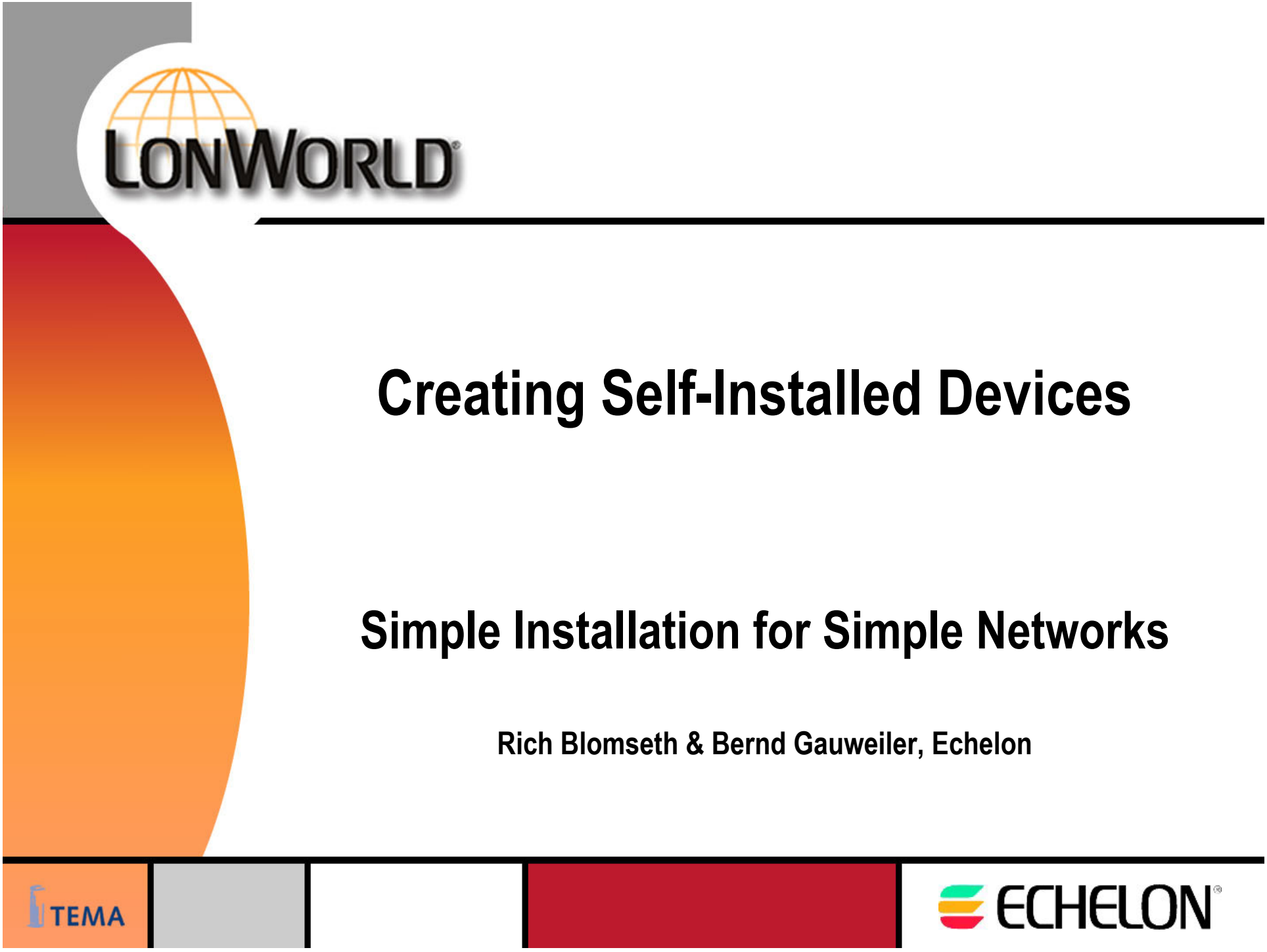

LONWORLD
Creating Self-Installed Devices
Simple Installation for Simple Networks
Rich Blomseth & Bernd Gauweiler, Echelon
TEMA
ECHELON

Agenda

- **Part 1**
  - Self-installation definition
  - Applications for self-installation
  - NodeBuilder® 3.1 and Echelon support for self-installation
- **Part 2**
  - A practical example

