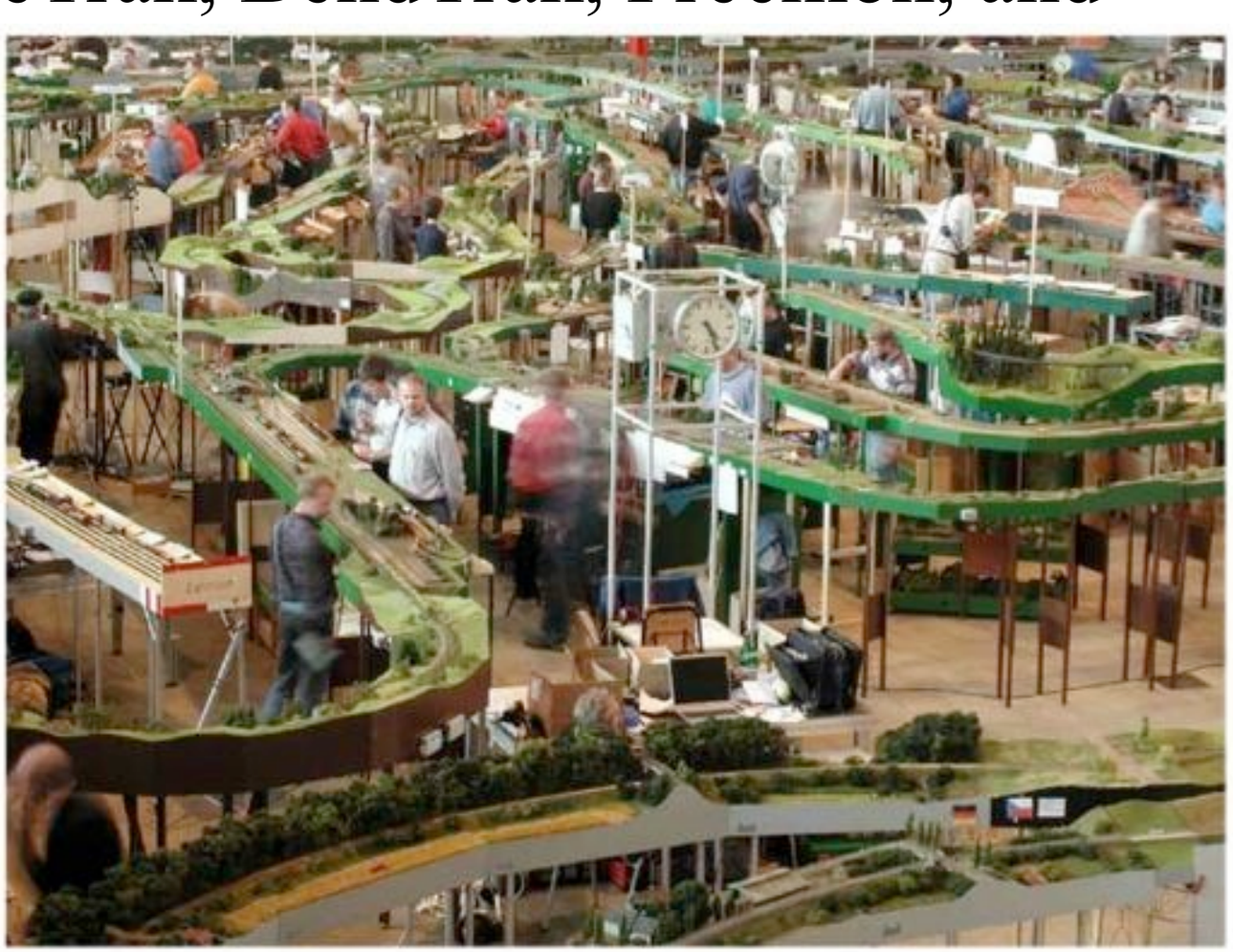
Autoconfiguration in OpenLCB

The Problem

- At the present, there are minimal supports for Modular Layouts.
- Eg: Ntrak, oNeTrak, BendTrak, Freemon, and Freemo ...

