- Click in the field “c’down” above to set a countdown.
- Click on the button below to start the countdown and the local time.

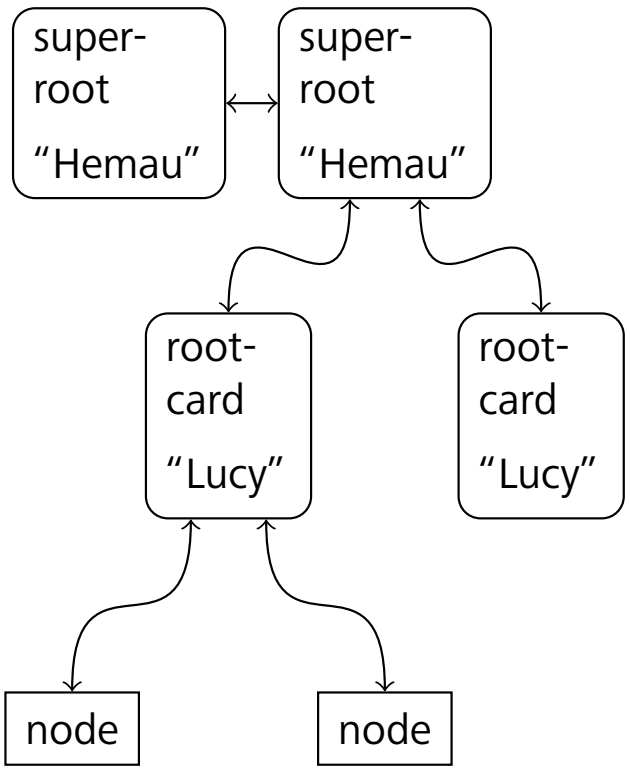
Start/Stop Timer

Synchronization and Error Reporting on *QPACE*

Basic idea: global signals (GS)

- We want to have a low-latency signal tree that can reach any node in a partition
- Independent of the torus network
- Three different signals:
  - *kill* high priority to broadcast of an error condiditon to all nodes. If one node sends *kill*, all nodes receive *kill* immediately.
  - *true* acts as a global barrier. Once all nodes send *true*, all nodes receive *true*
  - *false* is also a global barrier. If all nodes send either *true* or *false*, and at least one node sends *false*, all nodes receive *false*

Implementation sketch



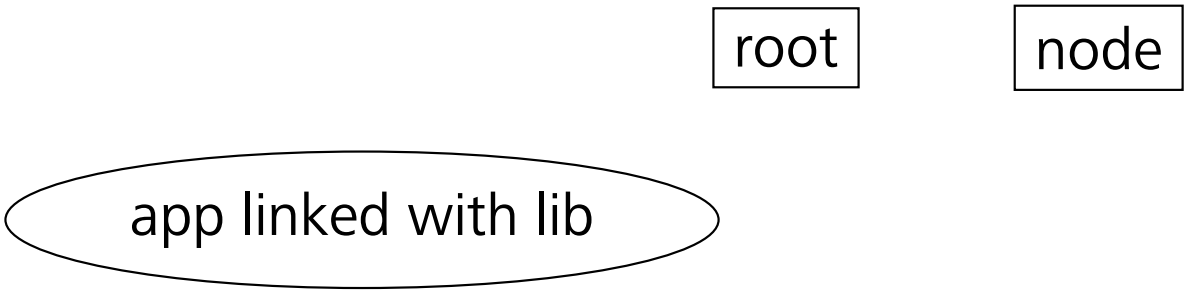
- (super) rootcards do not register GS $\Rightarrow$ low latency
- (super) rootcards contain combinatorical logic to reduce GS.
- *kill* is reduced via a global “or”.
- *true* and *false* are reduced via a global “and”.
- Nodecards register GS, send interrupt to the Cell.
- Linux kernel module handles interrupts.
- Low level access via character device.

Error reporting via kill

app linked with lib

- User program just needs to link with a library.

Error reporting via kill



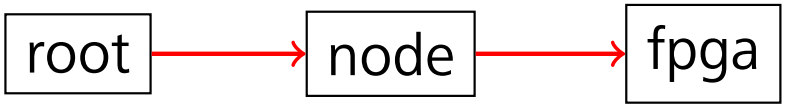
- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.

Error reporting via kill

app linked with lib

- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.

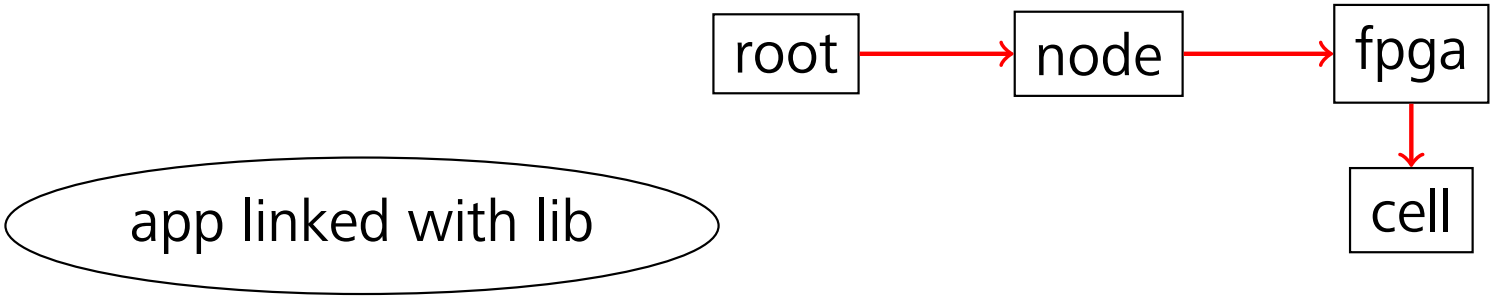
Error reporting via kill



app linked with lib

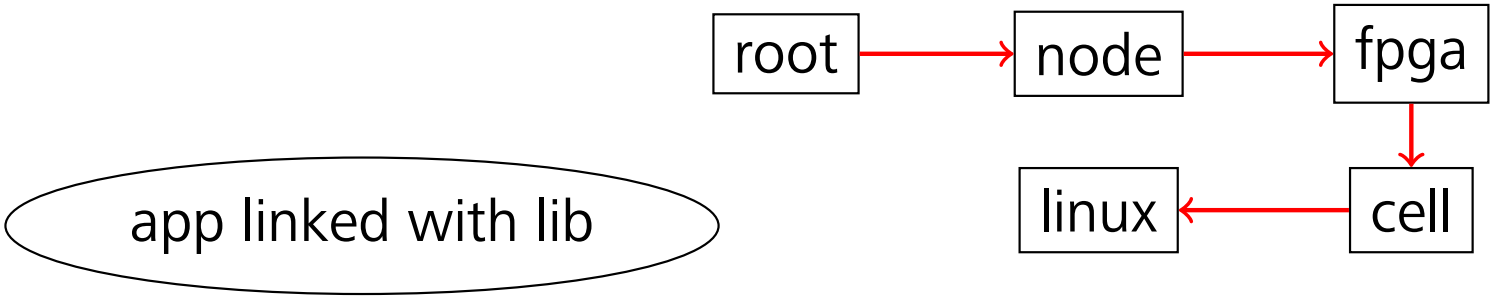
- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.
- FPGA detects signal

Error reporting via kill



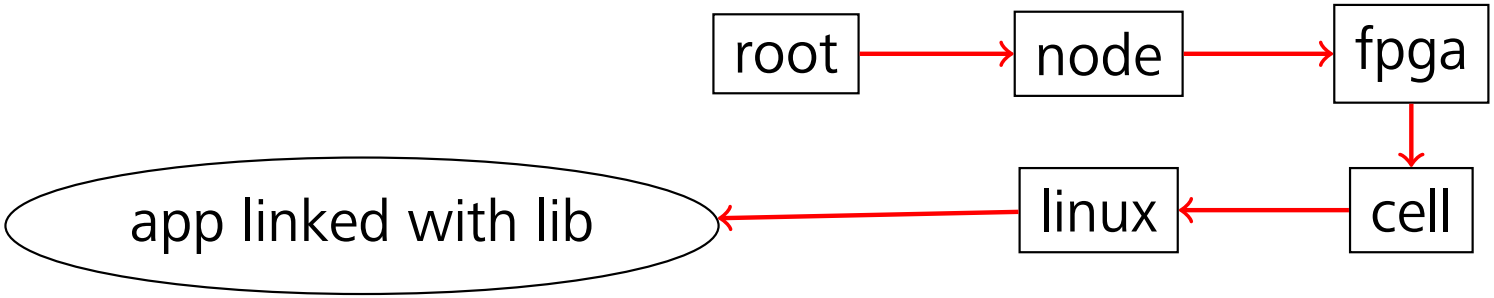
- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.
- FPGA detects signal
- FPGA sends interrupt to the Cell.

