# Our Current Focus:

## Synthesis of formal models from the ADSL

Important features of the models we're building:

- Efficient – they have a small number of state variables
- Generic – they use the same model framework for synchronous, quasi-synchronous, and asynchronous systems
- Automatic – model generation and verification is as automatic as possible

# Progress Since October

- Updated OM(1) model:
  - complete byzantine faults model
  - auxiliary system abstraction

- Fully inductive proof of *validity* under suitable maximum fault assumption
  - inductive proof is significantly faster to check
  - scales significantly better

# Current OM(1) Model

**Summary** of our OM(1) model:

- an ideal "model" of the models we wish to synthesize from the ADSL
- quasi-synchronous approximation of the classical synchronous OM(1)
- built on a general and extensible timing and message passing framework (based on calendar automata)

➡ incorporates a *fault model* which is compositional rather than baked in to the node behavior

➡ contains an *inductive proof* of key properties

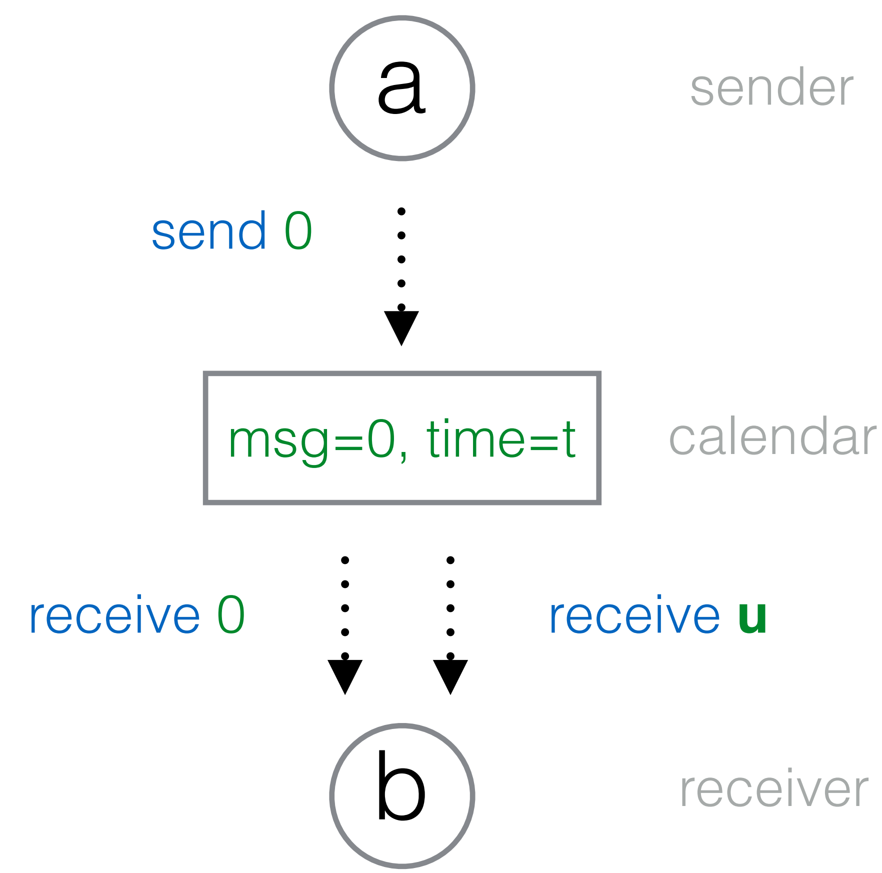
# Fault Model

- We have implemented a *Byzantine fault model* on top of the calendar automata
- It can easily be extended to a *hybrid fault model* including manifest and symmetric faults
- Messages are modeled by non-negative integers
- Each node is non-deterministically initialized faulty or not

| | |
|---|---|
| -1 | missing |
| 0 | good |
| >0 | faulty |

# Fault Model

- When a node sends a message, it appear on the calendar as normal
- When a message is read, the result is either:
    - (sender is not faulty) the intended message
    - (sender is faulty) uninterpreted constant of type Message
- A-priori, the number of distinct messages generated in a trace is ***unbounded***




**u** is a non-negative integer that we know nothing about
**u** is a (potentially) different constant on each receive

# Fault Model

- Faults are manifested through reading from the global calendar
- Model nodes can be synthesized without any awareness of the fault model and don't need to be changed if the fault model changes
- The extent and type of faults is easily controlled from a single place in the model
- ... but use of <u>uninterpreted constants</u> limits our choice of model checking tools

# Verification

- For infinite models, we cannot use symbolic (BDD) model checking, nor ordinary bounded model checking
- *Infinite bounded model checking* is an SMT-based approach to model checking infinite systems
  - It is supported by SAL (`sal-inf-bmc`)
  - The main technique used is **induction:**
    - user provides inductive invariants of the system
    - model checker tries to prove a given property by induction using the invariants
    - ... or else provides a counter-example trace.
  - Proof by induction generally requires more work on the user's side (providing invariants)
  - Proof by induction is generally <u>much faster</u> and <u>scalable</u>