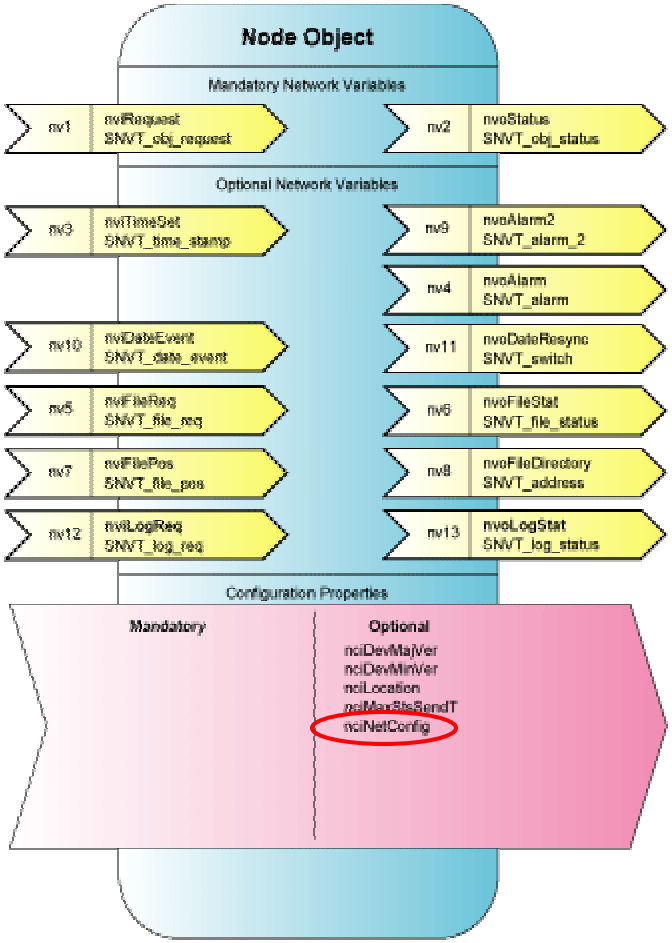
Self-installation Definition

- **Network configuration data**
  - The following network address components contained within a device: device domain IDs, device subnet IDs, device node IDs, device group IDs, network variable selectors, aliases, and NV/message destination addresses
- **Self-installed device**
  - A device that modifies its network configuration data, but does not modify the network configuration data of other devices

Applications for Self-installation

- **Networks that do not require complex interactions between devices**
  - A network view is not required to create connections
  - Devices do not require knowledge of other devices
  - Typically single vendor
- **Example: a lighting system with lamp and switch modules**
- **With proper design, a self-installed device can also be used in a network tool-installed network**
  - Increases the market for the self-installed device

NodeBuilder 3.1 and Echelon Support



- **NodeBuilder 3.1 support**
  - update_clone_domain(), update_address(), update_nv(), and update_alias() functions
  - SCPTnwrkCnfg configuration property type
  - nciNetConfig member of SFPTnodeObject functional profile
- **New Echelon support policy for self-installation**
  - www.echelon.com/support/selfinstall.htm

LONWORLD

Creating Self-Installed Devices Part 2

A Practical Solution

Bernd Gauweiler, Echelon

TEMA

ECHELON

Three Commandments

You shall communicate using...

I. **... a configured clone domain configuration**

(avoids requirement for unique addresses)

II. **... group or broadcast addressing**

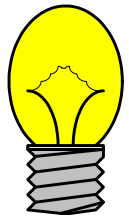
(works on local knowledge alone)

III. **... network variables for data exchange**

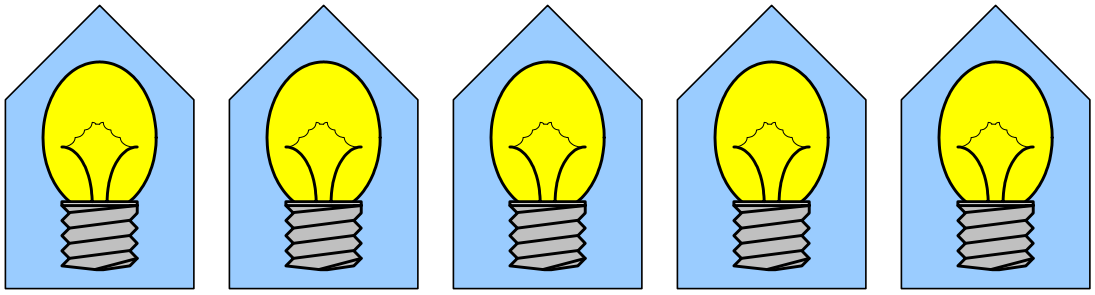
(allows use of self-installed devices in managed network.
Implement SCPTnwrkCnfg!)

Know Thy Self!

- Local knowledge is required



- Requiring local knowledge alone is essential



Know Thy User!