Error reporting via kill



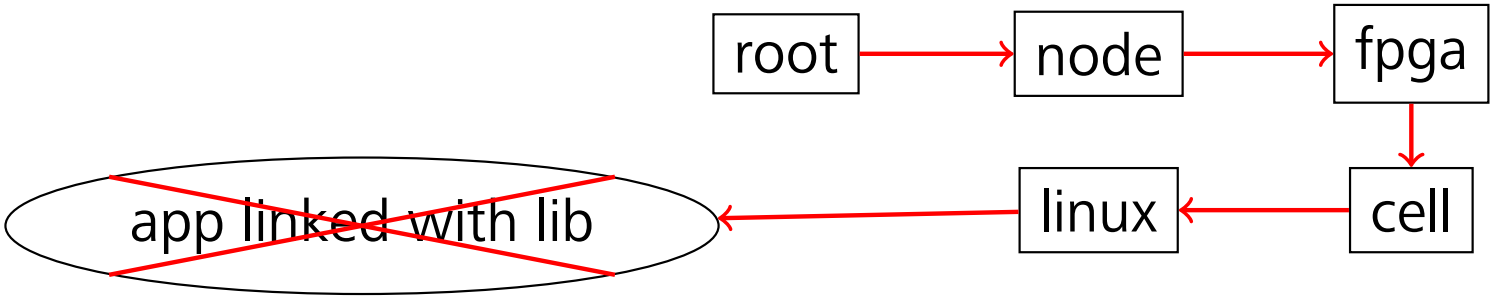
- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.
- FPGA detects signal
- FPGA sends interrupt to the Cell.
- Linux kernel module handles interrupt

Error reporting via kill



- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.
- FPGA detects signal
- FPGA sends interrupt to the Cell.
- Linux kernel module handles interrupt
- Module sends signal SIGIO to userland.

Error reporting via kill



- User program just needs to link with a library.
- Rootcard and nodecard suitably configured.
- Nodecard receives a *kill* from rootcard.
- FPGA detects signal
- FPGA sends interrupt to the Cell.
- Linux kernel module handles interrupt
- Module sends signal SIGIO to userland.
- Interrupt handler installed by library prints error information, terminates program.

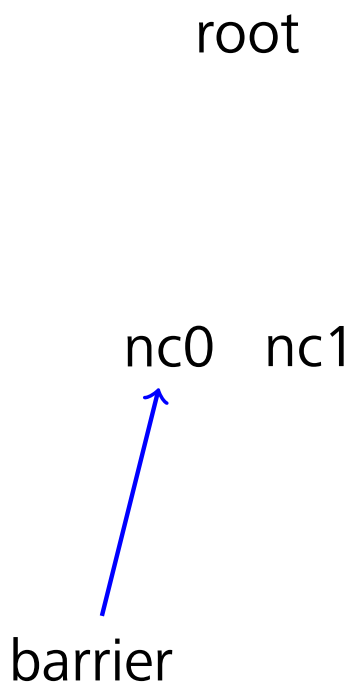
kill causes

Kill can be caused by a number of events:

- Software exception (user triggerd, explicit)
- Hardware exception, e.g., torus
- Kill sent by any other node
- Empty nodecard slot
- Invalid use of GS

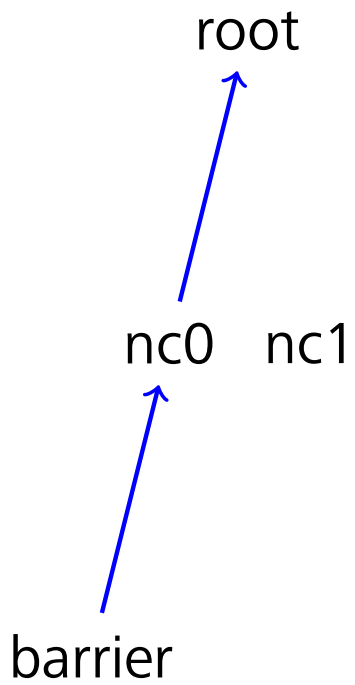
Synchronization via true/false

- User calls barrier, GS sent.



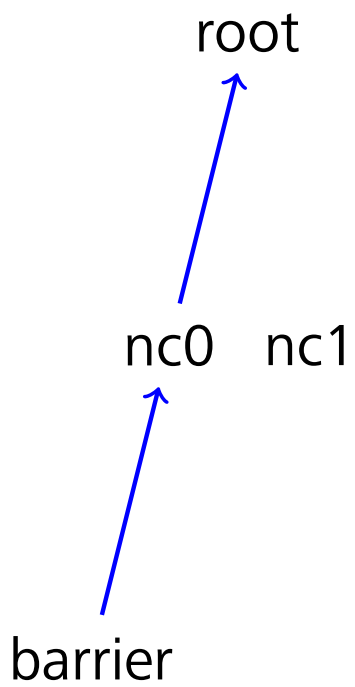
Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.



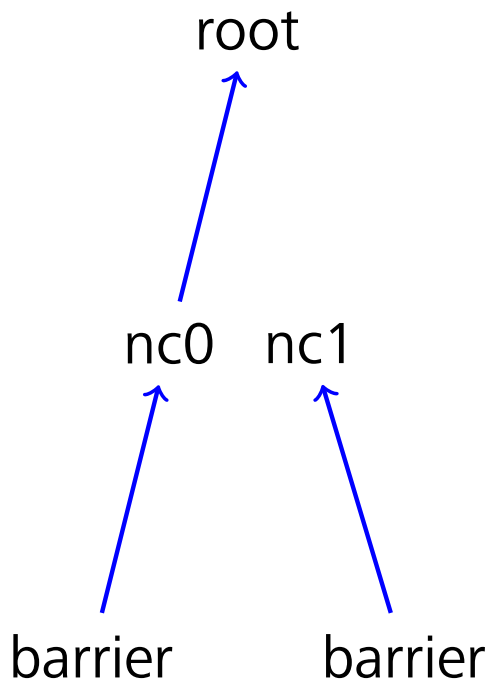
Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.
- Root reduces (global "and"), nothing is sent.



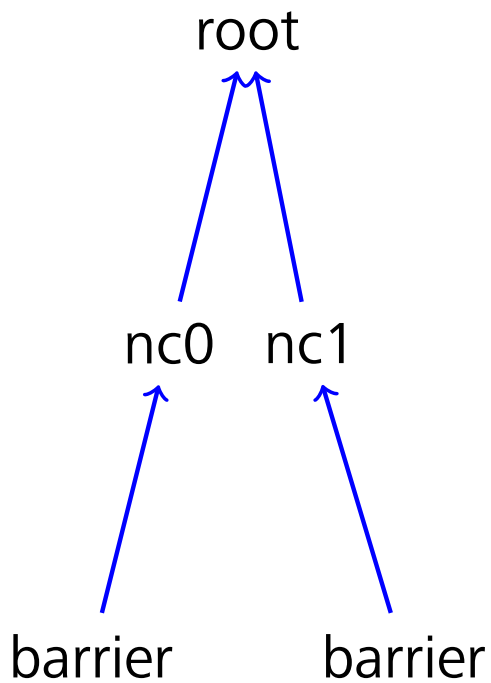
Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.
- Root reduces (global "and"), nothing is sent.
- barrier on other node called.



