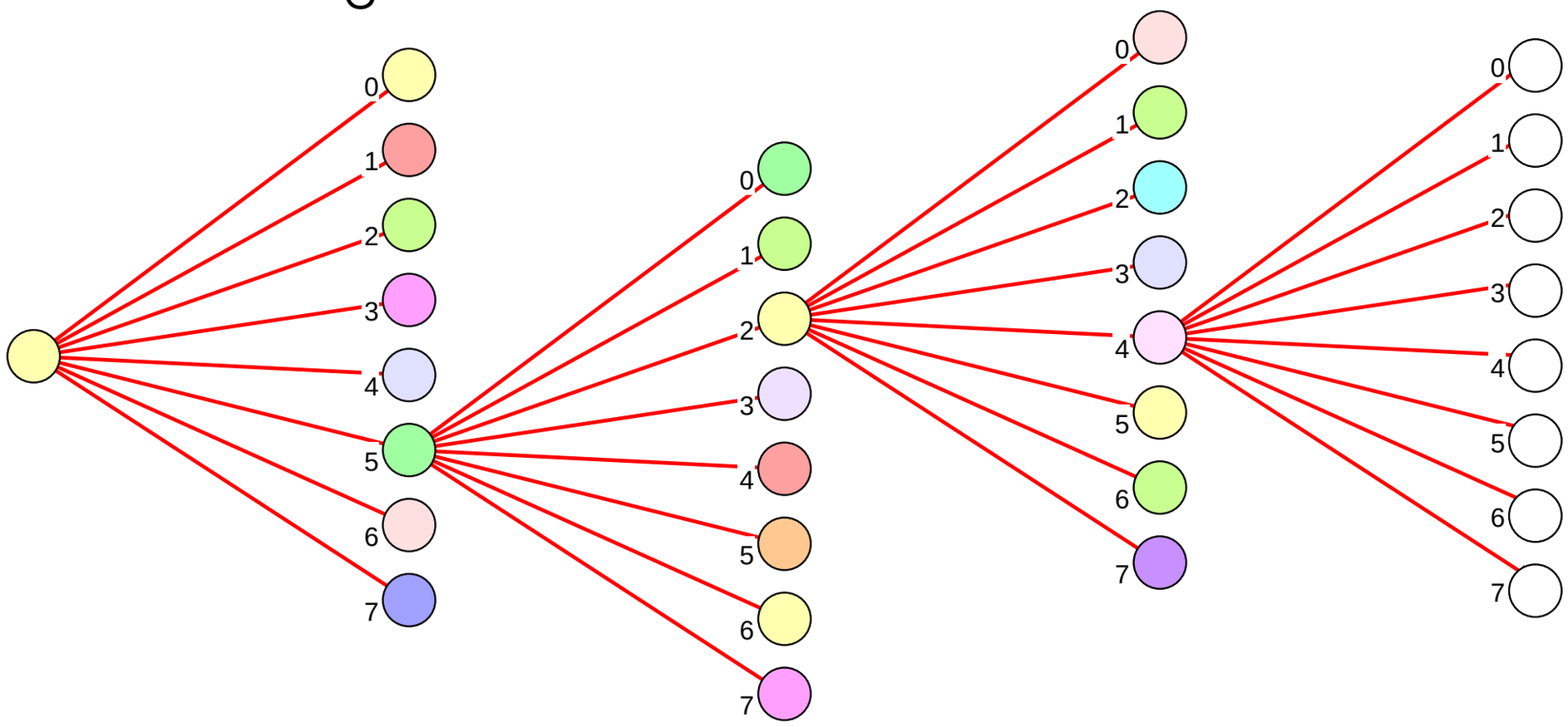


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

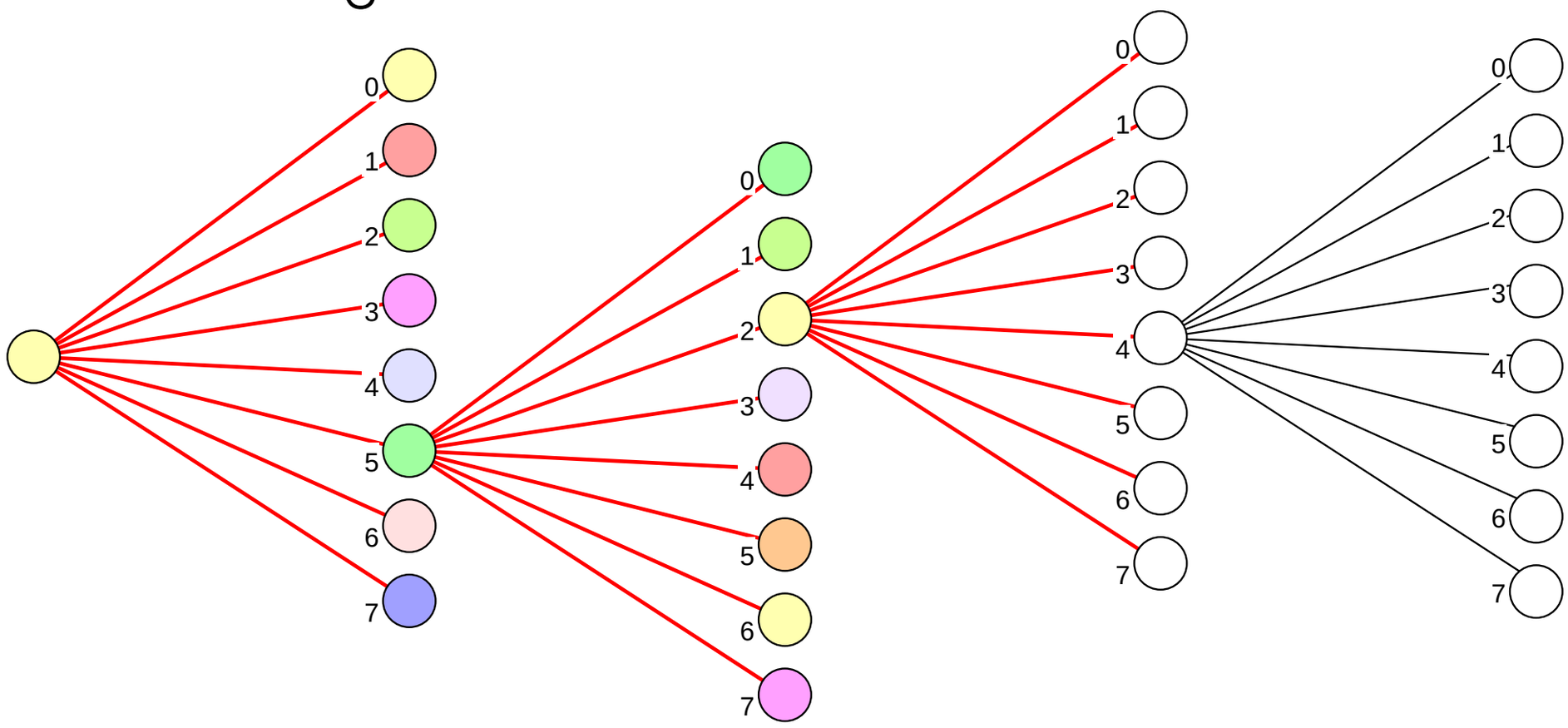


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

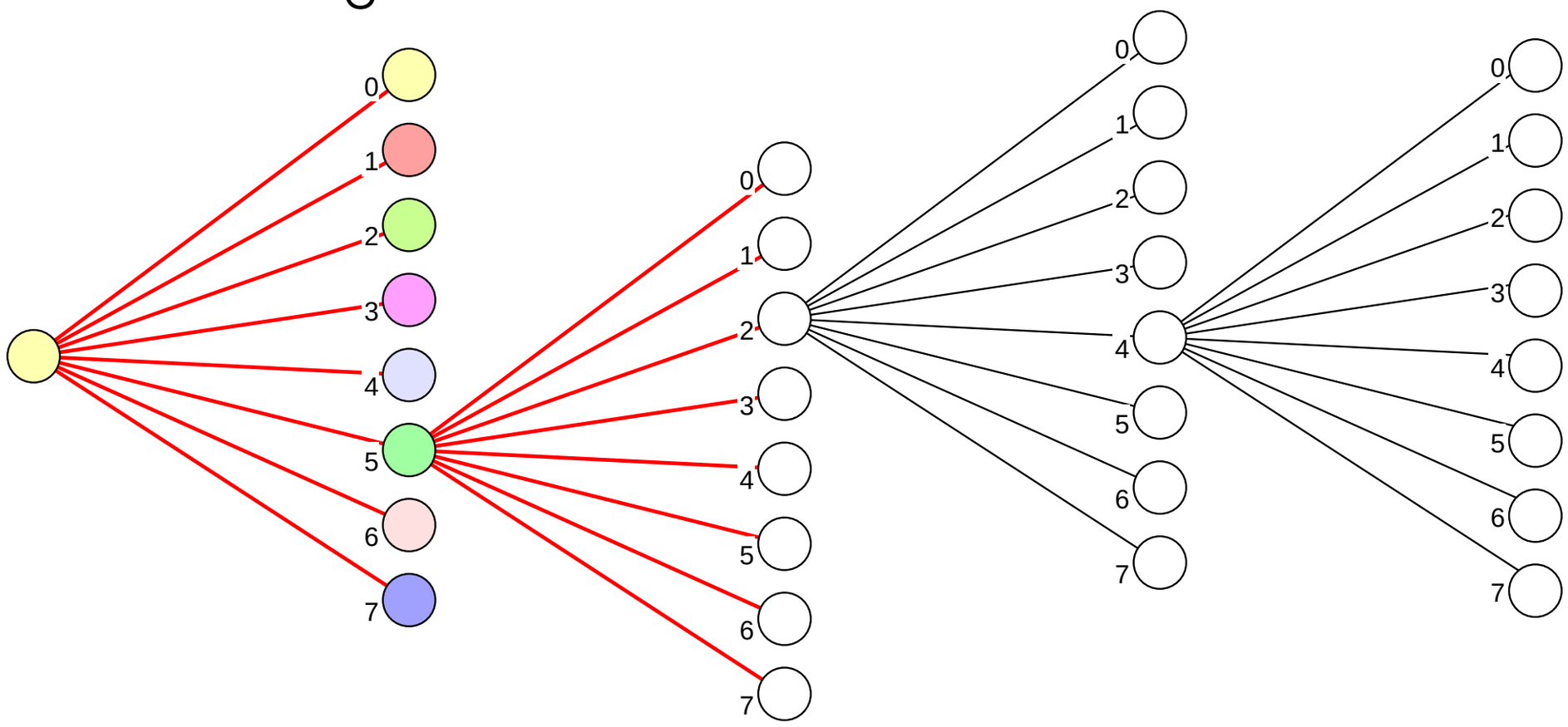


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

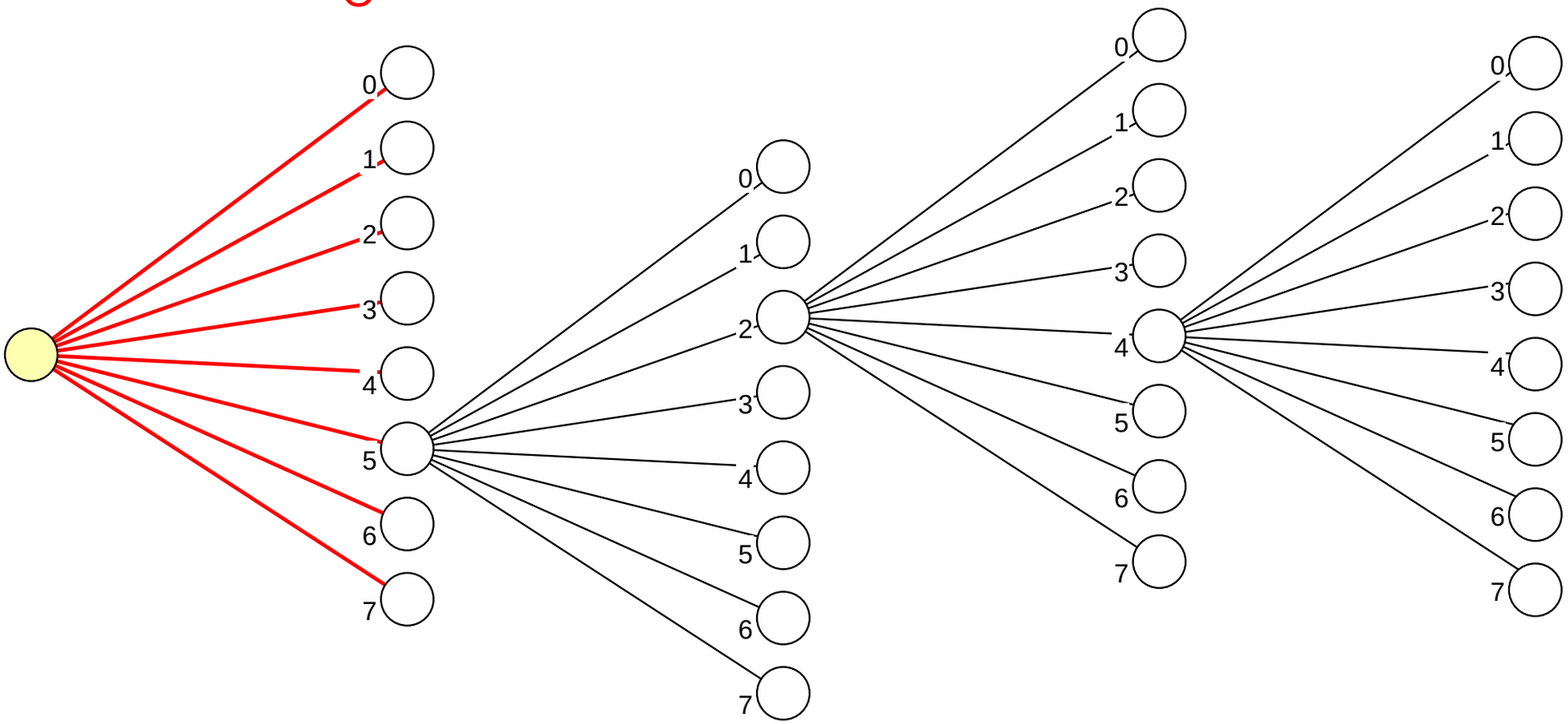


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