Self-Installation aims at technology-unaware user:

- **Works out-of-the box**
- **No additional hardware or software required**
- **Simple and familiar programming model**

Programming Model

- **Devices fall into *categories***

  lighting devices, heating devices, etc.

- **Within each category, devices form *parties***

  stairwell lighting party, lounge lighting party, etc.

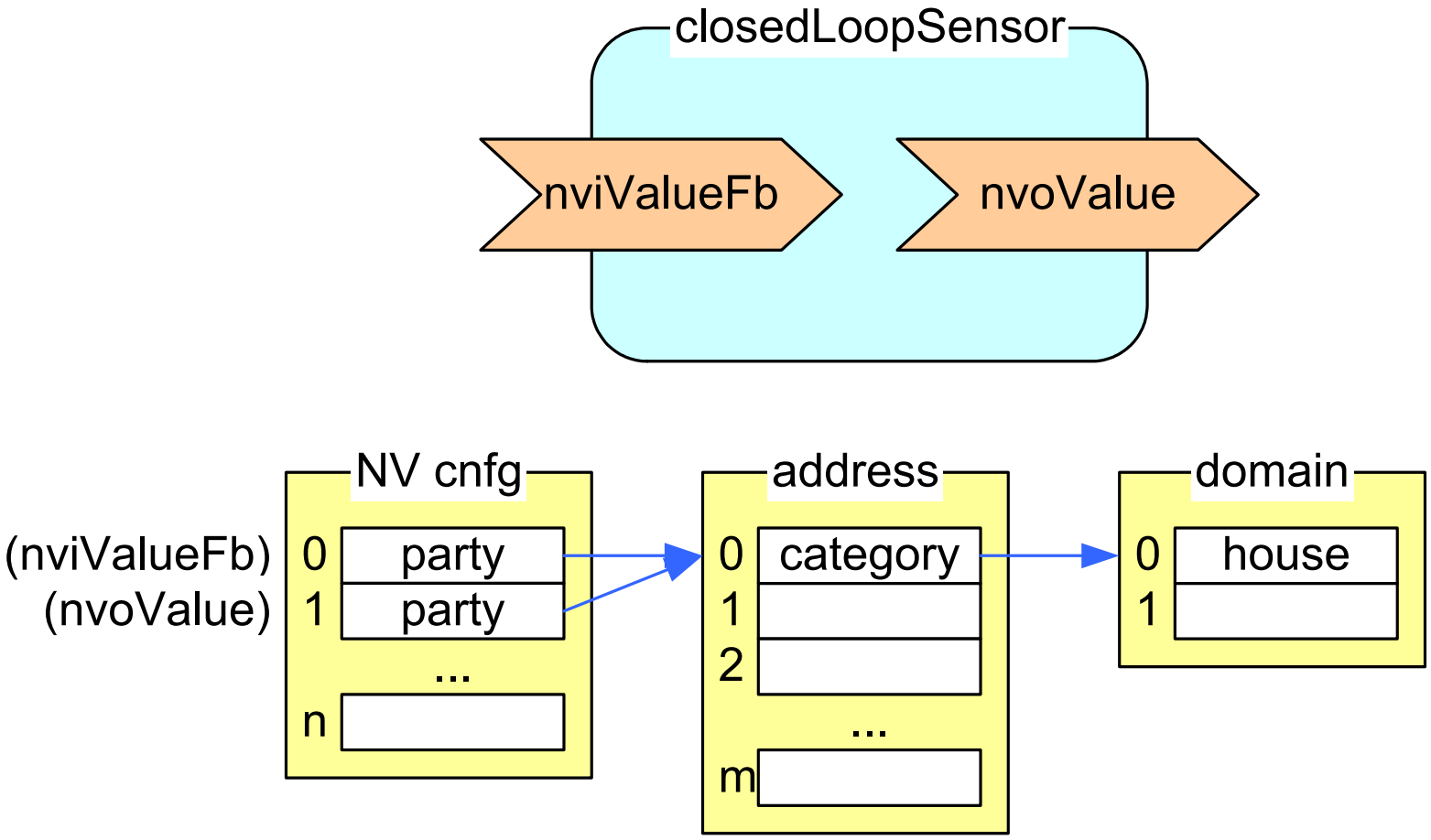
- **Devices belong to a *house***

  Devices shall not interfere with each other across property boundaries without prior consent

Let's Talk LonTalk®

- Each house uses its own *domain*
- Each category of devices uses its own *group*
- Each party uses its own *selector*
- *Alternative mapping possible*