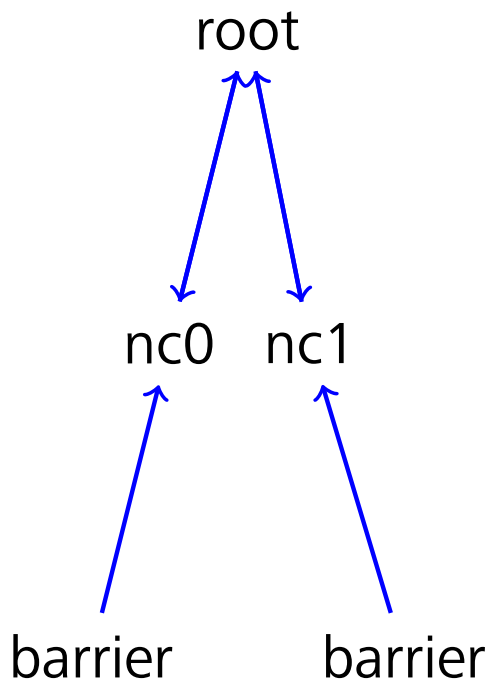
Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.
- Root reduces (global "and"), nothing is sent.
- barrier on other node called.
- Signal propagates.



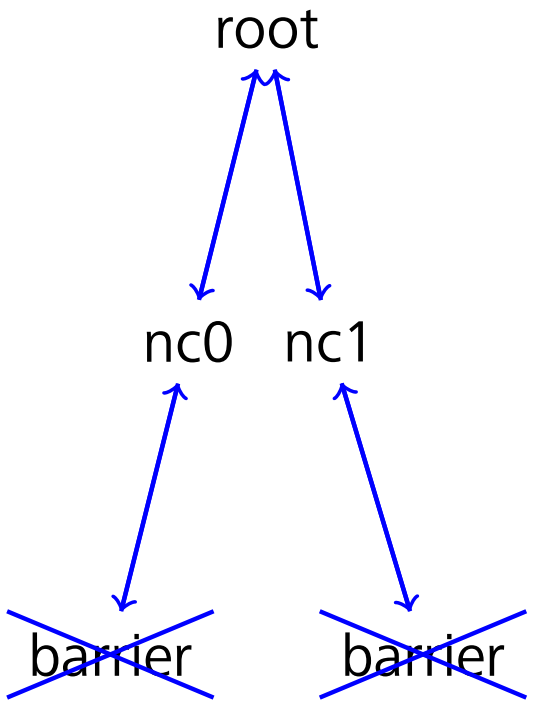
Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.
- Root reduces (global "and"), nothing is sent.
- barrier on other node called.
- Signal propagates.
- Reduced signal propagates back.



Synchronization via true/false

- User calls barrier, GS sent.
- Signal propagates.
- Root reduces (global "and"), nothing is sent.
- barrier on other node called.
- Signal propagates.
- Reduced signal propagates back.
- barrier released.



High level code: timed barrier

```
struct timeval tv = {
    .tv_sec = 60; /* one minute timeout */
    .tv_usec = 0;
};
unsigned char gs = GSTrue;
int err = gsTimedBarrier(gsAutoHandle(), &gs, &tv);
if (err == -2) printf("timeout"); exit(-1);
if (err) printf("other error occured"); exit(-1);
printf("Signal %s received.\n",
    gs ? "true" : "false");
```

Implemented as `write()`/`select()`/`read()` from a character device.

Broadcasts

Barriers can be used to broadcast data: receivers send "true", sender sends "true" or "false". Wrapper functions exist:

Sender:

```
double x = 7.7;
gsTimedBCSend(gsAutoHandle(), (unsigned char*)(&x),
sizeof(x), NULL);
```

Receivers:

```
double x;
gsTimedBCRecv(gsAutoHandle(), (unsigned char*)(&x),
sizeof(x), NULL);
```

Use case questionalbe: torus network is faster.

Performance

Performance of GS measured via broadcasts:

transfer	86 ± 3 kBit/s
latency	11.7 ± 0.4 μs

Measured for large transfers (1 GByte).

Latency mainly (6 to 7 μs) is spent in the FPGA/Cell interface, the rest in Linux system calls.

Low level interface

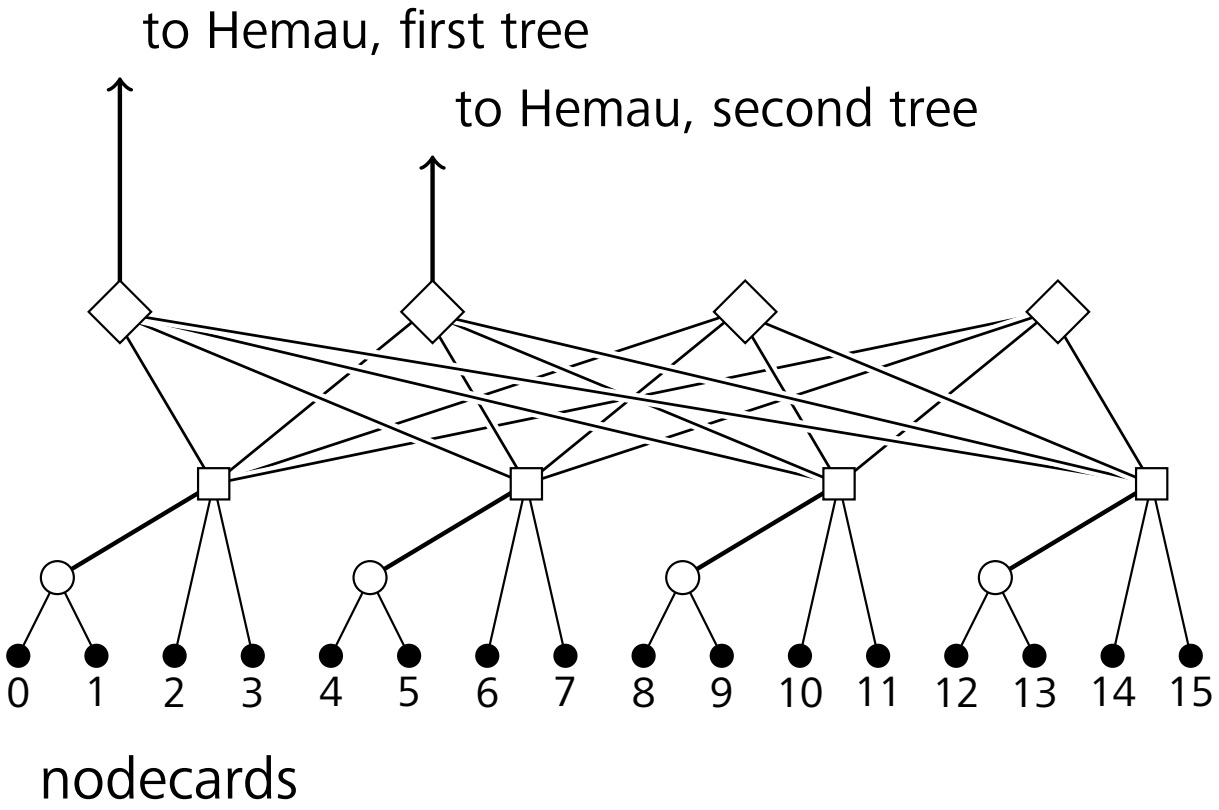
- GS accessible as Linux character device.
- Send/receive implemented as `read()`/`write()`
- low-level functions (statistics, ...) accessible via `ioctl()` (Only for very experienced users)
- Information (error states, statistics) easly accessible via proc file system:
  `cat /proc/gs`