# Proof by Induction

- We have completed a proof by induction of the validity property for OM(1)
  - Many auxiliary lemmas were required to complete the proof – they were hand generated
  - A main goal of the proof's construction was to separate out those auxiliary lemmas that we believe can be <u>generated automatically</u> in our translation
- As a side benefit, we have a proof which is much faster to verify and which scales much better than symbolic or ordinary bounded model checking

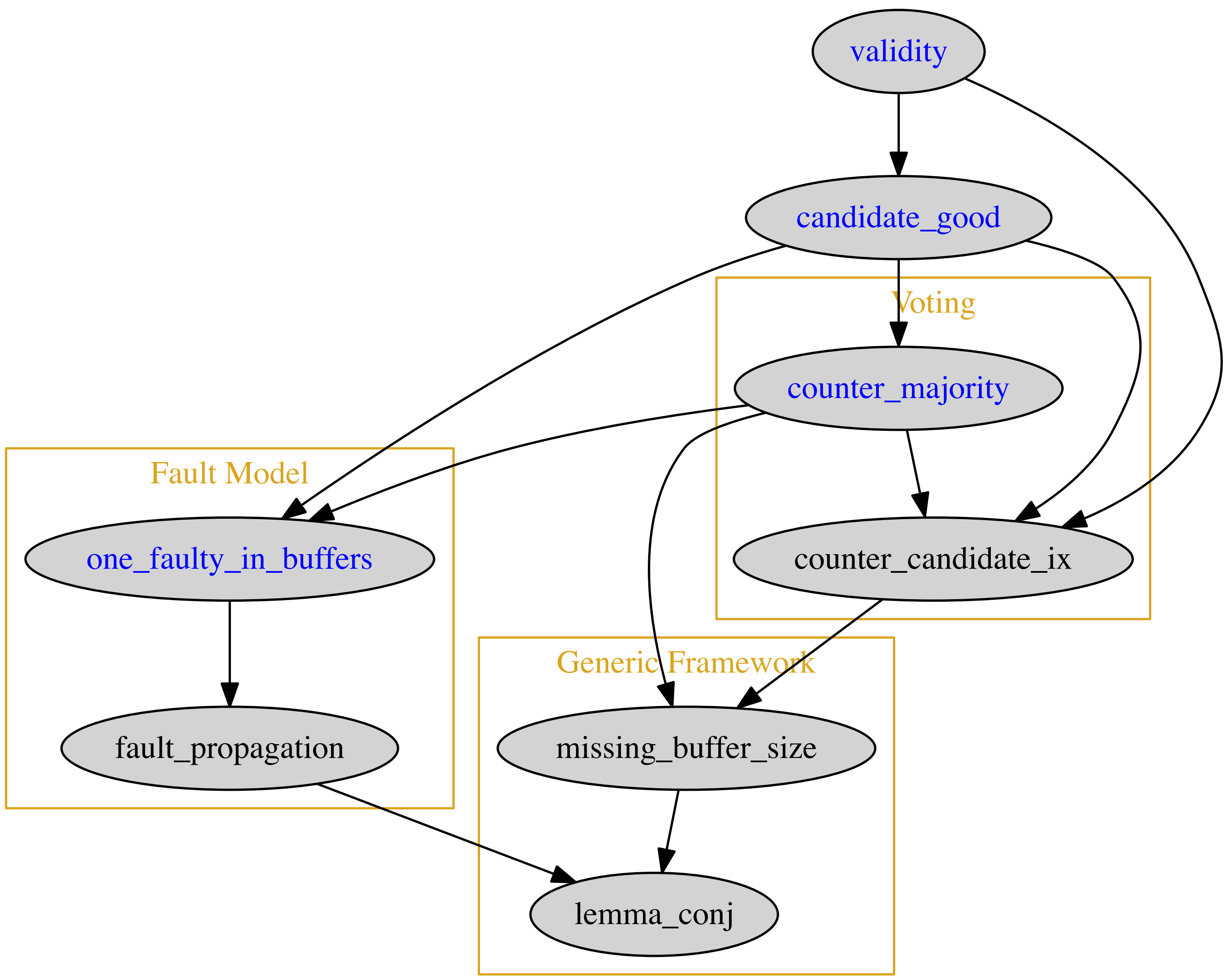
# Lemma structure

- `lemma_conj` : basic framework lemmas, e.g. "*time is always non-negative and increases monotonically*"

- `fault_propagation` : "*if a faulty message is received, then one of the nodes upstream must be faulty*"

- `one_faulty_in_buffers` : "under the maximum fault assumption, each receiver buffer contains at most one faulty value"

- `counter_candidate_ix` : a disjunctive invariant of the voting procedure




blue lemmas are specific to the property (validity) being proved

# Model Checking

- Model checking time is greatly reduced using induction
- Human time is greatly increased, however
- Proof-by-induction has worst-case exponential complexity in N, the number of system variables
- Ordinary bounded model checking and symbolic model checking have worst-case double-exponential complexity in N

| **v1** | **v4** | **v2016-01**<br>**current** |
|---|---|---|
| > 7200 s | 91 s | 11 s |

Time for verification

(Intel Xeon E5-2620 v2 @ 2.10GHz)