Neuron Firmware Tables



Trade-Offs and Limits

- Accept limitation
- Design for worst-case
- Keep it simple

```
void UpdateCategoryNumber(unsigned uNumber) {
        address_struct aAddr;

         aAddr = *access_address(ADDRESS_INDEX);

         aAddr.gp.type = 1;
         aAddr.gp.size = 0; // open group
         aAddr.gp.domain = DOMAIN_INDEX;
         aAddr.gp.member = 0;
         aAddr.gp.rpt_timer = REPEAT_TIMER;
         aAddr.gp.retry = REPEAT_COUNT;
         aAddr.gp.rcv_timer = RECEIVE_TIMER;
         aAddr.gp.tx_timer = TRANSMIT_TIMER;
         aAddr.gp.group = uNumber;

        update_address(& aAddr, ADDRESS_INDEX);
}
```

Basic Scenario

Use dials to determine party and house numbers

```
when (io_changes(ioDial1)) {
        UpdateHouseNumber(input_value);
}

when (io_changes(ioDial2)) {
        UpdatePartyNumber(myFb::nvoValue::global_index,
        input_value);
}
```

Local Self-Installation

Standard Neuron C Routines for local installation:

```
void UpdateHouseNumber(unsigned uNumber) {
        domain_struct aDomain;
        aDomain = *access_domain(DOMAIN_INDEX);
        aDomain.id[0] = uNumber;
        ....
        update_clone_domain(&aDomain, DOMAIN_INDEX);
}

void UpdatePartyNumber(unsigned uNvIdx, unsigned uParty) {
        nv_struct aNvCnfg;

        aNvCnfg = *access_nv(uNvIdx);
        aNvCnfg.nv_selector_lo = uParty;
        ....
        update_nv(&aNvCnfg, uNvIdx);
}
```

A Useful Hint for NodeBuilder Developers

Conditional definition of DOMAIN_INDEX allows for development and debugging within the NodeBuilder and LonMaker tools:

```
#ifdef _DEBUG
#        define  DOMAIN_INDEX    1
#else
#        define  DOMAIN_INDEX    0
#endif   // _DEBUG
```

Smart Scenario

Basic Scenario requires user to manage dials

Smart scenario for simple pushbutton programming:

1. Press button on device to start programming (host)
2. Host will automatically find unused party number (=selector)
3. Host invites other devices to party
4. Press button on desired invited devices (=guests)
5. Press button on host to confirm party invitation

Party Management

Devices use simple application message protocol to find unused party number, to exchange invitations, etc.

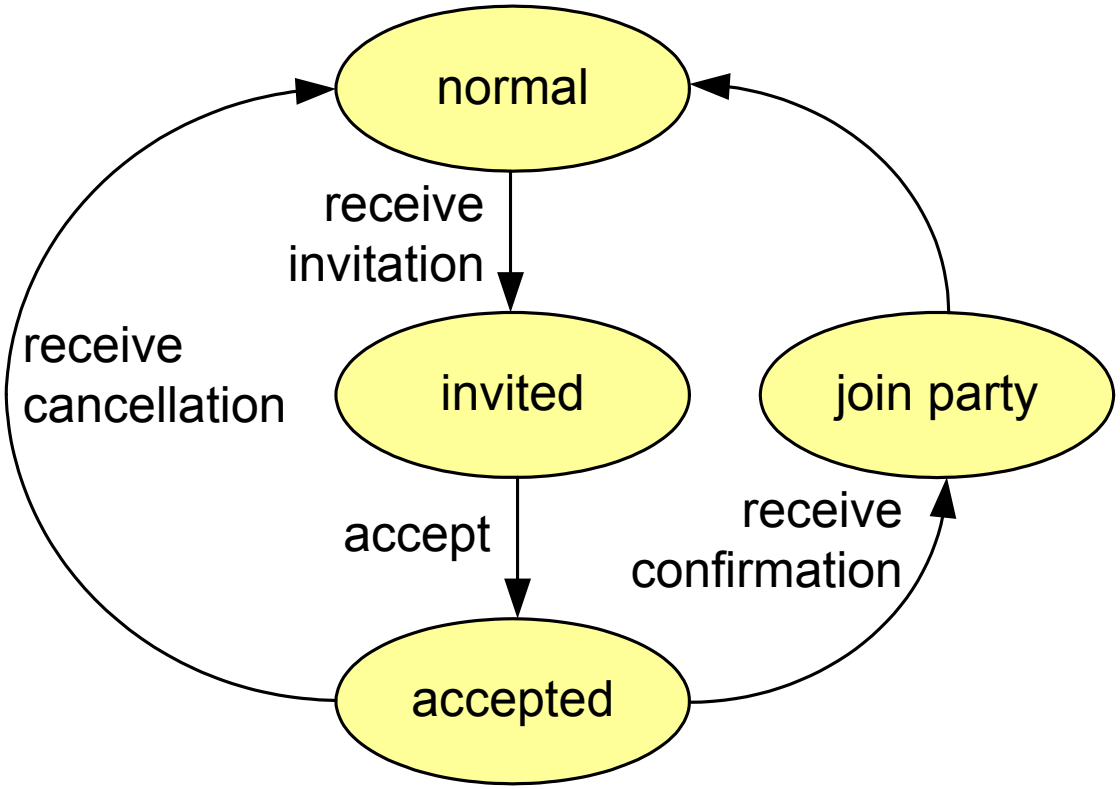
Application message hides proprietary management protocol from integrator