Details and Limitations

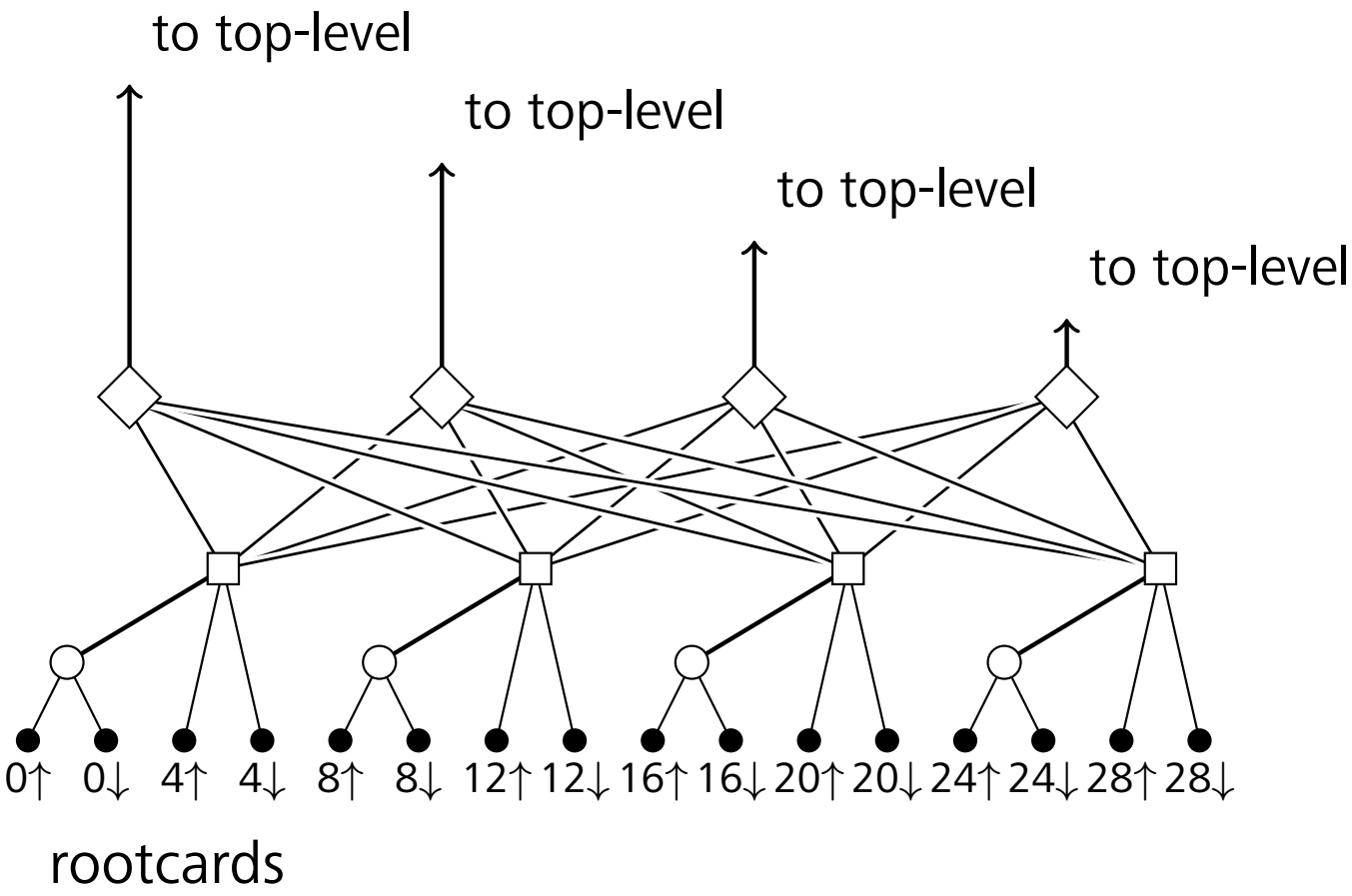
Possible partitions have some limitations:

- Number of rootcards and superroot-cards.
- Number of cables between superroot-cards.
- Available number of logic gates on rootcards and superroot-cards.