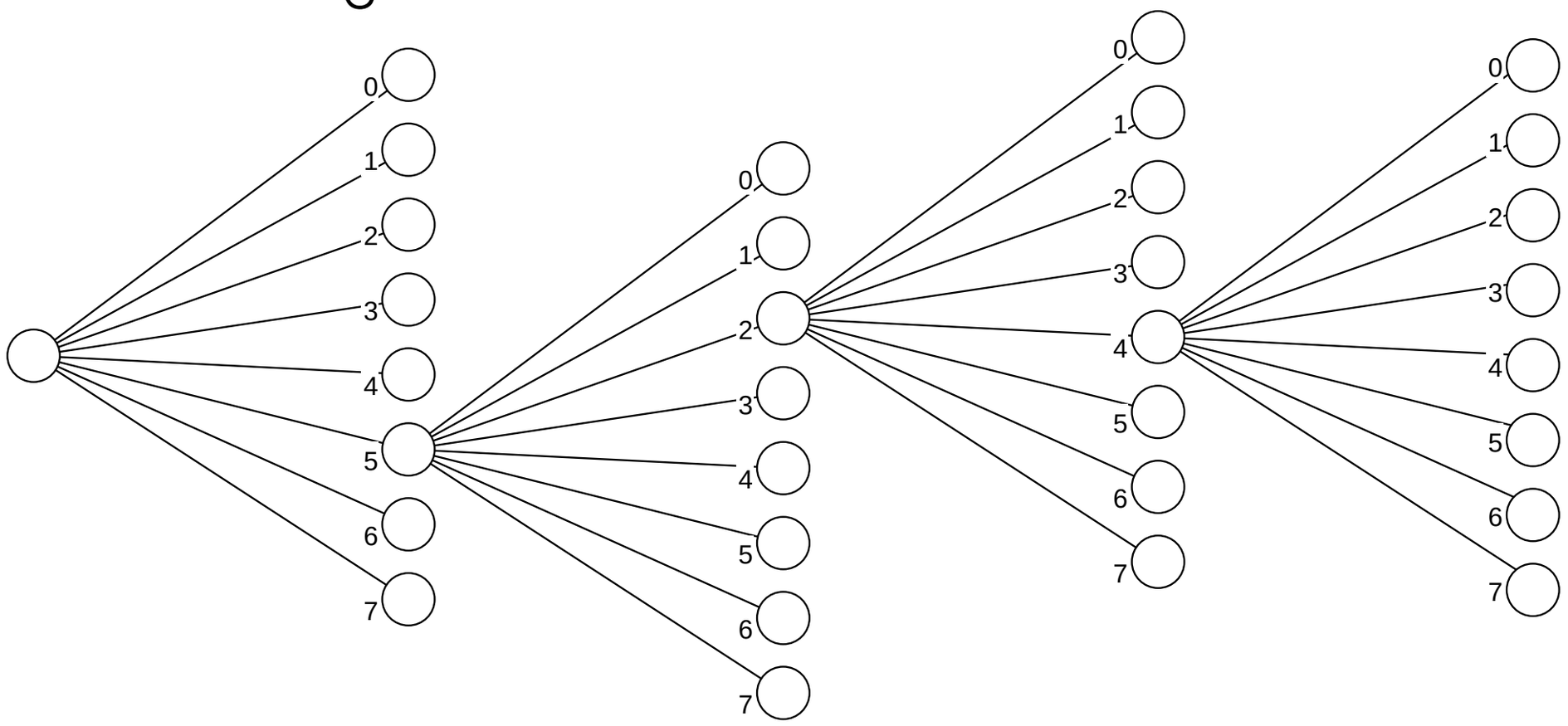


Example1: broadcast a reset signal

Downward data bus is common for all 8 ports

- set pass-through mode at stage 1
- set pass-through mode at stage 2
- set pass-through mode at stage 3
- issue reset for stage 4
- issue reset for stage 3
- issue reset for stage 2
- issue reset for stage 1

"Reset" will reset