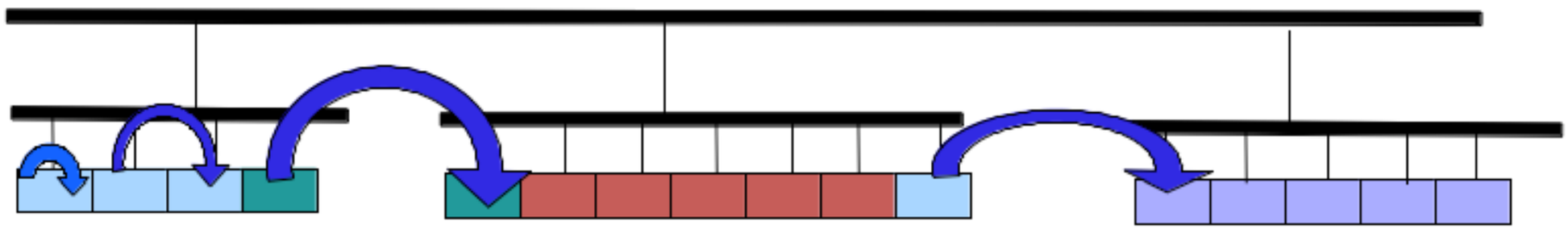


- Usually meets are made up of large and small club layouts, with some individual modules added.
- Setup is a lengthy process!
- ... most groups do not implement signaling
- ... or if they do it is via a specialized bus that forwards aspect indications between modules.
- But if we can help with set-up and signals, then more groups will want to implement them. Chicken and Egg Problem.

- Adding a Local Control Bus can help
- ... but if each module has at least one node
- ... and each node has multiple events
- ... How do we connect them?
  Also, the bus has a limit to how many nodes it can handle
- ... and there will be a lot of traffic.
- What can we do to help?

Fortunately OpenLCB already has some help with these ...

- Very large ID and Event spaces ...
  ... and they are unique ...
- so no collisions!
- Multiple segments
- ... with a high speed backbone bus
- Routable messages ...
- ... messages are only sent to the segments that need them.



Some things help reduce the problem:

Except for:

... global messages (Emergency Stop, Power-on, etc)

... configuration messages

... debugging

(these can be handled by smarter Gateways, I won't cover that here).

Most messages have a limited area of influence

... and its usually limited to

... the module (local turnout control, animation, etc)

The exceptions to the above are called edge events

... ie those that cross the edge of boundary of a module or group of modules.

These events will need to taught to the nodes responsible.

... this can be done in a number of ways:

Manually

... but it is a big job

A group of Club modules can be considered as a large conglomerate-module, and help reduce the problem, because the internal events are be predefined.

Or, automatically, as follows.

Auto-configuration:

There are at least two methods:

Using XML to describe the module or module-groups

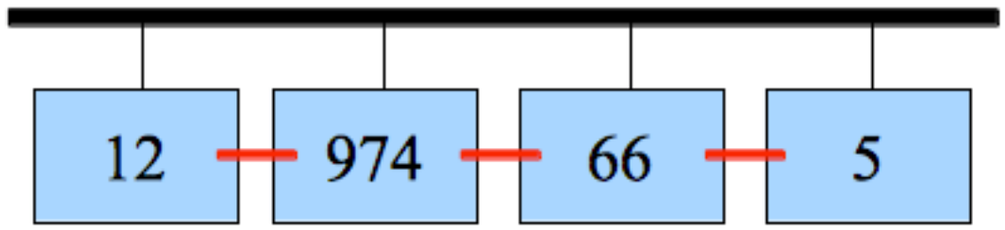
... with software to help the manual configuration.

... likely the XML files would be supplied by each individual and group.

Auto-Configuration using the Bus

OpenLCB would like to enable different solutions, some of dependent and some independent of large central processor.

We need to find a way for modules to discover their neighbours.



Two possible methods are:

1. Add a wire between modules
   ... and use some protocol for teh two modules to talk to each other.
   ... It could be as simple as pulling the link low and transmitting the module ID.
2. Or, let the trains discover the layout topology, which I think is much more fun!

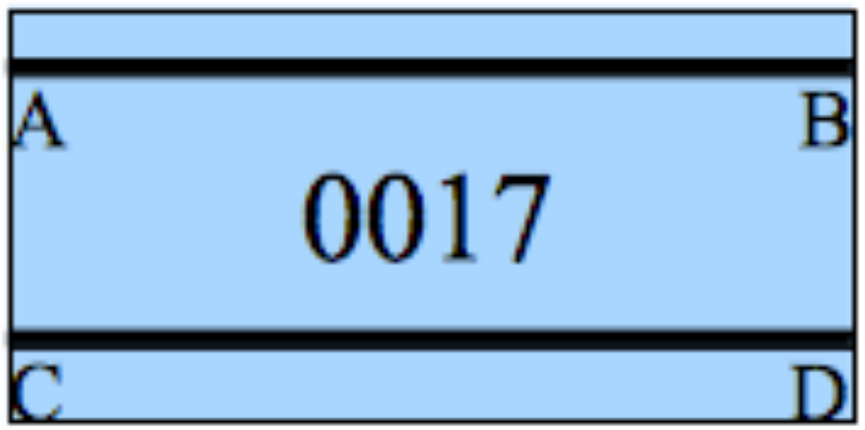
I have been playing with the latter.