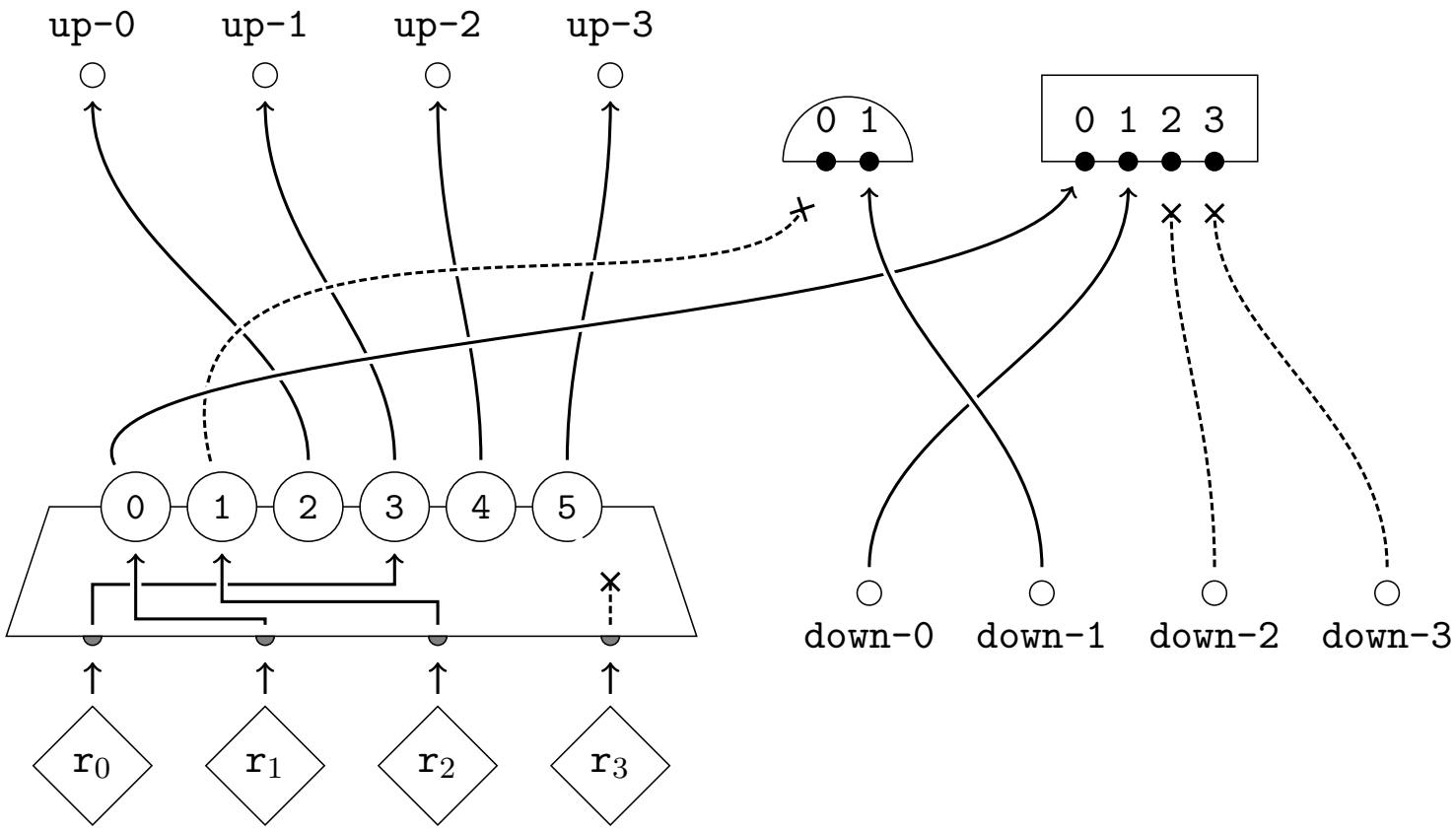
GS rootcard logic



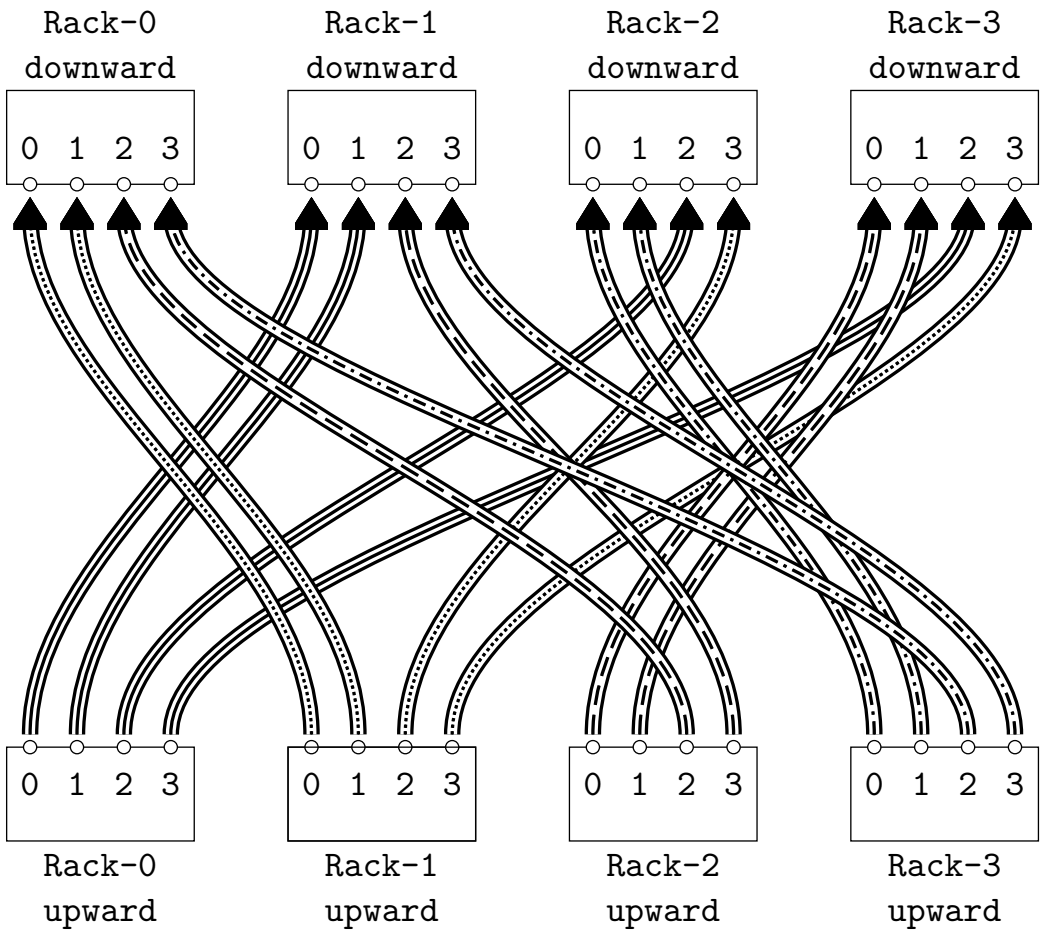
GS superroot logic



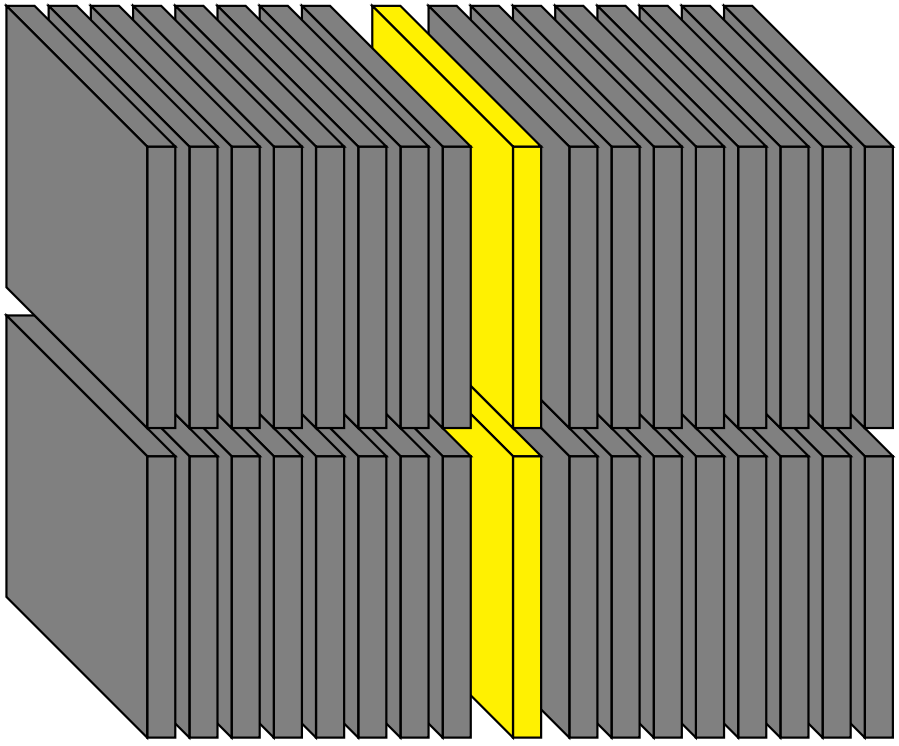
top-level logic



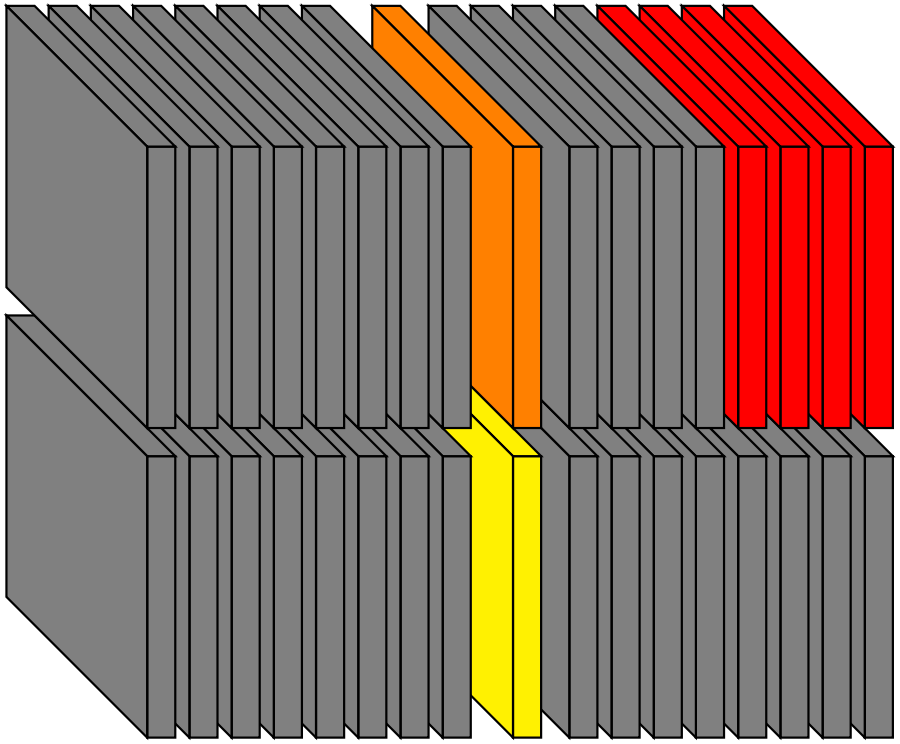
Inter-superroot cabling

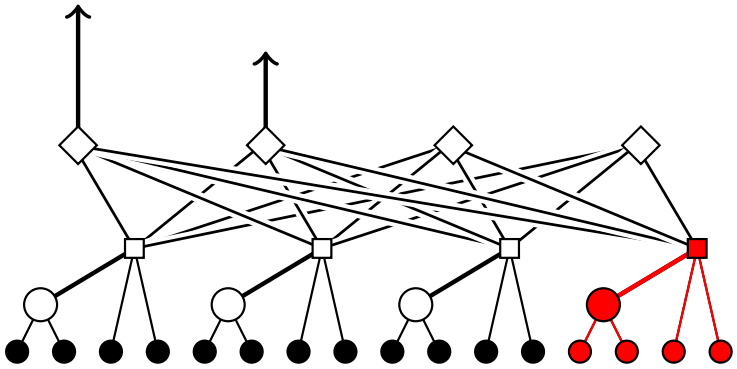


single backplane



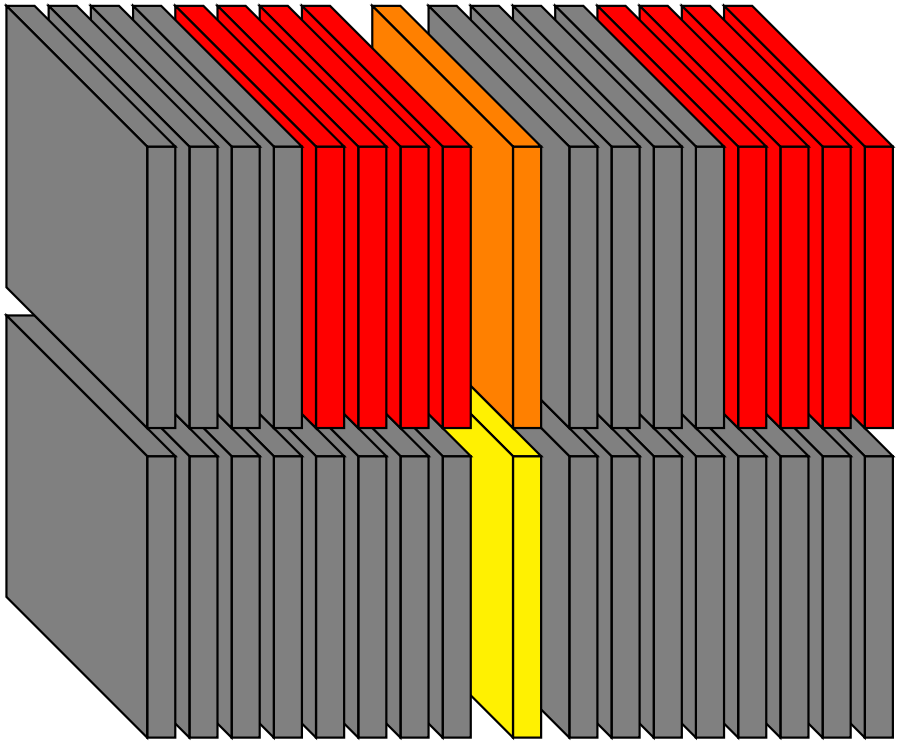
single backplane – $1 \times 1 \times 4$ – quad