the pass-through mode
to the run mode

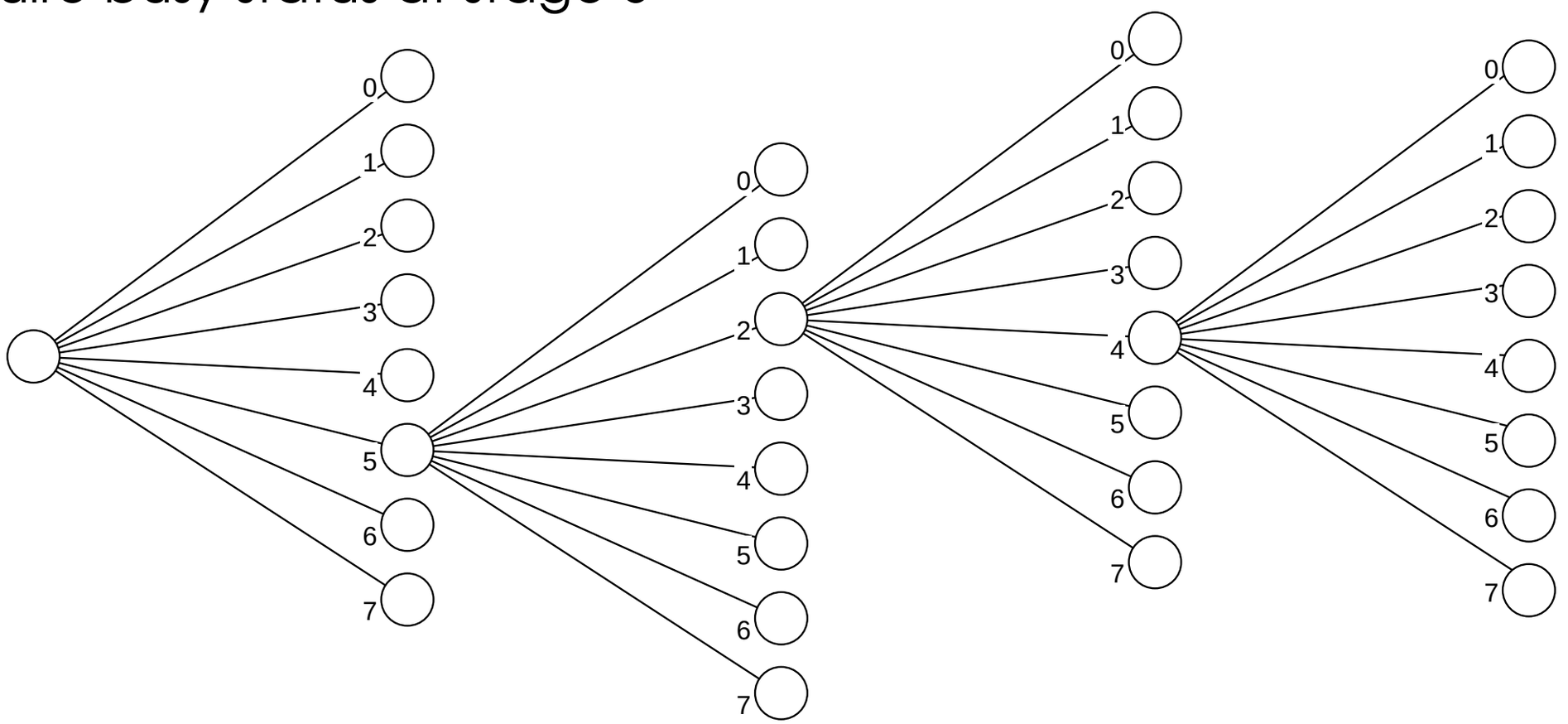


Example2: to a single destination node

- select node 5 at stage 1
- set mask at stage 1
- select node 2 at stage 2
- set mask at stage 2
- select node 4 at stage 3
- set mask at stage 3
- set pass-through mode at stage 1
- set pass-through mode at stage 2
- inquire busy status at stage 3