How it works:
Each modular group develop its own Protocol Events:
[source node, event#].
By using a preassigned block of Events.
For example: BendTrak might use
ABCDEFABCDEFxxxx.

The organization uses these to develop and **publish** a protocol
... where the bottom two bytes determine specific meaning.

Lets define an example protocol.

For BendTrak, each module has two ends and two tracks ... lets call them connections and label them A, B, C, D



And lets define some messages:

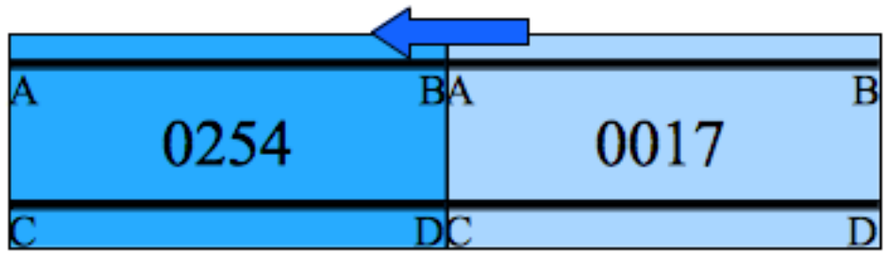
ABCDEFABCDEF0101 – last block at A is unoccupied

ABCDEFABCDEF0201 – last block at B is unoccupied

...

ABCDEFABCDEF0102 – last block at A is occupied

...



Now, when a loco crosses from one module at A to the next at B,

... several messages are sent:
When A on 0017 is occupied [0017, ABCDEFABCDEF0101]
When B on 0254 is occupied [0254, ABCDEFABCDEF0201]
When A on 0017 is unoccupied [0017, ABCDEFABCDEF0102]
When B on 0254 is unoccupied [0254, ABCDEFABCDEF0202]

The nodes on each module can easily recognize these
... by the ABCDEFABCDEF prefix
... and since their connector is/was just occupied or unoccupied
... they can determine the identity of their neighbour from
... its source node id:

So,
node 0017 knows that 0254B is its neighbour at its connection-A
node 0254 knows that 0017A is its neighbour at its connection B

We can then further define some more protocol, by having a signal associated

with each connector, and defining more events for these say:

ABCDEFABCDEFcc10 – signal ss is DARK

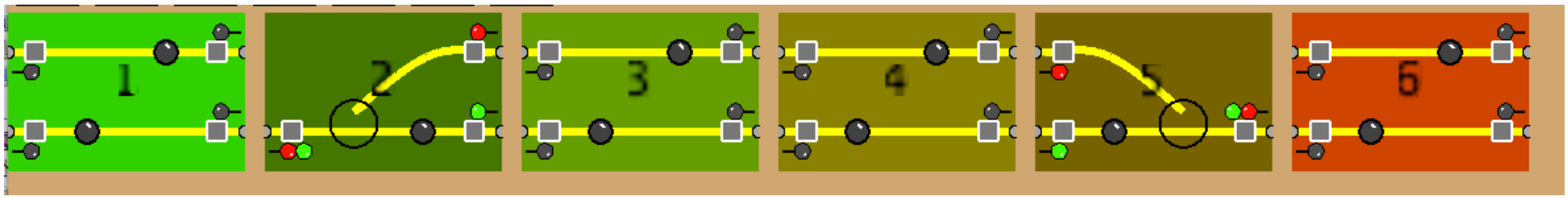
ABCDEFABCDEFcc11 – signal ss is STOP

ABCDEFABCDEFcc12 – signal ss is CAUTION

ABCDEFABCDEFcc13 – signal ss is PROCEED

This allows each connector to automatically respond to it distant signal.

I had originally hoped to show you these modules auto-configuring.



... and then run a train across them to auto-configure their signals.

Unfortunately, the modules are only half finished :-(

However, the good news is that I had made a mock-up / simulation

with a language called Processing
... and I can demo that for you.