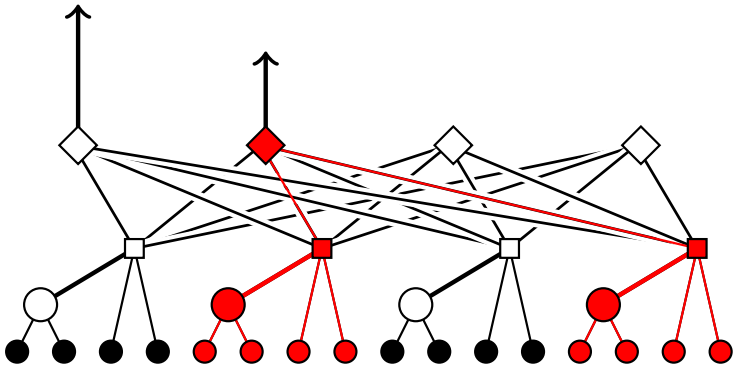




one quad

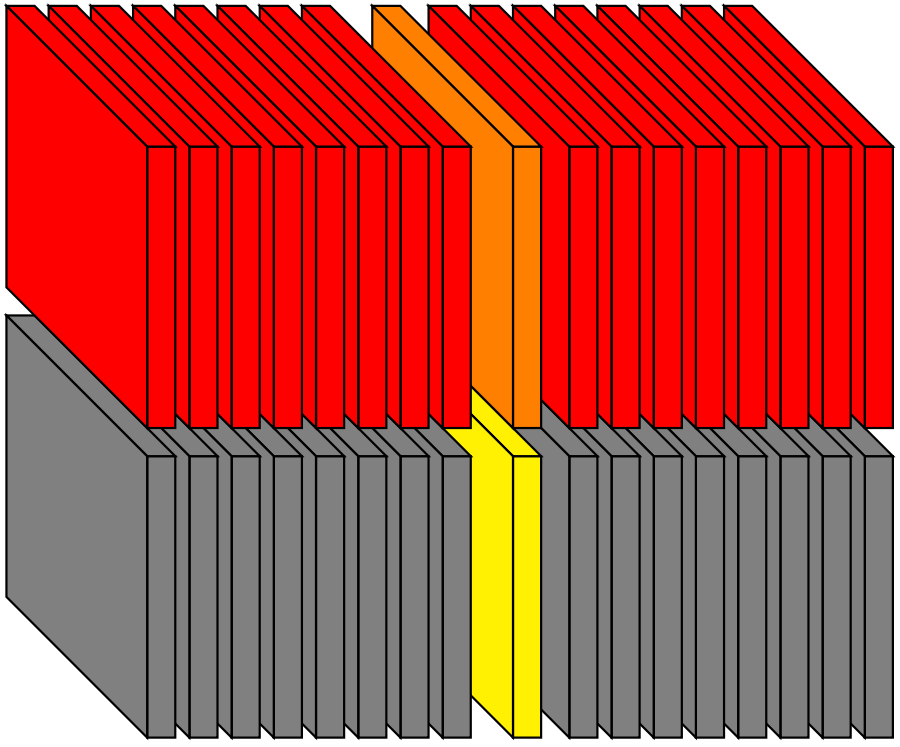
single backplane – $1 \times 2 \times 4$ – row

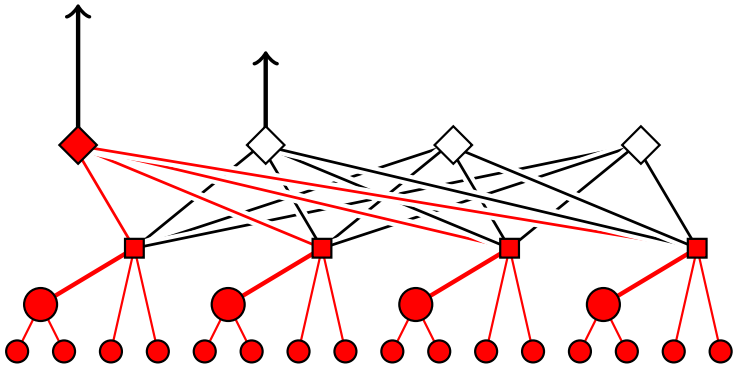




two quads, one row

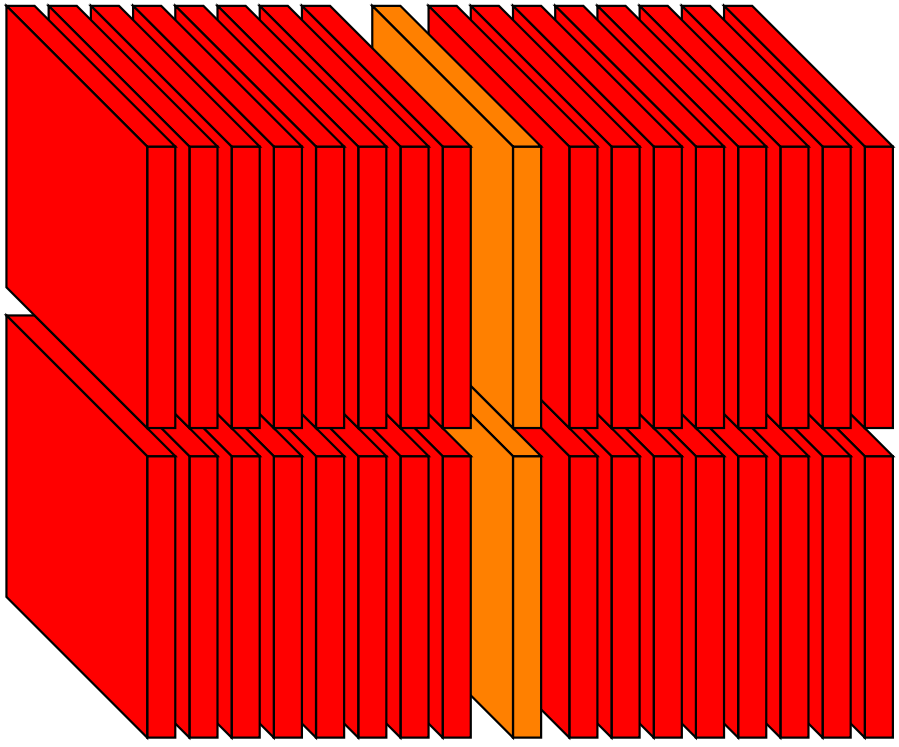
single backplane – $1 \times 2 \times 8$ – row