Downward data bus is common for all 8 ports, but each port can be disabled

i.e., set to "sync" state, so serial bus synchronization is kept

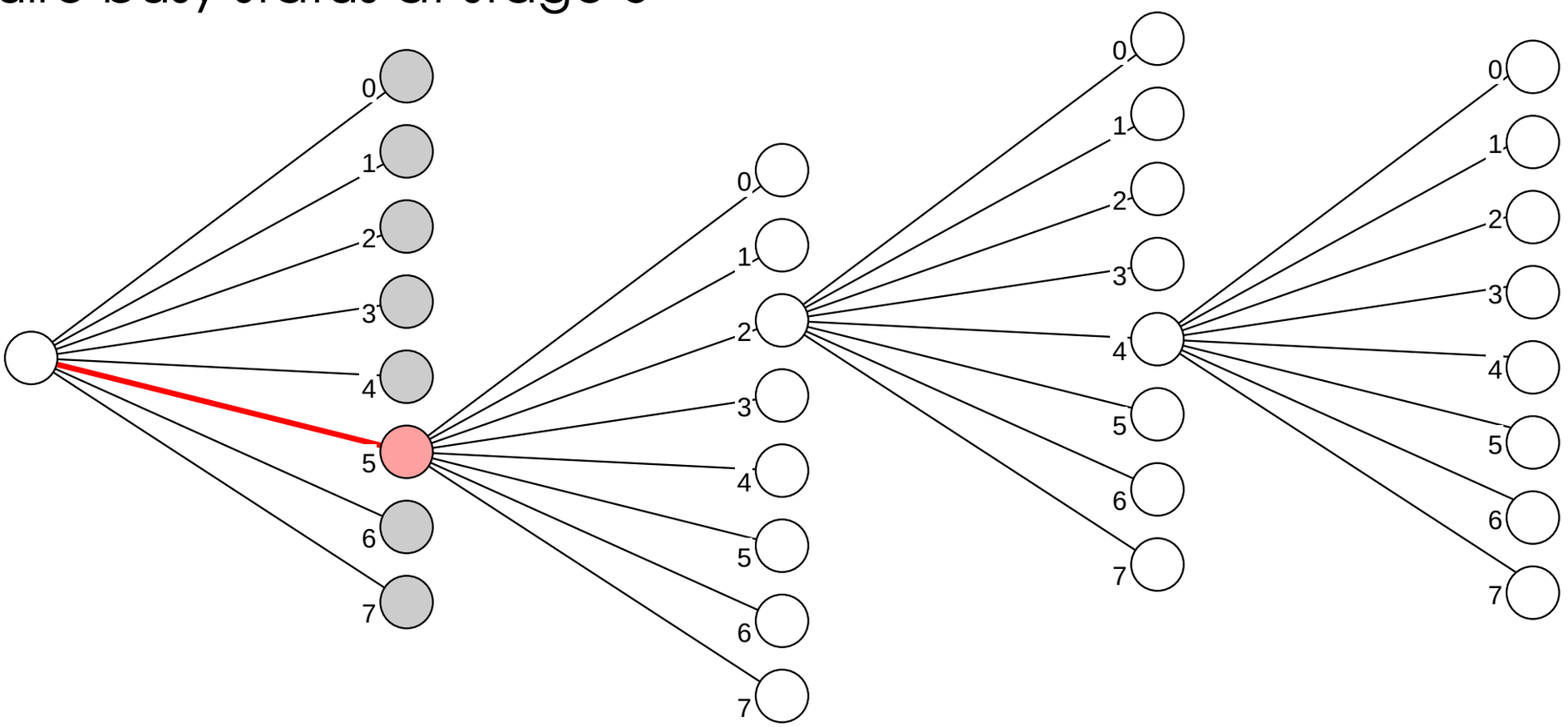


Example2: to a single destination node

- select node 5 at stage 1
- set mask at stage 1
- select node 2 at stage 2
- set mask at stage 2
- select node 4 at stage 3
- set mask at stage 3
- set pass-through mode at stage 1
- set pass-through mode at stage 2
- inquire busy status at stage 3