Use signed application messages

Example implementation uses UFPTnodeManager

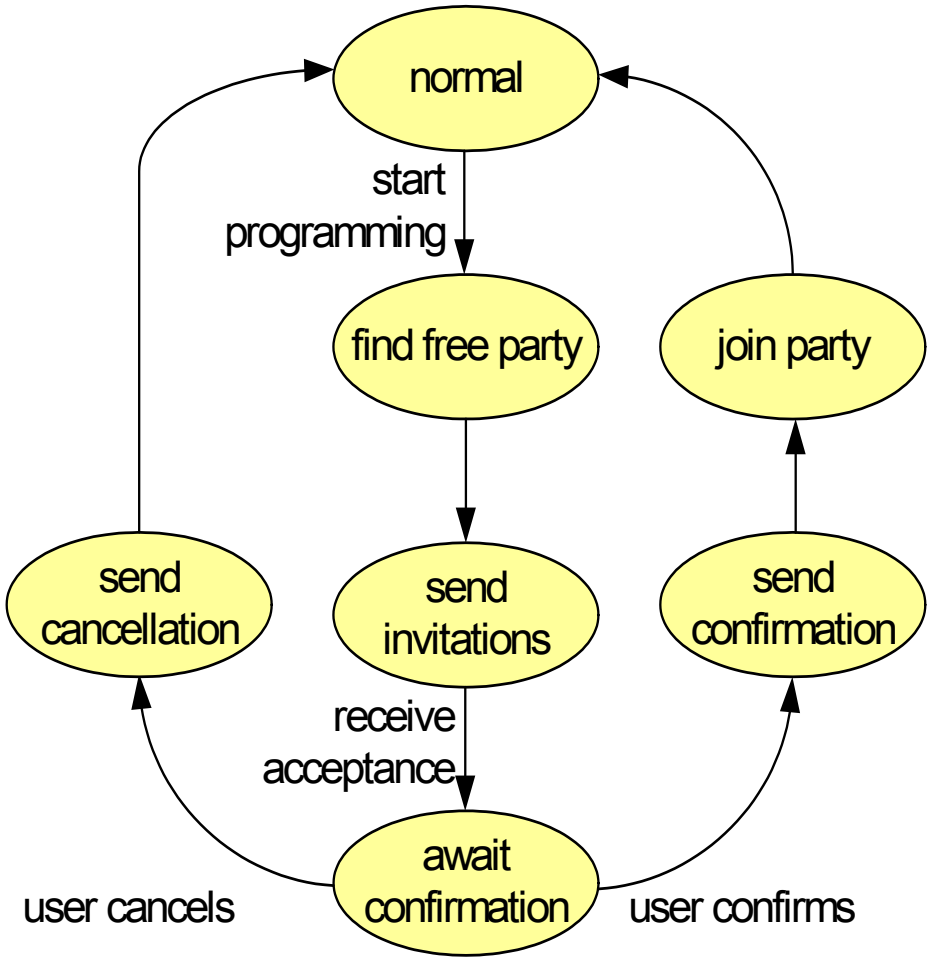
Devices May Become Guests

Devices implement simple state machine to join parties:



Devices May Become Hosts

Devices implement similar state machine to become hosts



All devices implement both host and guest machines

Domain Identifiers

- Use 3-byte domain identifiers
- Use domain[0] for production releases
- Use domain[1] for debugging versions

- Automatic domain look-up possible:
  - \+ Device could query and join an existing domain, move between domains, create new domains
  - \- Development effort with unknown benefit

What's Next?

Visit the Echelon booth to see working smart self-installed devices.

Watch www.echelon.com/NodeBuilder for updates on a new technical paper, discussing these issues in much more detail.

Thank you.