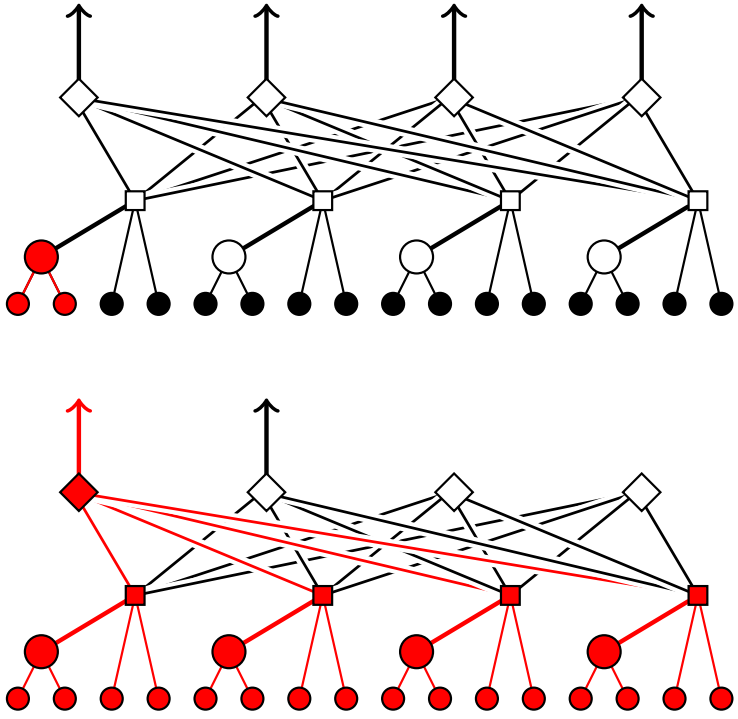




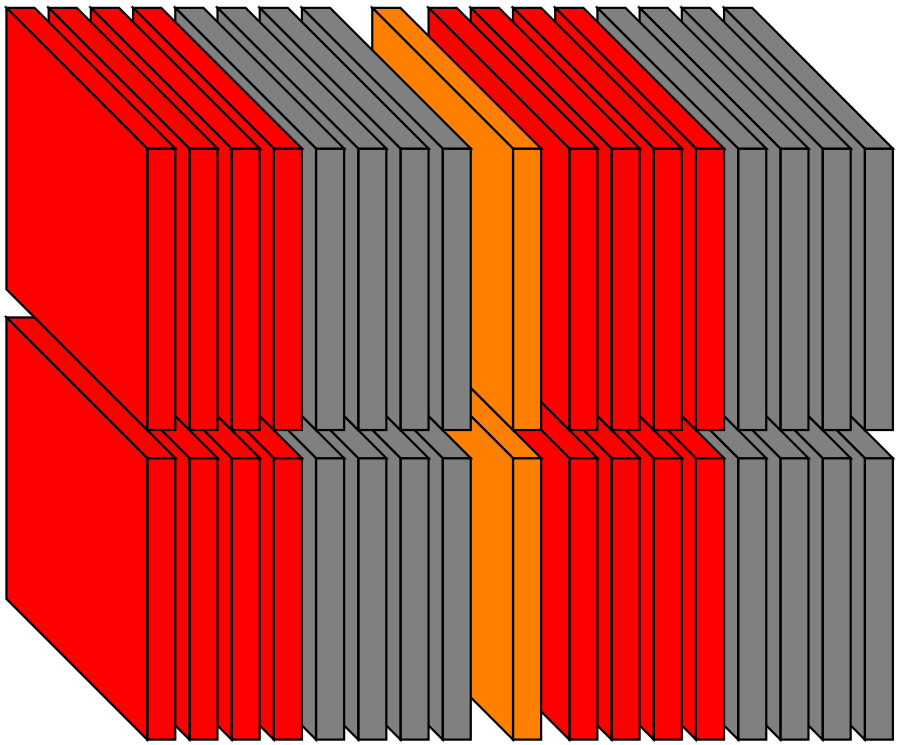
four quads, one row

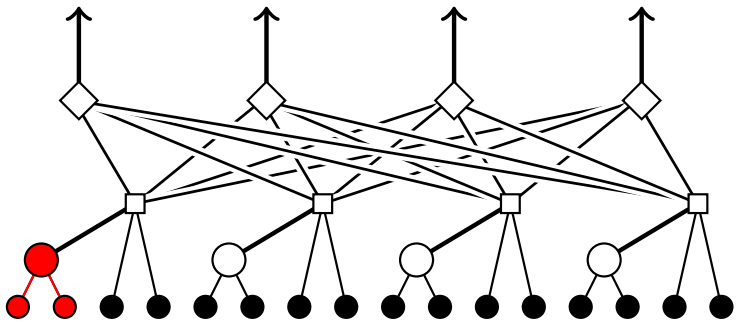
single backplane – $1 \times 4 \times 8$ – super pair/quad

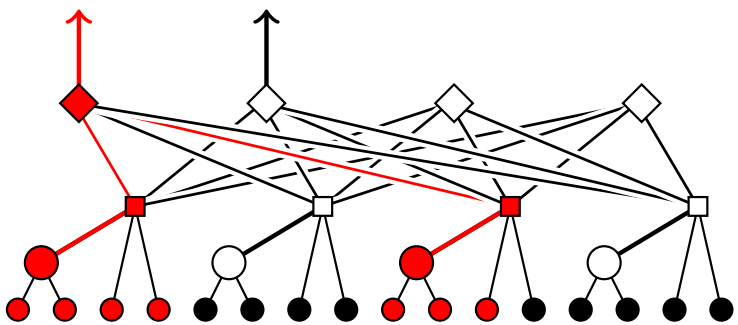




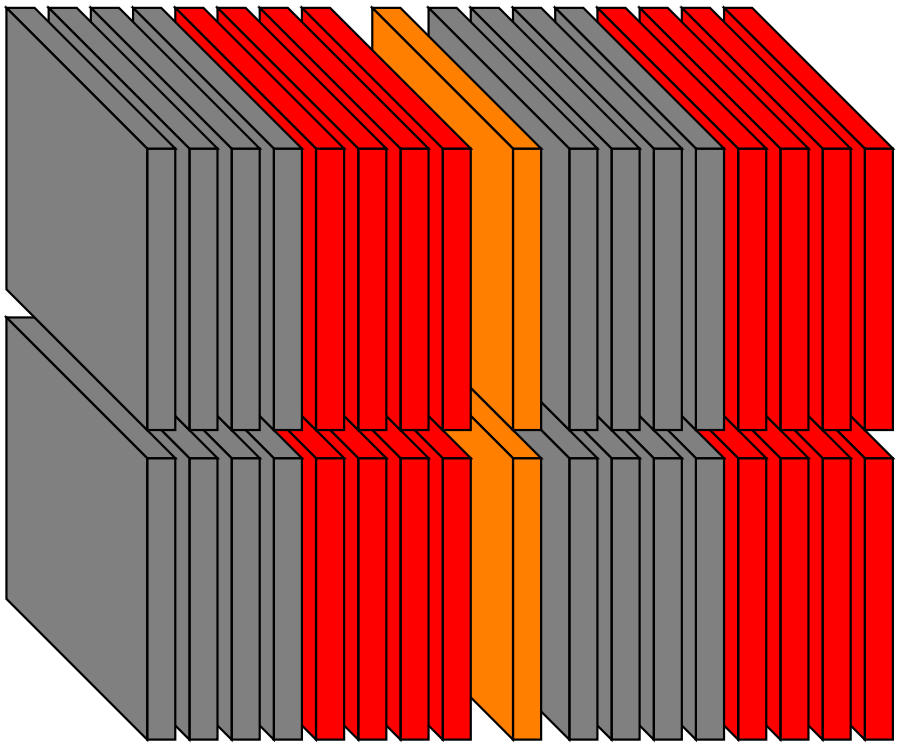
single backplane – $1 \times 4 \times 4$ – super pair/quad, first tree