Downward data bus is common for all 8 ports, but each port can be disabled

i.e., set to "sync" state, so serial bus synchronization is kept

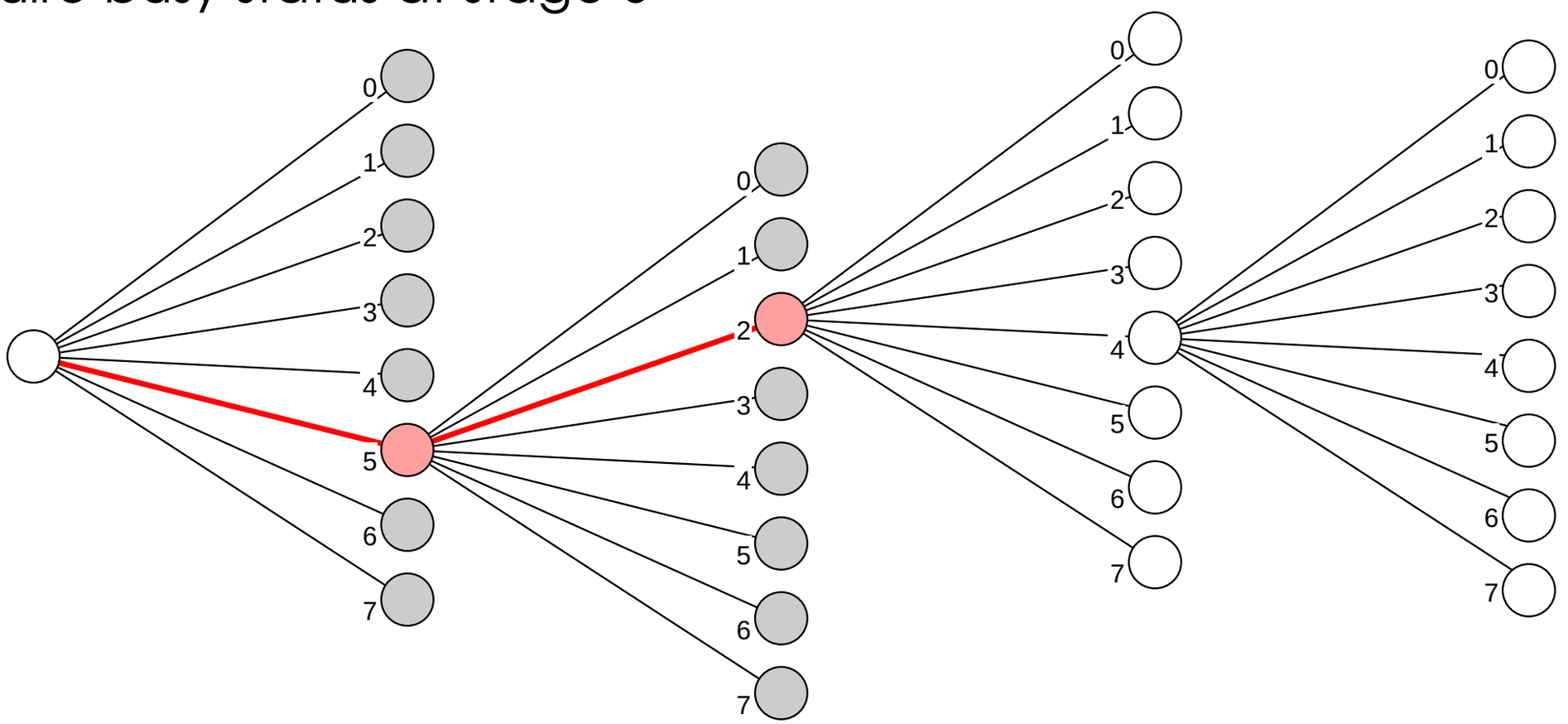


Example2: to a single destination node

- select node 5 at stage 1
- set mask at stage 1
- select node 2 at stage 2
- set mask at stage 2
- select node 4 at stage 3
- set mask at stage 3
- set pass-through mode at stage 1
- set pass-through mode at stage 2
- inquire busy status at stage 3

Downward data bus is common for all 8 ports, but each port can be disabled

i.e., set to "sync" state, so serial bus synchronization is kept

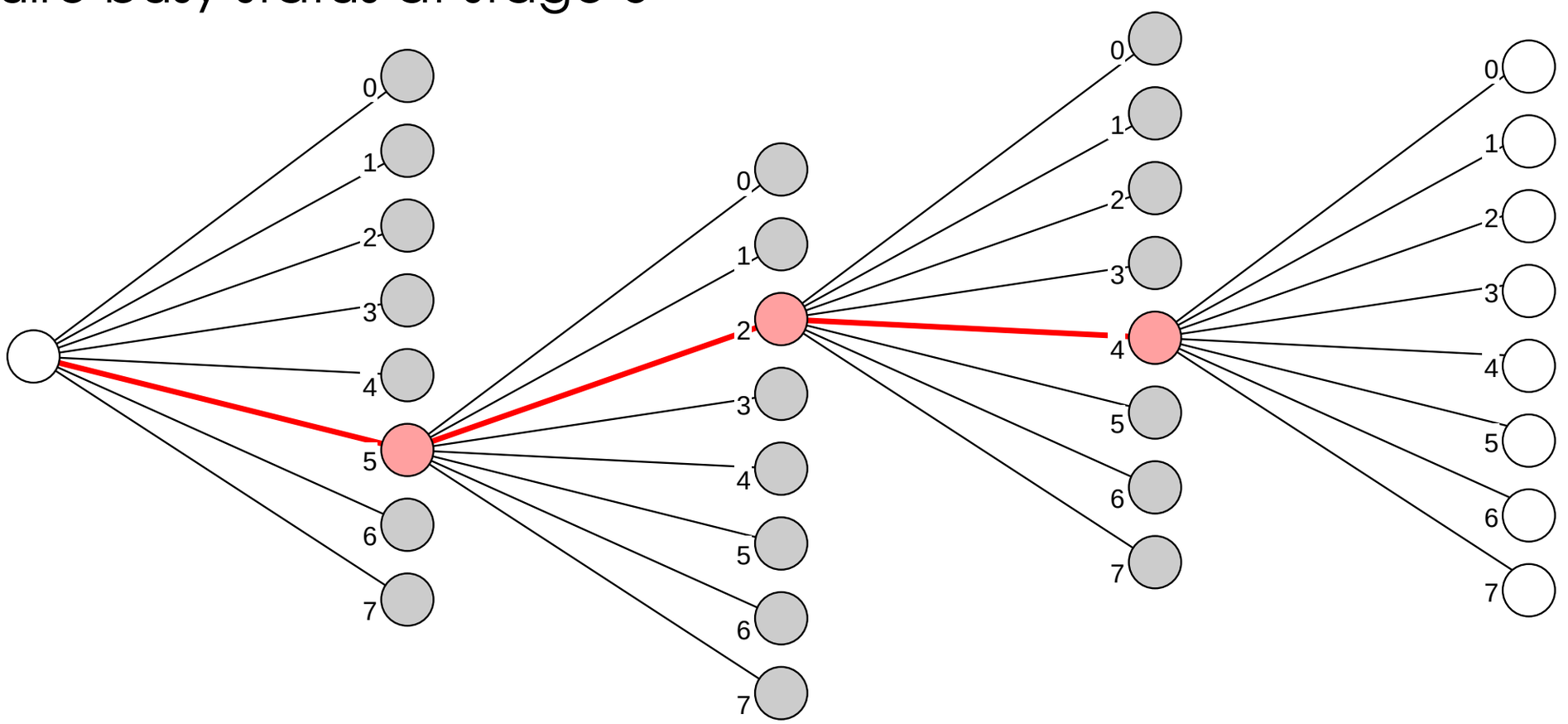


Example2: to a single destination node

- select node 5 at stage 1
- set mask at stage 1
- select node 2 at stage 2
- set mask at stage 2
- select node 4 at stage 3
- set mask at stage 3
- set pass-through mode at stage 1
- set pass-through mode at stage 2
- inquire busy status at stage 3

Downward data bus is common for all 8 ports, but each port can be disabled

i.e., set to "sync" state, so serial bus synchronization is kept

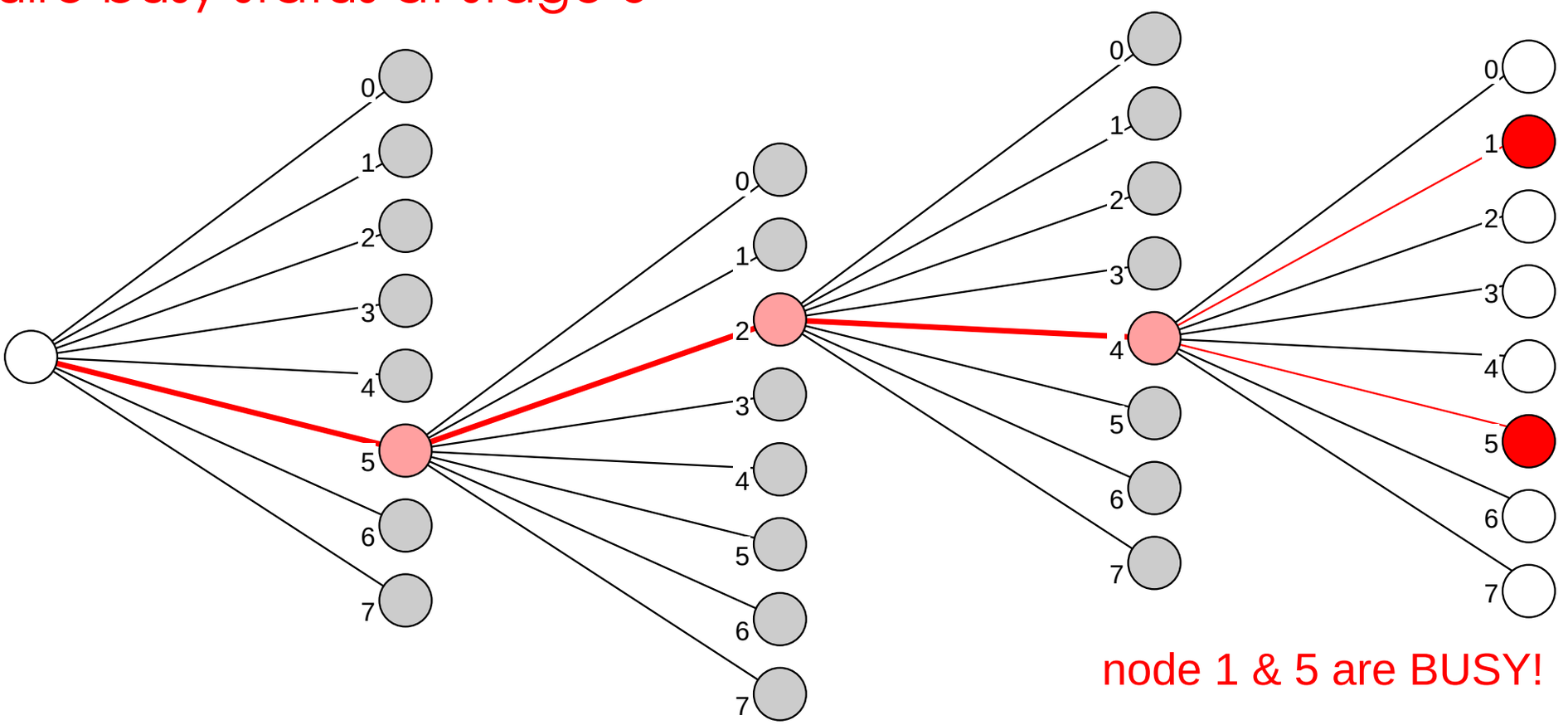


Example2: to a single destination node

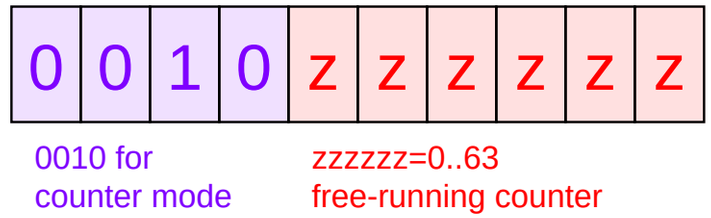
- select node 5 at stage 1
- set mask at stage 1
- select node 2 at stage 2
- set mask at stage 2
- select node 4 at stage 3
- set mask at stage 3
- set pass-through mode at stage 1
- set pass-through mode at stage 2
- inquire busy status at stage 3

Downward data bus is common for all 8 ports, but each port can be disabled

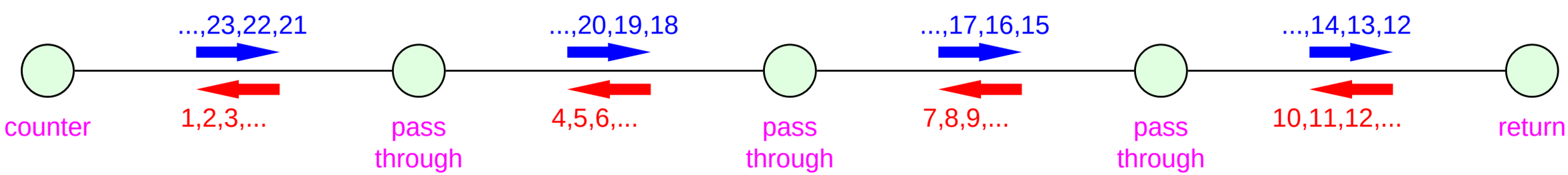
i.e., set to "sync" state, so serial bus synchronization is kept



Counter mode



- Round-trip counter
  - Master node runs a 6-bit counter
  - Intermediate nodes pass through down/up-wards word
  - The last node just returns what was received
  - Master node calculate the difference = round-trip latency
- Checked for every node at run start
  - Identifies faulty/unstable connections
  - Nearly random sequence — stabilizes the serial bus
- Latency is measured in situ
  - The slowest (CDC, 4 stages) takes 66 clocks for round-trip
  - Busy collection time of 780ns



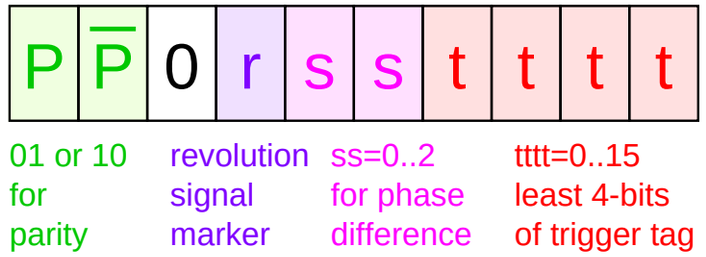
```
IO->SW delay=12
-(stage1:1)-
-(stage2:1)-
IO->SW8->SW(TOF)1->RX delay=41
-(stage3:5)-
--
IO->SW2->SW(ACC)5->SW[2345678]->RX delay=55
IO->SW2->SW(ACC)6->SW[2345678]->RX delay=55
IO->SW2->SW(ACC)7->SW[45678]->RX delay=55
IO->SW2->SW(ACC)8->SW[45678]->RX delay=55
--
IO->SW3->SW(TRG)5->SW[2345678]->RX delay=55
IO->SW3->SW(TRG)6->SW[2345678]->RX delay=55
IO->SW3->SW(TRG)7->SW[345678]->RX delay=55
IO->SW3->SW(TRG)8->SW[145678]->RX delay=55
--
IO->SW4->SW(KLM)1->SW[45678]->RX delay=55
IO->SW4->SW(KLM)2->SW[45678]->RX delay=55
IO->SW4->SW(KLM)3->SW[45678]->RX delay=55
IO->SW4->SW(KLM)4->SW[45678]->RX delay=55
IO->SW4->SW(KLM)5->SW[2345678]->RX delay=55
IO->SW4->SW(KLM)6->SW[2345678]->RX delay=55
--
```

```
-(stage4:86)-
IO->SW1->SW(CDC)1->SW1->SW[12345678]->RX delay=2
IO->SW1->SW(CDC)1->SW2->SW[2345678]->RX delay=2
IO->SW1->SW(CDC)1->SW3->SW[12345678]->RX delay=2
IO->SW1->SW(CDC)1->SW4->SW[2345678]->RX delay=2
IO->SW1->SW(CDC)1->SW5->SW[12345678]->RX delay=2
IO->SW1->SW(CDC)1->SW6->SW[2345678]->RX delay=2
IO->SW1->SW(CDC)1->SW7->SW[12345678]->RX delay=2
IO->SW1->SW(CDC)1->SW8->SW[2345678]->RX delay=2
--
IO->SW1->SW(CDC)2->SW1->SW[12345678]->RX delay=2
IO->SW1->SW(CDC)2->SW2->SW[2345678]->RX delay=2
IO->SW1->SW(CDC)2->SW3->SW[2345678]->RX delay=2
IO->SW1->SW(CDC)2->SW4->SW[2345678]->RX delay=2
--
--
-(stage5:89)-
TTSW(stage1)=1
TTSW(stage2)=1
TTSW(stage3)=5
TTSW(stage4)=86
TTRX(stage5)=89
```