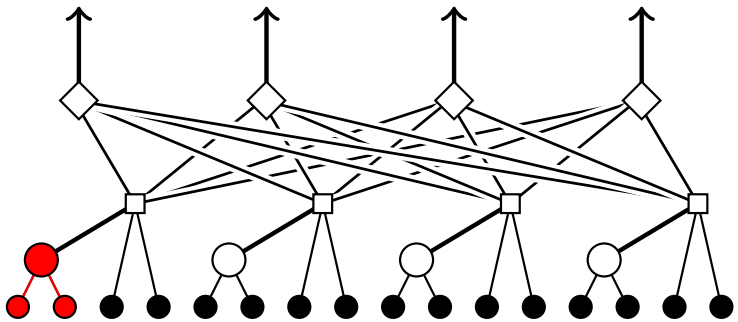


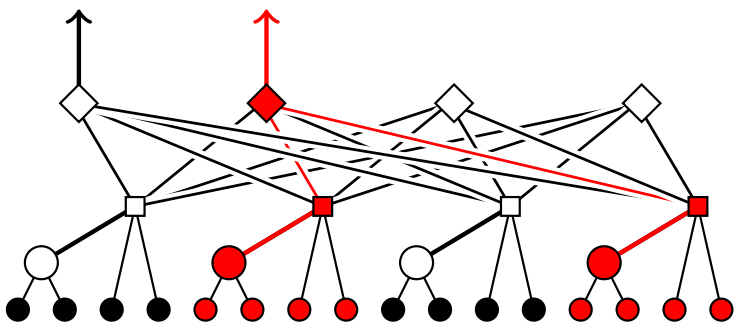




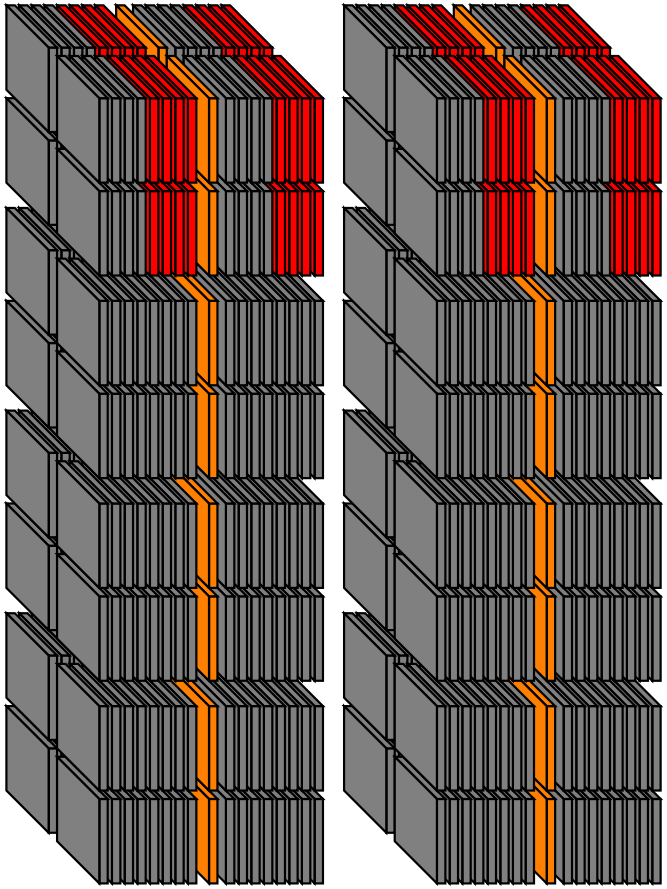
single backplane – $1 \times 4 \times 4$ – super pair/quad, second tree

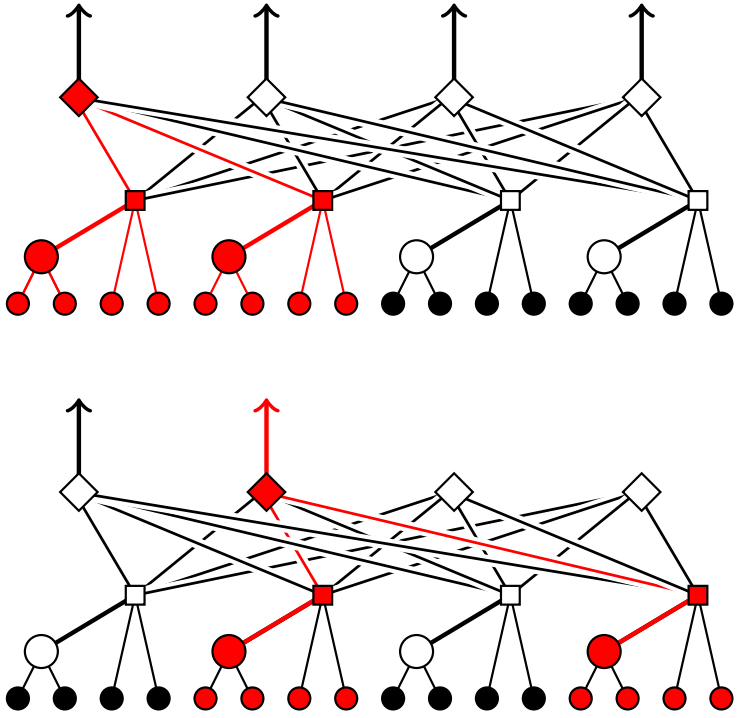




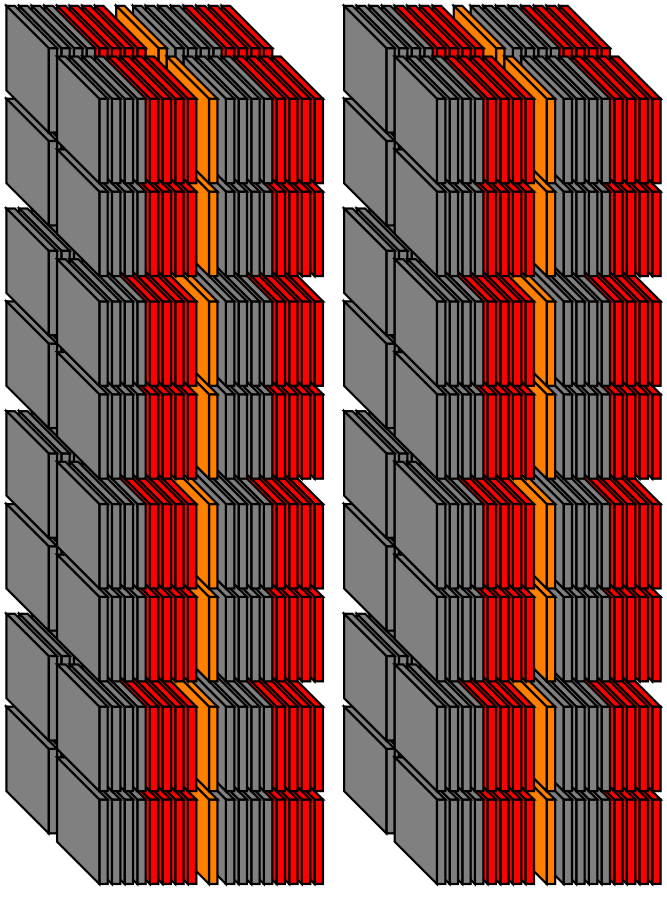


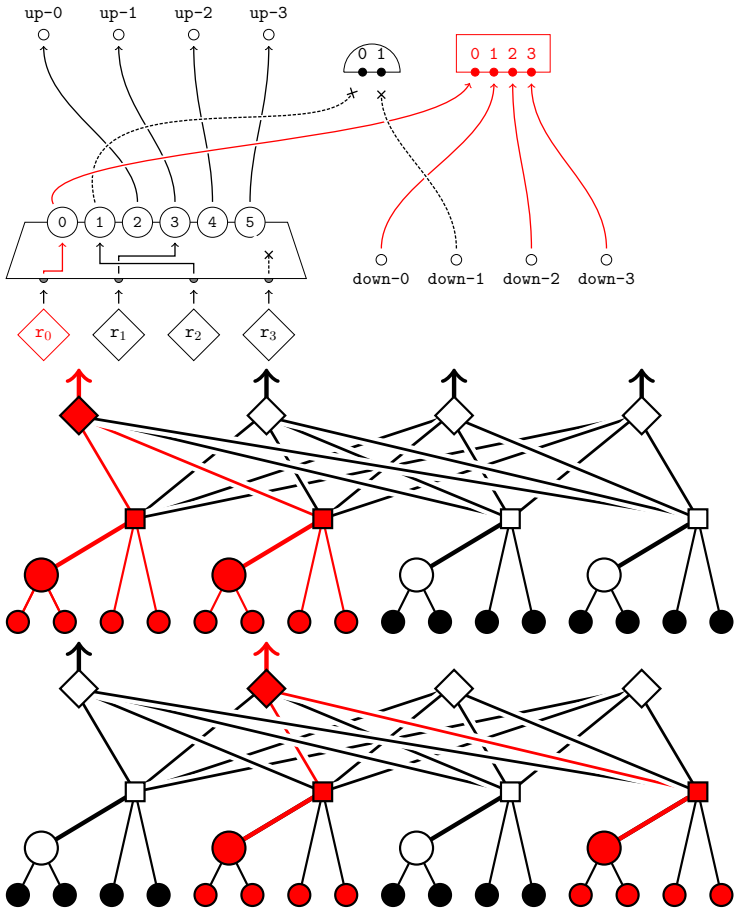
two racks – $4 \times 4 \times 4$ – super row, second tree





two racks – $4 \times 16 \times 4$ – top-level quad, second tree





Thank you for your attention!