Alllinks are checked and latencies are measured in O(1s)

Run mode

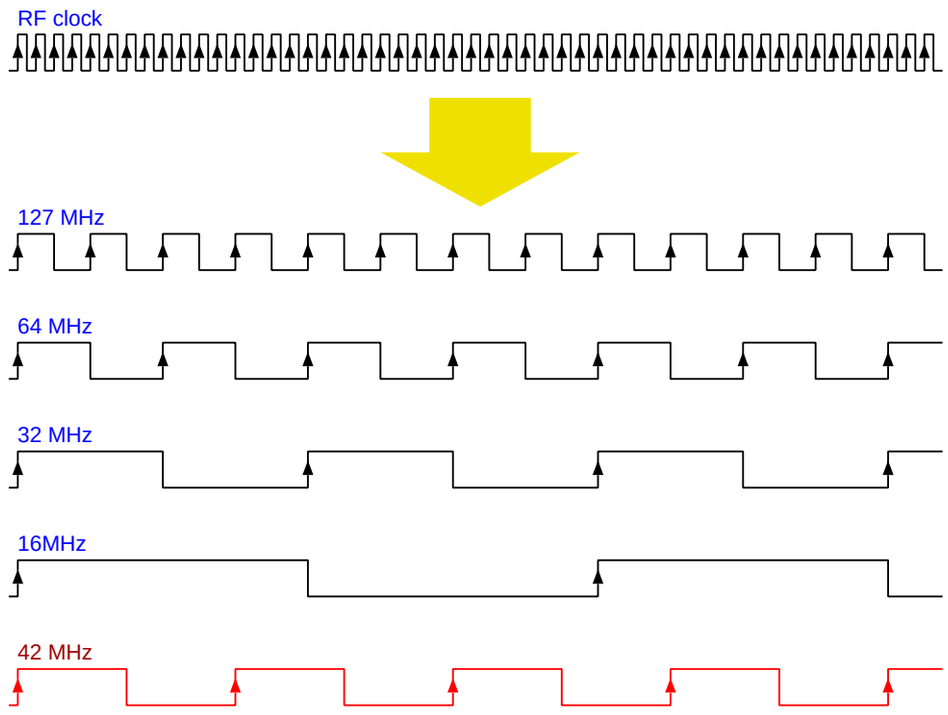
- Replacement of parallel bus during data taking
  - Trigger info is always sent
  - Command sequence is not allowed during the run
  - Latency in delivering trigger info: only useful for the data attached to the event
  - Revolution signal is attached



- Busy signal is always collected
  - Only 1-bit is used in the 10-bit bus
  - More elaborated status signals have been studied, but so far not used in real operation

Clocks at Belle

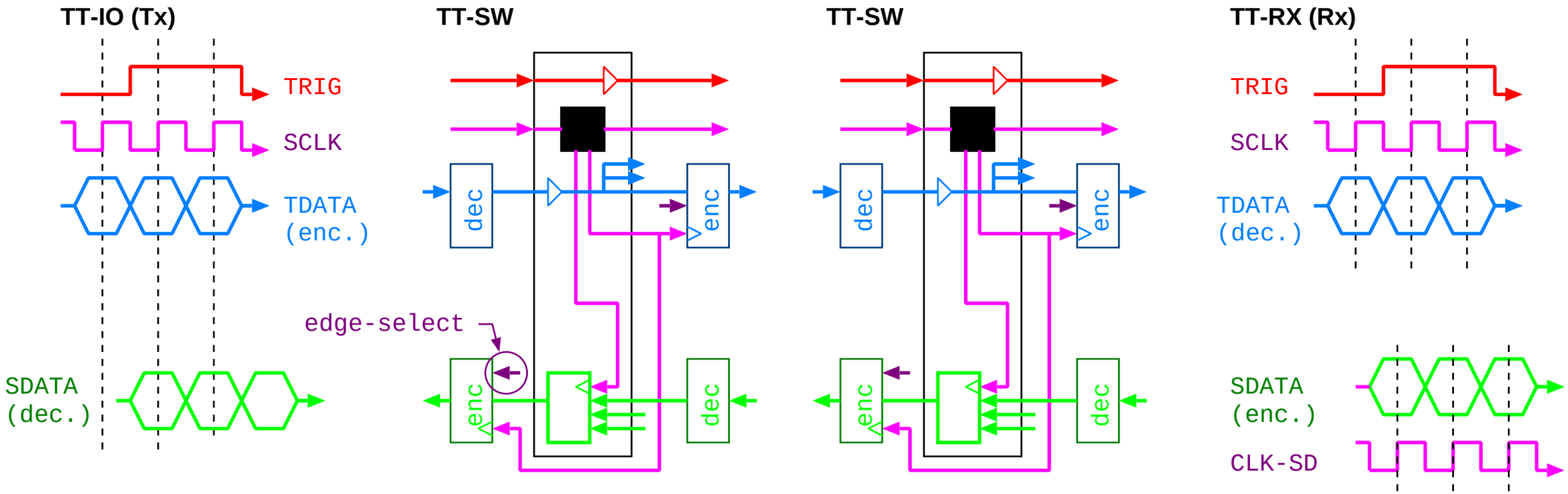
- Many clock frequencies
  - 509 MHz — RF frequency for beam
    (in every 3 or 4 RF bucket spacing)
  - 127 MHz (1/4) — fastest reduced RF
  - 64 MHz (1/8) — for L1 trigger timing
  - 32 MHz (1/16) — for SVD readout
  - 16 MHz (1/32) — for old timing system
  - 42 MHz (1/12) — for COPPER
    (chosen to be compatible with chips for LHC)
    42 MHz is generated in TT-IO using DCM of Xilinx FPGA
  - 100 kHz (1/5120) — Revolution signal, to synchronize different clocks



- Simple minded clock distribution
  - NIM, ECL or LVDS lines, no feedback
  - Possible clock drift (O(1ns)) is monitored and offline-calibrated

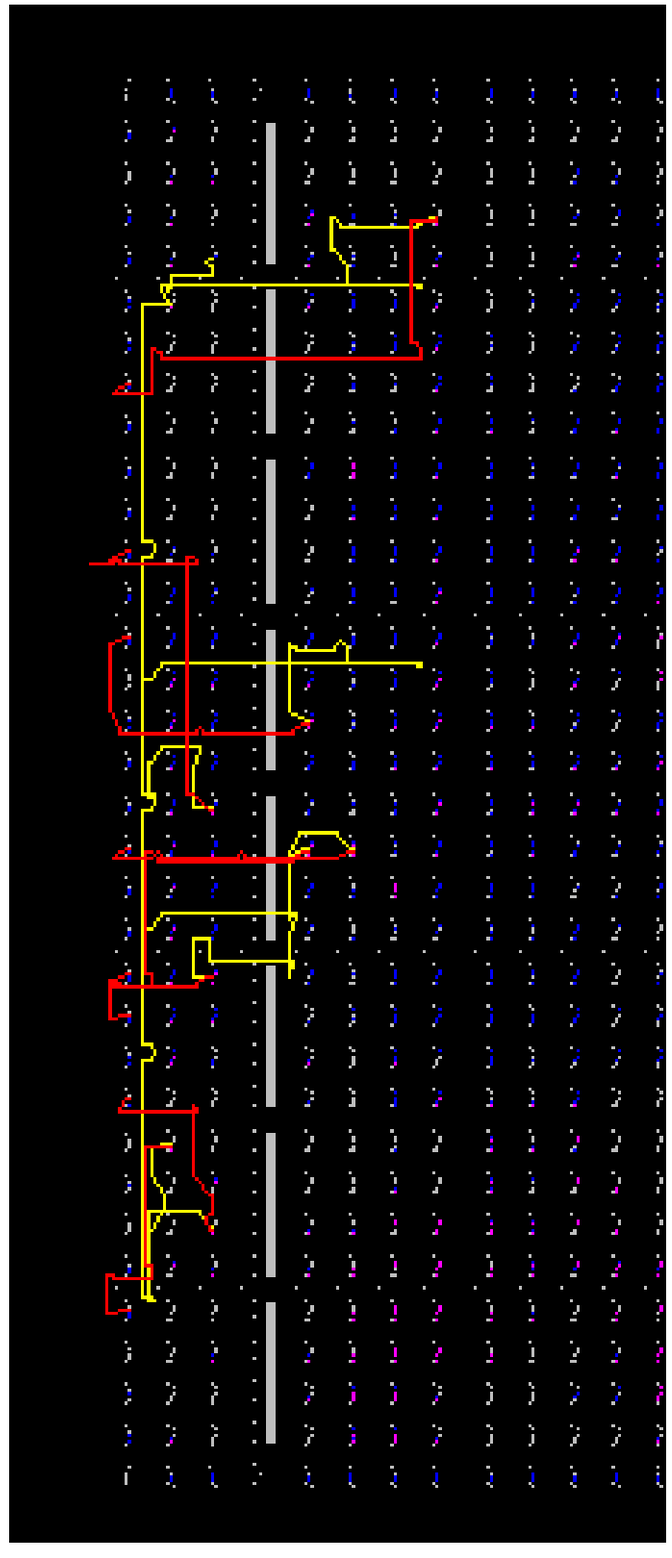
Clock phase of return path

- Down-going: data/clock phase is fixed
- Up-going: data/clock phase has to be adjusted
  - Cable length dependent, need a tuning
  - FPGA is flexible to flip the phase (, or add some delays)
    Clock edge selection (four places per stage)
  - Checked for every inter-stage connection in situ



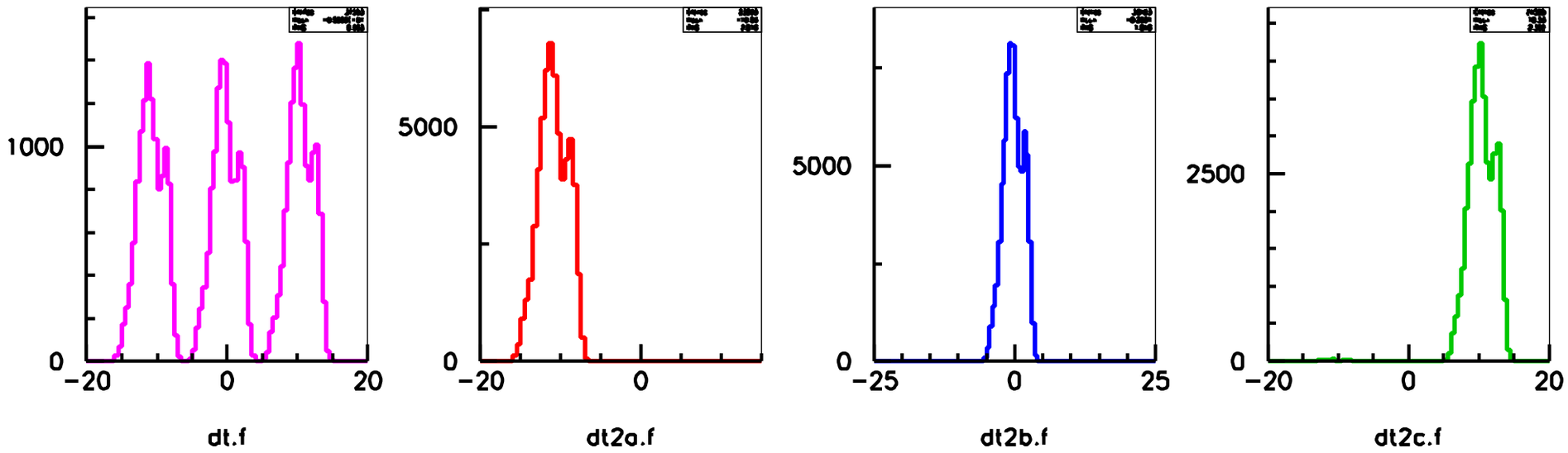
Clock skew and latency

- Non-negligible skew between ports
  - because the clock signal goes through FPGA
  - up to 2 ns skew, still marginally acceptable
- FPGA routing
  - Tuning with Xilinx FPGA_EDITOR and LOC consraints by hand
  - Result: typically 0.2 ns skew between ports (almost impossible to tune further)
  - Directed routing constraint to fix the routing
  - Then other part of the firmware does not affect the skew and latency



Mixed clock configuration

- 42 MHz (COPPER) vs 64 MHz (L1 trigger)
  - Always 3 phase ambiguity between events recorded in FASTBUS (e.g. TOF) and COPPER (e.g. CDC)
  - Phase info is attached to every event and offline corrected



FASTBUS-COPPER time difference

- Controlling FASTBUS readout system
  - TT-IO is reprogrammed for a timing receiver emulation for FASTBUS
  - TT-SW can select which trigger source to be used:
    Trigger for COPPER is synchronized to 42 MHz
    Trigger for FASTBUS is synchronized to 64 MHz

Conclusion

- Serial bus based fast control system at Belle
  - Successful implementation in running Belle DAQ
    everything is synchronized (or otherwise DAQ monitors will complain)
  - Compact implementation for >100 final destinations,
    Fast and flexible control of destination nodes
  - Possible with very limited human resources
- For Belle II upgrade (for SuperKEKB)
  - Serial bus based fast control system will be used
  - Trigger-busy handshake will be given up ➡ pipelined readout
  - Clock distribution scheme will be redesigned